CLID: A general approach to validate security policies in a dynamic network

Yanyan Yang, Charles U. Martel, S. Felix Wu
Department of Computer Science
University of California, Davis
{yangyan, martel, wu}@cs.ucdavis.edu

Abstract - Many researchers have considered security policy management, including how to configure policies manually and even how to automatically generate security policies based on security requirements. Both can be error prone, especially when properties of the network topology change, because security requirements are usually not bound to any particular route path. Our DETER lab emulation results show that conflicts could be caused by these factors. Therefore, a systematic way to validate the correctness of the security policies is essential. This paper presents an approach, CLID (Conflict and Looping Identification and Detection), to verify whether a set of security policies (e.g. IPSec/VPN tunnels) satisfy the given security requirements, without causing any conflicts. This approach utilizes the definition of a security policy to include network routing data as well as traffic selector information, thus it works for general network topologies. We also analyze and justify the correctness of the approach. The paper concludes with our simulation results and addresses future work.

I. INTRODUCTION

Network security is becoming indispensable, but new vulnerabilities emerge every day. While IPSec [6][7][8] is a useful IP layer security protocol which can provide authentication and encryption for end-to-end traffic flow, configuring IPSec VPN tunnels is notoriously complicated because it has so many options (key exchange, ciphers, authentication etc) to configure. Thus the ultimate solutions to the security requirements are often prone to errors.

How to easily, efficiently and correctly manage security policies (tunnels) is a popular topic in the security policy management [9][10][11][12]. Although the original requirements are not bound to any particular route, as demonstrated in this paper later, the correct configuration of IPSec VPN policies MUST consider its active interaction with Internet routing dynamics. In other words, the security policies together with the validation algorithm must utilize routing information in order to resolve potential complex policy conflicts. These may be due to routing dynamics or even network mobility, such as the issues being investigated in the NEMO, Network Mobility, working group [13][14] under IETF.

In our previous papers [1][3], we focused on the correct policy configuration under the scope of "linear routing paths", i.e., within a single linear path, we can guarantee that no conflicts occur among the IPSec tunnel policies for the routers and traffic on this linear path. For instance, in [1], we proposed an optimal solution for handling overlapping tunnels for a linear routing topology. This paper considers much more general settings, and we will demonstrate later the details and examples of general dynamic network conflicts, and then show how to detect such conflicts.

Policy configuration mistakes can sometimes be caused by human carelessness, while dynamic routing changes, either expected or unexpected, can also cause problems in delivering the packet. One minor mistake or one subtle change (e.g. in routing) can cause insecure message transmission or even packet looping. Furthermore, the more complex a network topology is, the more security requirements and corresponding security policies that need to be enforced to satisfy the requirements. However, the subsequent issues of security breaches could be caused by more interactions among security policies and more interactions between security policies and routing policies.

Figure 1 and Figure 3 illustrate these security issues, respectively.

- Routing interference causing security violations

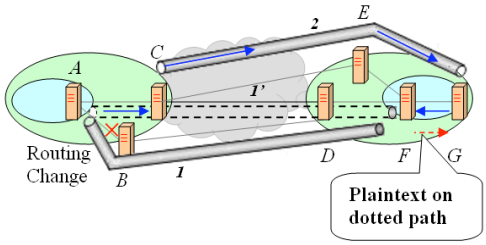

Figure 1 Tunnel Overlap due to routing interference

When tunnels are configured and built purely according to the basic security requirements without considering network topology and routing information, and when two or more tunnels share common routers, it is easy to create overlapping tunnels unintentionally. This can potentially cause security conflicts that violate the original requirements.

In Figure 1, we have two security requirements that (*Req1*) require traffic from source *A* to destination *F* or *G* to be encrypted when routed from *A* to *F* and (*Req2*) traffic from source *A* or *B* to *F* or *G* to be encrypted when routed from *C* to *G*. Thus we build IPSec tunnels to cover these security requirements. Traffic flow sent from *A* to *F* or *G* are encrypted through tunnel 1 across routers $A \Rightarrow B \Rightarrow D \Rightarrow F$, and traffic flow which reaches *C* while being sent from *A* or *B* to *F* or *G* are encrypted through tunnel 2 across routers $C \Rightarrow E \Rightarrow F \Rightarrow G$. The security policies are valid until a routing change occurs. The link between *A* and *B* goes down, so now

every packet goes through the other link from *A* to *C*. Please note that now we have two different route paths: *A*⇒*C*⇒*D*⇒*F* and *C*⇒*E*⇒*F*⇒*G*, and they are NOT on a single linear route path. When a traffic flow is being sent from *A* to *G*, it used to enter tunnel 1 to reach *G*. Now it first enters tunnel 1' so will follow different routers. When it gets to *C* due to the routing change, it will then enter tunnel 2. After it gets to *G*, it will be decapsulated and decrypted. Now *G* sees its destination as *F* in its outer IP header added to the original packet by tunnel 1. So the traffic is sent to *F*. At *F*, it will be decapsulated again. Now *F* sees the original packet and sends it to *G* in plaintext, which violates *Req2*.

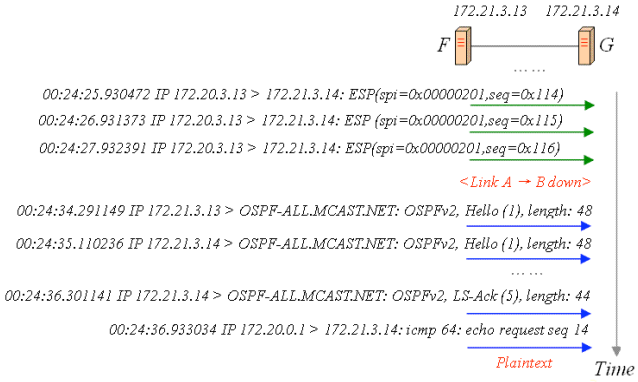

Figure 2 DETER lab emulation results of example 1

We have emulated this example at DETER [17] lab test bed using OSPF as the intra-domain routing protocol and BGP as the inter-domain protocol. The results show in Figure 2 that it would only take a few seconds for the hacker to eavesdrop every single packet from A to G at the link from F to G. In the figure, we show the traffic flow from F to G. It is seen that the traffic was sent with encryption until the link from A to B is down. Then after the routing gets updated along the network (OSPF update), the traffic gets redirected and thus sent in plaintext. This particular example shows that even if the current topology is known, changes in the route path may create security violations (e.g. a traffic flow that is required to be encrypted along its routing path is sent in plaintext). Note that 172.20.3.13 is one of the interfaces at router C, while 172.20.0.1 is one of the interfaces at router A.

This particular example also shows that creating security policies without knowledge of the network topology (including how other tunnels are built) and routing may generate security violations.

Therefore, we need a good way to identify potential security violations, so we observe that the following two conditions are necessary for problems to occur:

1) Redirection:

A packet enters one tunnel, and within this tunnel enters a second tunnel. Thus the packet gets temporarily sent to a new destination (i.e. with its new outer IP header including the end of the second tunnel as its destination).

2) Decapsulation and Forwarding (DF):

A packet reaches the end of a tunnel, gets decapsulated, and then discovers it has a new destination (in its inner IP header). Thus the packet is forwarded to the real destination.

Basically, the redirection happens when two tunnels (i.e. policies) interact with each other, while DF occurs when the real destination is discovered. When both conditions occur, the packet may be sent in plaintext or un-authenticated. Although redirection and DF are necessary to cause conflicts for most cases, it is also noted that when shadowed tunnels are present, security conflicts may occur without redirection and DF (details can be referred in Section III.D).

- Routing interference causing packet looping

When tunnels are configured with incongruous options (e.g. traffic selectors), the traffic flow can be routed in a way that causes another type of problem – packet looping. Packet looping can occur even on a single router with two different outgoing network interfaces. For instance, we have configured IPSec policies on a Windows 2000 box such that one outgoing interface always forwards the packet to the other interface after the IPSec outbound processing, and vice versa. Therefore, even the IKE traffic can not go through correctly.

With a few more routers and with routing dynamics, the situation is even more complicated. Sometimes, during the phase of IKE processing, the loop didn't exist, but, after the tunnels were built, some new routing dynamics might introduce unexpected packet looping. For instance, in Figure 3, two encryption policies are created to build IPSec tunnels to cover the security requirements, where traffic flow sent from *B* to *D* is encrypted and traffic flow sent from *C* to *A* is also encrypted. Tunnel 1 is configured to have traffic selector from {*B*, *C*} to {*A*, *D*, *E*}, while tunnel 2 is configured to have traffic selector from {*A*, *B*, *C*} to {*D*, *E*}. When a traffic flow is sent from *B* to *D*, it first enters tunnel 1 at *B*. However, when it travels to *C*, the routing table reroutes the flow to *E* and thus it enters tunnel 2, it will be encrypted to enter tunnel 2 and thus forwarded to *E* with the IP header from *C* to *A*. *E* gets the packet and directly forwards it to *B*. Now again *B* sees the packet header with the current header from *C* to *A*, so, it will again encrypt and encapsulate the packet and then resends it through tunnel 1. *C* gets the packet with header from *B* to *D*. It will again encrypt, encapsulate the packet and sends it to tunnel 2.

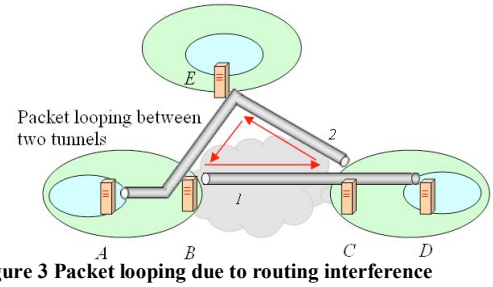

Figure 3 Packet looping due to routing interference

Therefore, in this scenario, as the tunnels are inappropriately configured, the packet is repeatedly encrypted and encapsulated for the two tunnels without ever reaching its destination.

Figure 4 DETER lab emulation results of example 2

This typical example was also emulated at DETER lab test bed. In the figure, we show the traffic flow from B to C. It is seen that the traffic was sent with encryption with packet length growing gradually (a new layer of packet header is added when entering a tunnel). The results show that the data packets get encrypted and encapsulated repeatedly and finally exceed the Maximum Transfer Unit (MTU), which causes the packets be dropped by the network eventually and thus cause the data transmission fail.

Our previous work [1][3] tried to solve the problem by automatically generating the policies based on a set of security requirements. However, while the security requirements are not and should not be bound to a particular route path due to the dynamic routing aspect of current networks, the policy generation and validation must depend on exact route paths as well as their potential dynamics. Both examples above show that the tunnels may satisfy the requirements for a particular routing topology at a particular period of time. But, a general approach, as proposed in this paper, is necessary to guarantee the correctness of IPSec/VPN configuration for all cases. Furthermore, in order to consider all possible general network topologies, we re-define the security policies (i.e. tunnels) to include routing information. We have evaluated our proposed solution via a simulation study with thousands of randomly generated security requirements. Our results indicate that the performance of the proposed scheme is quite reasonable and without false negatives under our conflict definitions, while it yields a small number of false positives (i.e. no conflict but all conflict detection conditions met) in some rare cases as described in Section III.D.

The rest of the paper is illustrated as follows. Section II gives a brief overview of the related work, including our previous work. Then we define the problem and introduce our algorithm along with the justification for its correctness in Section III. Section IV presents our simulation results and analysis. Finally, we conclude our paper and outline future work.

II. RELATED WORK

As we addressed in the earlier section, much research has focused on unilateral policy management by assuming a particular network topology and neglecting the routing impact in the network. Thus the approaches derived from these assumptions may miss configuration errors when the network topology becomes more complicated and when dynamic routing changes the topology.

A. Bundle and Direct approach

[3] and [4] introduced two approaches to automatically generate security policies as well as to ensure the correctness of the policy solutions. Bundle approach builds tunnels for disjoint traffic flows, which are separated from the overall traffic. It guarantees to satisfy the security requirements as it builds policies for each network flow, i.e. bundle, however, it may generate a lot of redundant tunnels for different bundles but for the same network region. On the other hand, Direct approach creates policies for each security requirement and thus achieves better performance (fewer tunnels), but again it may create redundant policies (e.g. as some requirements may be redundant) and the generated policies may have security conflicts and violations among each other.

B. BANDS and Ordered-Split algorithm

Among our previous work, Ordered-Split algorithm [1] focuses on a linear topology and finds a correct policy solution with the minimum number of tunnels. In contrast, BANDS [2] is an inter-domain security policy management system, which finds out the route path first and then computes the final policy solutions using Direct approach for the security requirements along the path. BANDS runs on a general network topology but doesn't consider routing interference that may also impact the final policy solutions.

C. Other security policy management research

[16] is an extension work of the Ordered-Split algorithm that analyzes the traffic probability together with the original algorithm to optimize the solution, while [15] proposes a Zero-Conflict algorithm that yields a faster time complexity using the concepts of requirement groups and cut points. Other research focuses on the policy management alone. [10] demonstrated an algorithm for distributing policies among a number of management stations, while [9] discussed an approach to conflict handling relying on a priori models. These two papers focus on what needs to be done for general policy management and policy-based network management, yet not specific to IPSec/VPN tunnel management as we focus here.

III. A GENERAL ALGORITHM: CLID

In this section, we first describe the problem definition and detail a general validation algorithm (CLID) to apply to any network topologies. Note that the tunnel definition includes the sequence of the routers it visits, its traffic selector and the routing information of the routers in the tunnel. The former reflects the routing information, while the latter specifies one of the pre-conditions for the traffic flows to go through the tunnel. Therefore, if the routing changes, the sequence of the routers in a tunnel may change accordingly. CLID algorithm never assumes any particular fixed route but only on any policy data (i.e. any route for any tunnels) that are collected when validation is performed.

A. Problem definition

Given a set of security requirements and a set of tunnels (where each tunnel is described by a sequence of routers visited, its security function and the associated traffic selector), determine if the set of tunnels correctly achieves these security requirements (correctness is e.g. if a packet is required to go from A→B encrypted, it should never be in plain text along the way). Below we define requirements, tunnels and conflicts.

- Security requirements

A security requirement defines how a network flow should be protected during the transmission. In this paper, a security requirement is defined as:

<selector> X: i → j (auth/enc) [trusted: a, b, c, …]

Where *i* and *j* are both routers. This requirement specifies for all the traffic *X* (based on traffic selector[1] that can be represented as <*src, dst*>) being forwarded by router *i*, if the traffic will eventually be routed to router *j*, then regardless

[1] A traffic selector can be a 5-tuple including the source IP address, source port, destination IP address, destination port and protocol in the IP header. For simplicity, we will only use the IP address pair as the selector.

which route path it will take from *i* to *j*, all the information bits must be authenticated/encrypted from *i* to *j*, except at trusted nodes *a*, *b*, *c* etc. The traffic selector can be either the original source and destination of the packet, or any intermediate ones that get prepended to the IP header due to encapsulation. Note that *src* and *dst* can be a set of possible sources/destinations. We include two end point routers (*i* and *j*) in the requirement definition as there are some cases that traffic flows are required to be encrypted between two subnets or two intranets: router *i* and *j* are usually the edge router of the networks. How the data are routed and handled between these routers is transparent and dynamic to the requirements.

For instance,

$$req = <\{1\}, \{7,10\}>: 1 \rightarrow 7 \text{ (enc) } [trusted: 3, 5]$$

means that for all the traffic flows that start at router 1 and go to router 7 or 10 (i.e. the selector, which is basically the source and destination in the IP header), if the traffic will be routed from 1 to 7 it is required to be encrypted from 1 to 7, except at trusted 3 and 5.

- Tunnel definition:

A tunnel is represented by a sequence of routers that it visits along the route path.

$$\mathbf{<src, dst> [(r_i, r_{i+1}, \ldots, r_n), ENC/AUTH]}$$

Which describes that the tunnel is built from router r_i to router r_n with encryption (or authentication). Any traffic whose source starts at *src* and ends at *dst* (the source and destination IP addresses in the IP header) is sent through the tunnel, where r_i, r_{i+1}, …, r_n are a sequence of the routers that the tunnel visits. Note that src and dst can be a set of possible sources/destinations.

For instance,

$$tunnel = <\{1\}, \{7, 10\}>[(1, 2, 4, 7), ENC]$$

means that for all of the traffic flows that start at 1 and end at 7 or 10, they will be encrypted in the tunnel with IPSec SA[2] from router 1 to 7. It also indicates that the route path for the tunnel is 1 → 2 → 4 → 7.

- Security conflict definition

We define security conflicts as:

1. Requirement violations that break any of the original security requirements, which include that no plaintext is sent along the route path from router *i* to router *j* (except at the trusted nodes), if the traffic flow required to be encrypted is routed from router *i* to router *j*, or;
2. Packet looping among tunnels, where traffic flow is being repeatedly sent among the tunnels without reaching its destination.

For packet looping conflicts, note that it may occur in other ways (e.g. pure routing looping), but that is beyond the scope of this paper.

B. Validation algorithm

B.1. Algorithm steps:

[2] A Security Association (SA) is a set of security information that specifies a particular kind of secure connection between one device and another.

Step 1. Tunnels for the same traffic selector are sorted by the order they are enforced in a local PEP (Policy Enforcement Point). This arranges the tunnels in the way that the traffic would select accordingly.

Step 2. The algorithm then verifies if the tunnel set covers the security requirements. For each requirement, if there exists a tunnel or a chain of tunnels to cover it by providing the specified requirement, while only doing encryption and decryption operations at trusted nodes, it satisfies the requirement (if no two tunnels conflict, the requirements will be satisfied by this test).

Step 3. Then we check if for any pair of tunnel definitions the router sequences of the two tunnels have one or more common routers.

Step 4. If the two tunnels intersect with one or more routers (**pre-condition**), we need to check further whether one tunnel will fit the traffic selector of the other tunnel. Having the overlap may cause a security conflict but also may not be a problem, as illustrated in the following example.

$$req_1 = <\{1\}, \{7\}>: 2 \rightarrow 7 \text{ (enc) } [trusted: \varnothing]$$

$$req_2 = <\{2, 4\}, \{7\}>: 4 \rightarrow 5 \text{ (enc)} [trusted: \varnothing]$$

$$tunnel_1 = <\{1\}, \{7\}> [(2, 4, 5, 7), ENC]$$

$$tunnel_2 = <\{2, 4\}, \{7\}> [(4, 6, 5), ENC]$$

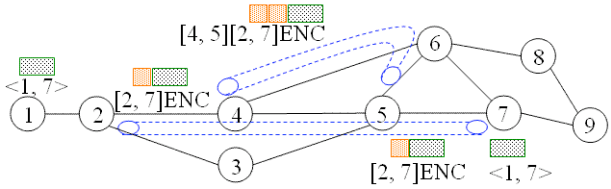

Figure 5 No conflict with intersecting tunnels

A packet is being sent from router 1 to router 7. According to the selectors, it will enter $tunnel_1$ at router 2 and then enter $tunnel_2$ where an outer layer will be encrypted and encapsulated. Although two tunnels intersect with two routers: 4 and 5, the outer part of the encrypted traffic in $tunnel_2$ will be decrypted and decapsulated at 5. It then will be forwarded to 7, then it remains encrypted until it is decrypted and decapsulated at router 7. Note that if $tunnel_2$ had source selector {3,4} instead of {2,4}, messages in tunnel1 would not enter $tunnel_2$ at router 4. This is formalized in Condition I a) below.

Step 5. If pre-condition (tunnel t_i and tunnel t_j intersect with one or more routers) and the following three conditions are met, **the tunnels violate** the given security requirements in a way that traffic may be sent in plaintext somewhere along the route path. When Condition I a) below is met, we say that the two tunnels ***overlap***.

- Condition I:
  a) t_j's start router s_j is an internal router in t_i (so is neither the first nor last router of t_i) and t_i's start router s_i is in the source set of t_j's selector and t_i's end router e_i is in the destination set of t_j's selector;
  b) t_j's end router e_j is not an internal router in t_i.
- Condition II: The destination in the traffic selector of tunnel t_i includes routers other than e_i;

- Condition III: Define src = t_i.src ∩ t_j.src and dst = t_i.dst ∩ t_j.dst. If both src and dst are not empty and there is no tunnel or a chain of tunnels that covers the route from e_i to e_j, for traffic from source selector *s* to destination selector *d* for some pair of routers *s* in src and *d* in dst.

Consider the following example, where two tunnels are built to meet the requirements that specify to encrypt data from router 1 to router 7 or 9, and from router 2 or 4 to router 7 or 9. As we can see, each tunnel alone seems satisfy its requirement.

req_1 = <{1}, {7, 9}>: 2 → 7 (enc) [trusted:∅]

req_2 = <{1, 2, 4},{ 7, 9}>: 4 → 9 (enc)[trusted: ∅]

$tunnel_1$ = <{1}, {7, 9}> [(2, 4, 5, 7), ENC]

$tunnel_2$ =<{1, 2, 4}, {7, 9}> [(4, 6, 8, 9), ENC]

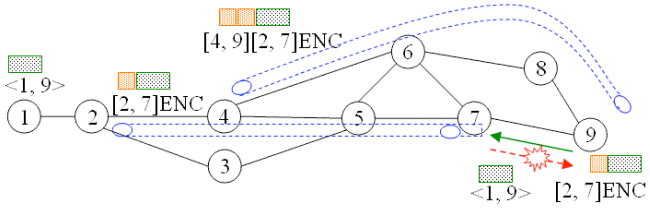

Figure 6 Router intersection causing conflict

These tunnels overlap by intersecting with route 4, where $tunnel_1$'s encapsulated traffic is re-directed into $tunnel_2$. Suppose a packet is being sent from router 1 to router 9. It will be encrypted and encapsulated into the tunnel from router 2 to router 7. When it arrives at router 4, it will be encrypted and encapsulated again so it will be re-directed into second tunnel from router 4 to router 9. The packet will then be sent along the router path, i.e. router 4 → 6 → 8 → 9. Once it hits router 9, it will be decapsulated and decrypted and now router 9 sees the IP header with destination router 7 (for the first tunnel). Router 9 will then send the packet to router 7. After Router 7 decapsulates and decrypts the packet, it finds that the packet's destination is router 9 and then sends the packet to router 9. This violates req_2, which requires the traffic that fits the selectors and routed from 4 to 9 be encrypted until it visits router 9 its final time. These two tunnels conflict as all the conditions are met:

- Condition I: a) the tunnels intersect with router 4, the start router of $tunnel_2$, and $tunnel_2$'s end router 9 is not an internal router in $tunnel_1$;
- Condition I: b) $tunnel_1$'s start router 2 is in the source set of $tunnel_2$'s selector and $tunnel_1$'s end router 7 is in the destination set of $tunnel_2$;
- Condition II: the destination in the traffic selector of $tunnel_1$ includes router 7 and 9;
- Condition III: src = {1}, dst = {7, 9}, there is no tunnel from router 7 to router 9.

In other words, when a re-direction happens and the packet travels back and forth from the end router of the previous tunnel and the end router of the latter tunnel without having security enforced, it violates the requirement.

However, if the dynamic routing changes the $tunnel_1$ to:

$tunnel_1$ = <{1}, {7, 9}> [(2, 3, 5, 7), ENC]

There is no intersection between $tunnel_1$ and $tunnel_2$, thus the security conflict problem will be eliminated.

In the scenario with the packet sent from router 1 to 9, the packet will be encrypted and encapsulated to enter $tunnel_1$ and be routed to 2→3→5→7. And $tunnel_2$ will no longer be entered with the new routing.

Generally, pre-condition and condition I ensures that two tunnels "overlap", while condition II defines that the traffic selector parts of the two tunnels muddle up. In other words, these two tunnels and their traffic selectors overlap and thus the security protection may be broken on the later part after the last common router on the path. Note that the conditions in CLID algorithm do not test shadowed tunnels (please refer to Section III.D for more details).

Step 6. If tunnels t_1, t_2, …, t_n are such that Condition I and the following condition are met, **the tunnels cause** packets **looping**.

- Condition IV:

  t_1 and t_2 overlap

  t_2 and t_3 overlap

  …

  t_n and t_1 overlap

  Where t_1, t_2, t_3, … , t_n are tunnels.

Given the following example,

$tunnel_1$ = <{1, 2}, {3, 6}> [(1, 2, 3), *ENC*]

$tunnel_2$ = <{1}, {3, 6}> [(2, 4, 1, 6), *ENC*]

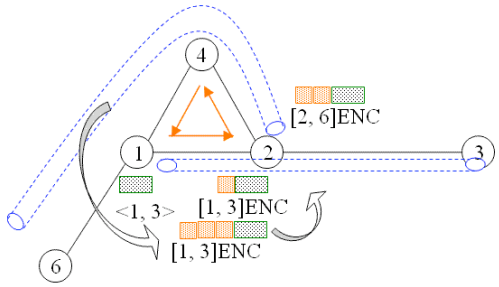

Figure 7 Packet looping by routing interference

A packet is being sent from router 1 to router 3. First it will be encrypted and encapsulated in $Tunnel_1$. When it hits router 2, it will be again encrypted and encapsulated in $Tunnel_2$. According to the tunnel definitions, the packet will be routed to route 4 and then router 1. The encrypted packet will now enter a loop and thus be repeatedly encapsulated for the two tunnels without ever reaching its destination.

One way to solve the packet looping problem is as a graph problem. Tunnels are nodes in the auxiliary graph. There is an arc from $node_1$ (for $tunnel_1$) to $node_2$ (for $tunnel_2$) if $tunnel_1$ and $tunnel_2$ overlap (meet Condition I a)). Looping exists if there is a directed cycle in the graph.

The packet looping definition specifies the packet circles

among the tunnels back and forth. However, sometimes a packet with source IP *i* can be looping back to the tunnel that starts at *i*, in certain scenarios as follows.

tunnel1 = <{1, 2}, {3, 5}> [1, 2, 4, 5] ENC

tunnel2 = <{{1, 2}, {5}> [2, 3] ENC

With the two tunnel definitions above, a packet with router 1 as source and router 5 as destination will first enter $tunnel_1$ at router 1 and then enter $tunnel_2$ at router 2. When the packet is decrypted and decapsulated at router 3, according to the current routing table and the traffic selector in the IP header (1 as source and 5 as destination), the packet will be re-routed back to router 1.

With the definition of $tunnel_1$, theoretically, it might enter $tunnel_1$ again to cause looping. However, most of the security routers will consider this as IP Spoofing and thus drop this packet, when they receive a packet with its own source IP in the header. Therefore, our validation algorithm will not cover this.

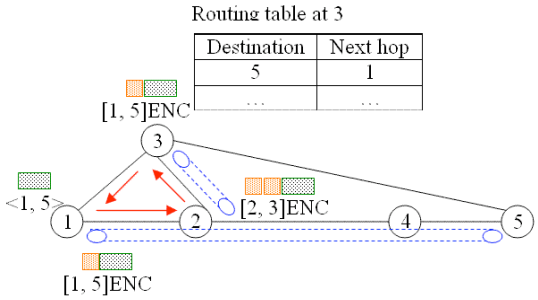

Figure 8 Packet looping considered as IP spoofing

C. *Examples*

✓ Security requirements:

REQ#1: <{1}, {7, 10}>1→7(ENC)[TRUSTED: 3]
REQ#2: <{1, 2}, {7,10}2→10(AUTH)[TRUSTED:7]
REQ#3: <DOMAIN A, {6, 8, 9}>2→6(ENC)[TRUSTED:∅]
REQ#4: <{1,2}, {6, 8}>3→8(ENC)[TRUSTED:6]

REQ#1 requires that any packet that starts at router 1 to destination router 7 or 10 needs to have data encryption enforced while going from 1 to 7. Router 3 is trusted.

REQ#2 requires that any packet that starts at router 1 or 2 to destination router 7 or 10 needs to have data authentication enforced while going from 2 to 10. Router 7 is trusted.

REQ#3 requires that any packet that starts at any router in Domain A to destination router 6 or 8 or 9 needs to have authentication enforced while going from 2 to 6. Domain A is defined as {1, 2, 3}.

REQ#4 requires that any packet that starts at router 1 or 2 to destination router 6 or 8 needs to have data encryption enforced while going from 3 to 8. Router 6 is trusted.

✓ Tunnel definitions:

According to the security requirements defined above, a naïve network security administrator may manually build the following four tunnels, i.e. one tunnel for each requirement (see Figure 9).

In the tunnel definitions above, each tunnel is a sequence of routers that the tunnel visits, where:

a. Tunnel 1 encrypts the traffic (whose source is router 1 and destination is router 7 or 10) and travels across router 1, 2, 4, and 7.

b. Tunnel 2 authenticates the traffic (whose source is router 1 or 2 and destination is router 7 or 10) and travels across router 2, 5, 7 and 10.

c. Tunnel 3 encrypts the traffic (whose source is any router in Domain A and destination is router 6 or 8 or 9) and travels across router 2, 3 and 6.

d. Tunnel 4 encrypts the traffic (whose source is router 1 or 2 and destination is router 6 or 8) and travels across router 3, 6 and 8.

Tunnels	Sequence of Routers	Traffic Selector
Tunnel 1	(1, 2, 4, 7)ENC	src = 1, dst = {7, 10}
Tunnel 2	(2, 5, 7, 10)AUTH	src = {1, 2}, dst = {7, 10}
Tunnel 3	(2, 3, 6)ENC	src = Domain A, dst = {6, 8, 9}
Tunnel 4	(3, 6, 8)ENC	src = {1,2}, dst = {6, 8}

Apparently, if we look at each tunnel individually, it satisfies its requirement. However, when we put all the tunnels together, we may experience some security violations as described below. Note that below, T_intersection contains the routers which occur in both of two tunnels.

✓ Example 1: for Tunnel 1 and Tunnel 2

T_intersection = {2, 7}

The start router of Tunnel 2 is an internal router of Tunnel 1 and the end router of Tunnel 2 is not an internal router of Tunnel 1. The start router and end router of Tunnel 1 are 1 and 7. They are in source set and destination set of Tunnel 2's traffic selector, respectively. And src = {1}, dst = {7, 10}. There is no tunnel from router 7 to 10 for traffic <src, dst>, where src = {1}, dst = {10}.

Suppose that a packet is being sent from router 1 to router 10. According to the traffic selector at router 1, the packet is encapsulated and encrypted through a tunnel from router 1 to router 7. When it arrives at router 2, according to the traffic selectors at router 2, it is encapsulated through an authentication tunnel from router 2 to router 10. No matter how the packet is being routed, it will be de-tunneled at router 10 first and then be routed back to router 7. After the packet gets decrypted at router 7, it is being sent from router 7 to router 10 in plaintext without authentication enforced. In short, the packet is re-directed to the tunnel path of a previous tunnel. Therefore this violates the original security requirement REQ#2 that requires authentication encryption for traffic from 2 to 10.

✓ Example 2: for Tunnel 1 and Tunnel 3

T_intersection = {2}

The start router and end router of Tunnel 1 are 1 and 7. Since 7 is not in destination set of Tunnel 3's traffic selector, respectively. 1 and Tunnel 3 do not overlap and there is no conflict between them.

✓ Example 3: for Tunnel 1 and Tunnel 4

T_intersection = ∅

T_intersection is empty, therefore Tunnel 1 and Tunnel 4 do not conflict.

✓ Example 4: for Tunnel 2 and Tunnel 3

T_intersection = {2}

The start router and end router of Tunnel 2 are 2 and 10. They are neither in source set nor in destination set of Tunnel 3's traffic selector, respectively. Therefore Tunnel 2 and Tunnel 3 do not overlap/conflict.

✓ Example 5: for Tunnel 2 and Tunnel 4

T_intersection = ∅

T_intersection is empty, therefore Tunnel 2 and Tunnel 4 do not conflict.

✓ Example 6: for Tunnel 3 and Tunnel 4

T_intersection = {3, 6}

The start router of Tunnel 4 is an internal router of Tunnel 3 and the end router of Tunnel 4 is not an internal router of Tunnel 3. The start router and end router of Tunnel 3 are 2 and 6. They are in source set and destination set of Tunnel 4's traffic selector, respectively. And src = {1,2}, dst = {6, 8}. There is no tunnel from router 6 to 8 for traffic <src, dst>, where src = {1,2}, dst = {8}.

Suppose that a packet is being sent from router 2 to router 8. According to the traffic selector at router 2, the packet is encapsulated and encrypted through Tunnel 3 from router 2 to router 6. When it arrives at router 3, according to the traffic selectors at router 3, it is encapsulated through Tunnel 4 from router 3 to router 8. No matter how the packet is being routed, it will be de-tunneled at router 8 first and then be routed back to router 6. After the packet gets decrypted at router 6, it is being sent from router 6 to router 8 in plaintext. Similar to Example 1, the packet is re-directed to the previous tunnel path. Therefore this violates the original security requirement REQ#4 that requires encryption for traffic from router 3 to router 8.

✓ Overall topology

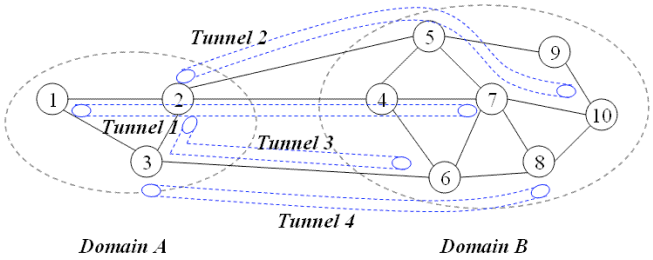

Figure 9 Possible overall topology for the example

Based on the security requirements and tunnel definitions, one of the possible network topologies can be illustrated in Figure 9. In this example, we see that each tunnel itself satisfies the security requirement, however, when they are put together in an inter-domain dynamic topology as follows, security violations may occur as some of them may conflict with each other.

If the tunnels are re-configured and re-built as in Figure 10, the conflicts will be eliminated.

Tunnels	Sequence of Routers	Traffic Selector
Tunnel 1	(1, 2, 4, 7)ENC	src = 1, dst = {7,10}
Tunnel 2	(2, 5, 7)AUTH	src = {1, 2}, dst = {7,10}
Tunnle 2'	{7,10}AUTH	src = {1, 2}, dst = {7,10}
Tunnel 3	(2, 3, 6)ENC	src = Domain A, dst = {6,8,9}
Tunnel 4	(3, 6)ENC	src = {1,2}, dst = {6,8}
Tunnel 4'	(6, 8)ENC	src = {1,2}, dst = {6,8}

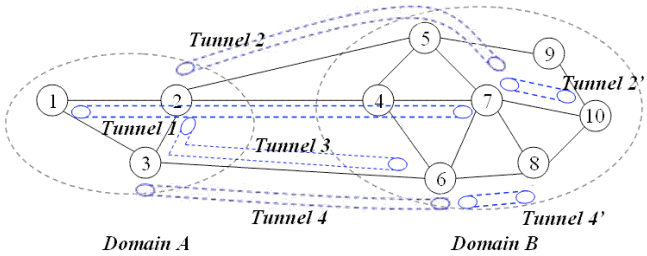

Figure 10 New tunnels without conflicts

For traffic that goes through the original tunnel 2, it now enters the new tunnels (tunnel 2 and tunnel 2') instead. For traffic that goes through the original tunnel 4, it now enters the new tunnels (tunnel 4 and tunnel 4') instead. No re-directions and DF will occur to cause security conflicts.

D. Algorithm Justification

1. Requirement satisfaction:

It is very straightforward that given a security requirement, if there exists a tunnel or a chain of tunnels with the defined security operation (e.g. encryption or authentication) and the corresponding traffic selector, the requirement is satisfied with the tunnel or the chain of tunnels as long as no tunnel overlap occurs;

2. Tunnel Validation:

1) As noted in (i), since each requirement is covered by a tunnel or a chain of tunnels, a problem may occur when a packet is redirected from its original tunnel or the chain of the tunnels. Basically, this may happen when two tunnels overlap.

As IPSec tunnels are usually created in tunnel mode to hide the original packet information and thus to provide better security, typically, a new IP header is inserted in front of the original IP header of the packet by encapsulation. Once the encapsulated packet arrives at other routers, depending on the traffic selectors and routing tables on the routers, this outer IP header could redirect the encapsulated packet into a new and yet unexpected tunnel. Therefore, Condition I tests that it is possible for the packet to be redirected to a different tunnel if the original tunnel overlaps with this new tunnel.

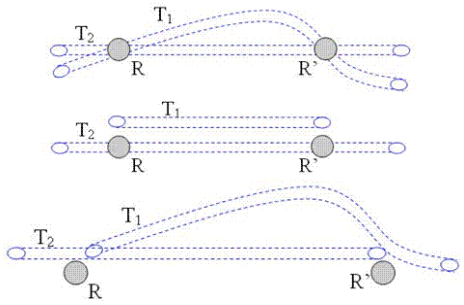

Figure 11 Tunnels that intersect

2) Condition I specifies that the endpoints of one tunnel t_j (e.g. T_1 in Figure 12) has to fit in the traffic selector of the other tunnel t_i (e.g. T_2), so the traffic tunneled in t_j will be re-directed into t_j, as traffic that goes into t_i goes into t_j. Thus the route paths of the two tunnels overlap at these routers. If two tunnels don't overlap, they don't interfere with each other, so no conflicts. This condition also tests if t_j ends inside t_i, which does not cause any violation.

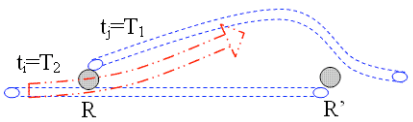

Figure 12 T_2 fits T_1's selectors

3) Condition II implies that traffic that ends at routers other than e_i may also enter t_i. This may cause potential violations as this type of traffic will need to continue after decapsulation at e_i. And it also indicates that there is at least one requirement *Req* for traffic from t_i.src to routers other than e_i otherwise it wouldn't be in the traffic selector when the tunnel is created.

As shown in Figure 13, packets that go into T_2 will go into T_1 (explained above in b), so they will arrive at *R''* first, be decapsulated and decrypted there and then be forwarded back to *R'* because the outer header contains it as the destination. When they get decapsulated and decrypted at *R'*, the real destination of the packets are discovered and the packets are again forwarded to *R''* as the next hop, but in plain-text this time if no tunnel covers it, which violates the original security requirement.

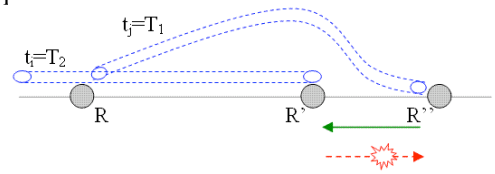

Figure 13 Security violation from *R'* to *R''*

4) Condition III ensures that, since we have *Req* above and there exists an overlap between t_i and t_j, for those traffic (src = t_i.src ∩ t_j.src, dst = t_i.dst ∩ t_j.dst, both src and dst are not empty), once it gets out of tunnel t_i, it is still required in *Req* to be covered from e_i to e_j. If there is a tunnel or chain of tunnels for such traffic, no violation occurs.

5) In order to detect packet looping between tunnels, the algorithm simulates the packet traveling by building a graph, where a node is a tunnel and an arc is for a pair of tunnels among which one fits the other's traffic selector. It is easy to see that looping happens when there is a directed cycle in the graph, i.e. the packet travels by starting at one tunnel and coming back again at this tunnel and so on.

3. Conditions' analysis for conflict detection

In this section, we illustrate how conditions detect violations.

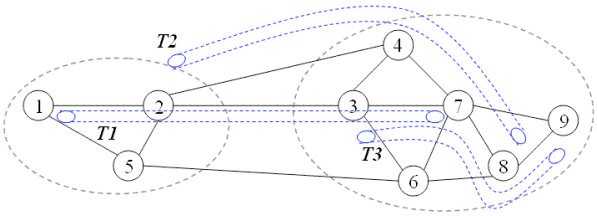

Figure 14 An example for conditions' analysis

In this example, we have three tunnels built for requirements and assume they meet all the requirements:

T1: <{1}, {7,9}>[(1,2,3,7),ENC]

T2: <{1,2}, {7,8,9}>[(2,4,7,8),ENC]

T3: <{2,3}, {8,9}>[(3,7,8,9),ENC]

T1 vs. T2: Condition I and II met; Condition III is met because src={1}, dst={7,9}, e_i=7, e_j=8, no tunnel from 7 to 8 for traffic <{1},{9}>. So packet from 1 to 9 in plaintext from 7 to 8, which violates the requirement that T2 is built for to cover anything sent from {1,2} to {7,8,9} and routed from 2 to 8. Suppose we add one tunnel T4: <(1),{9}>[(7,8), ENC]. Then T1 and T2 are conflict free.

Now we move on to T2 and T3. Condition I and II met; Condition III is met because src={2}, dst={8,9}, e_i=8, e_j=9, no tunnel from 8 to 9 for traffic <{2},{9}>. So packet 2 ⇒ 9 in plaintext from 8 to 9, which violates the requirement that T3 is built for to cover anything sent from {2,3} to {8,9} and routed from 3 to 9. Now suppose we add another tunnel T5: <(2),{9}>[(8,9), ENC]. Then T2 and T3 are conflict free.

T1 vs. T3: Condition I is not met, so no violations

Now we complicate the scenario by altering T3 to make T1 fit its selector:

T3: <{1,2,3}, {7,8,9}>[(3,7,8,9),ENC]

Changing the selector means that traffic from {1,2,3} to {7,8,9} must be encrypted when routed from 3 to 9, which implies as a new requirement R_{new}.

T1 vs. T3: Condition I is met now, as well as Condition II. Condition III is met because src={1}, dst={9}, but we only have T4 to cover from 7 to 8, no tunnel from 8 to 9. So packet sent from 1 to 9 is in clear text when routed from 8 to 9, violating R_{new}.

T1 vs. T2: No violations as no change to them.

T2 vs. T3: Condition I and II met; Condition III is met because src={1,2}, dst={7,8,9}, ei=8, ej=9, we have tunnel from 8 to 9 for traffic <{2},{9}>, but not for traffic <{1,2},{7,9}>. So packet send from 1 to 9 is in clear text when routed from 8 to 9.

Suppose we then make T5: <{1,2},{7,9}>[(8,9), ENC] to fail Condition III. We now re-check T1 vs. T3, T2 vs. T3 and T2 vs. T3. No violations as:

Packet 1⇒9: enters T1, T2, T3, back to 7, enters T4 at 7, and enters T5 at 8;

Packet 2⇒9: enters T2, T3, back to 8, enters T5 at 8;

4. Shadowed tunnels

Consider the following example that has two tunnels built for two requirements in a linear topology where router 1 – 4 are connected in a sequence:

R1: <{1}, (3,4}>: 1→3 (enc)

R2: <{1}, {4}>: 2->4 (enc)

T1: <{1}, {3, 4}> [(1,2,3), ENC]

T2: <{1}, {4}> [(2,3,4), ENC]

For traffic sent from 1 to 4, as it will enter T1 first where it gets encapsulated to have 3 as destination, it will not enter T2. When it gets to 3 and decapsulated, it will be forwarded to 4 in plaintext, which violates R2. In this particular scenario, because tunnel T2 is shadowed by T1, the conflict still occurs even without packet redirection.

Therefore, we define Tunnel t_i and t_j as shadowed tunnels, if they meet the following condition:

1). $src_i \cap src_j \neq \varnothing$

2). $dst_j \cap dst_j \neq \varnothing$

3). $s_i \notin src_j$ or $e_i \notin dst_j$

4). $s_j \in src_i$ and $e_j \notin dst_i$

5). s_j is in tunnel t_i

where s_i is the start router of tunnel t_i and e_i is the end router of tunnel t_i. However, a shadowed tunnel may or may not cause conflicts, e.g. a slight change to R1 and T1 above will still give shadowed tunnels but no conflict:

R1: <{1}, (3,4}>: 1→5 (enc)

T1: <{1}, {3, 4}> [(1,2,5), ENC]

The CLID algorithm can include additional algorithm steps to find potential shadowed tunnels so whoever runs the algorithm will know where to split the tunnel if conflicts do occur.

5. False positives

It is also noted that when all the conditions are met, CLID may cause false positive as no actual violation occurs. Considering the example above with T1, T2 and T3, assume T2 is to cover the following traffic routed from router 2 to 4.

T2: <{1,2}, {7,8,9}>[(2,4), ENC]

Because traffic routed from 1 or 2 to 9 does not visit 4 again after de-tunneled from T1, although T1 and T2 meet all the conditions, they would not cause any conflict, as the traffic reaches both router 4 (with T2) and router 7 (with T1) with encryption enforced.

IV. SIMULATION AND ANALYSIS

In practice, IPSec/VPN policies are deployed from small branch offices to large enterprise buildings to provide interoperable and cryptographically based security for end-to-end traffic flow. This occurs not only in inter-domain environments across the Internet but also in intra-domain environments which can be across the different departments within an organization, as the corporate authentication and confidentiality policies may differ among the departments. Typically, in a university or a large corporation, it may have a tunnel topology similar to the following figure, where tunnels are overlapping and routing may also interfere.

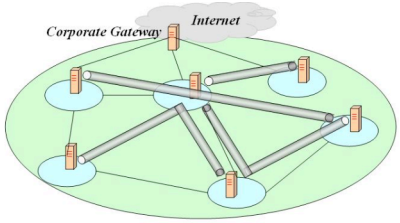

Figure 15 An intra-domain topology example

In the current industrial implementation, a router may be integrated with gateway or firewall as one security device (like security products from Cisco, Juniper or WatchGuard) and IPSec/VPN tunnels are usually built between these security devices. When there are multiple security devices present, a centralized security management system becomes a necessary feature for the advanced security appliances, because it has the capabilities to communicate and interact with each of the security devices within the domain (or even to the other domains). This makes it possible for the centralized management system to collect all the network data, including security requirements, tunnels and routing data. Therefore, the CLID approach can easily be incorporated with a centralized management system to gather information that it needs and then automatically identify and detect security conflicts.

Therefore, we simulate the CLID algorithm using C programming language in Linux. The CLID validation program reads in the requirements and policies, checks for pre-conditions and conditions for conflicts and looping and outputs the validation results. Note that as this is a brute force implementation to check tunnels, it is a preliminary simulation. We ran the simulation test against our earlier example shown in Figure 9 and Figure 10, and then we tested the performance of the simulated algorithm.

A. *Basic simulation test*

From the simulation, we observed that the CLID algorithm can not only identify and detect security conflicts, but also point out where in the network the conflicts could occur. For example, we passed in to the CLID validation program the requirement data as well as the tunnel definitions that defined in earlier examples, it will output if there is any conflict that may occur and also where the plaintext traffic or un-authenticated traffic would appear. For packet looping that occurs among the tunnels, the CLID validation program will also identify and detect what tunnels will cause the looping with a sequence of the routers involved.

```
bash# ./validate req_file_1 pcy_file_1
Security requirements are:
<{1}, {7, 10}> 1 -> 7 | ENC | 3
<{1, 2}, {7, 10}> 2 -> 10 | AUTH | 7
<{1, 2, 3}, {6, 8, 9}> 2 -> 6 | ENC |
<{1}, {6, 8}> 3 -> 8 | ENC | 6

Tunnels are:
Tunnel[0]:      <{1}, {7, 10}> [1 2 4 7 ]| ENC |
Tunnel[1]:      <{1, 2}, {7, 10}> [2 5 7 10 ] | AUTH |
Tunnel[2]:      <{1, 2, 3}, {6, 8, 9}> [2 3 6 ]| ENC |
Tunnel[3]:      <{1}, {6, 8}> [3 6 8 ]| ENC |

+ Tunnel[0] and Tunnel[1] conflict +:
       insecure traffic flow <{1},{7}> between 7 and 10
+ Tunnel[2] and Tunnel[3] conflict +:
       insecure traffic flow <{1},{8}> between 6 and 8
```

```
bash# ./validate req_file_2 pcy_file_2
Security requirements are:
<{1, 2}, {3, 5}> 1 -> 5 | ENC |
<{1, 2}, {5}> 2 -> 3 | ENC |
<{1}, {7}> 2 -> 7 | ENC |
<{2, 4}, {7}> 4 -> 5 | ENC |

Tunnels are:
Tunnel[0]:      <{1, 2}, {3, 5}> [1 2 4 5 ]| ENC |
Tunnel[1]:      <{1, 2}, {5}> [2 3 ]| ENC |
Tunnel[2]:      <{1}, {7}> [2 4 5 7 ]| ENC |
Tunnel[3]:      <{2, 4}, {7}> [4 6 5 ]| ENC |

1 0
==Tunnel[0] causes looping
0 1
==Tunnel[1] causes looping
```

Figure 16 The simulation output for the earlier example

B. *Performance simulation test*

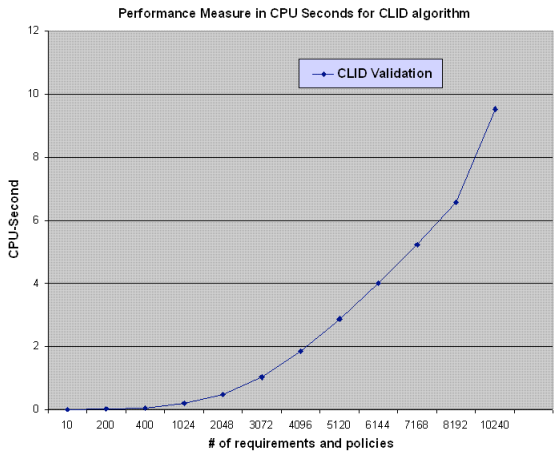

Figure 17 Performance test for CLID algorithm

To monitor the performance of CLID algorithm, we

randomly generated security requirements and policies with the test sizes varying from hundreds to thousands. From the results on the basic simulation, we observe that although it is a preliminary brute force simulation, it takes less than one minute to scan through all the requirements and policies to complete the validation process: check if each requirement is satisfied and then test the conditions for conflicts and looping. Each of the tests prints a result of log for details of the conflicts and looping. Only for the tests that have more than a few thousand requirements and policies, does the CPU time that CLID spends grow sharply.

Usually, for a small or medium-size organization, the number of security requirements and policies should be in reasonable size. While for a large enterprise, which may build branch offices all around the world, the number of requirements and policies might be hundreds or thousands or even more. The simulation tests demonstrate that it would cost a fairly small amount of time for a centralized policy management system to complete the validation using CLID algorithm.

V. CONCLUSIONS AND FUTURE WORK

With the wide deployment of IPSec/VPN tunnels that are to provide security to end-to-end traffic flows, security policy management becomes necessary yet error prone due to mistakes during manual configuration or due to the interactions among policies or due to the routing interference. Basically, there are two necessary conditions for potential conflicts to occur:

- Packet redirection to a new tunnel from the current tunnel.
- Packet decapsulation and forwarding that happens when a packet leaves a tunnel and gets forwarded to its real destination after the packet is decapsulated and discovered it.

Therefore, in this paper, we introduce our definitions for security requirements and tunnels. And then we describe the formal conditions with regards to the tunnel definitions to discover potential conflicts. With these conditions, we present a general approach (CLID) that can be applied to any network topologies to identify and detect the security conflicts (including requirement violations and packet looping) among the security policies. The CLID algorithm is shown to correctly detect conflicts and looping. It works on a general topology, although the definition of a tunnel specifies a route that the tunneled packet currently follows and the route may change at any time due to dynamic routing aspect in any topology. We also implemented the algorithm that takes a set of security requirements and a set of policies and output the validation results. The simulation demonstrates the ease of use of the validation algorithm and also that its scalability is quite feasible to be adopted for a centralized policy management system in practice.

We have done a set of very small scale experiments running at DETER testbed to demonstrate the potential problems among the policies and some of the results have been included in the introduction of the paper. The experiments can be further extended in order to get more realistic network topology scenarios to make larger scale experiments.

The CLID algorithm takes the requirements and policies as input and detects conflicts among them. Though not a real-time solution, it can be re-run with any new data collected over a *dynamic* network. This solution provides a way to get any updated and new requirements and policies from the network for CLID algorithm to deal with the *dynamics* of the network. To seek a more active way to deal with this can be extended as the future work of the CLID.

Furthermore, the CLID algorithm is designed to find security conflicts, however, after discovering the security issues, how to correct the policies and solve the conflicts is still open. CLID does not make the corrections to eliminate the potential security violations. Future research should focus on given the problems that CLID detects how to automatically generate a solution for the problems that CLID detects. Also a completeness proof of the CLID algorithm would be an extension to the existing work.

References

[1] Yanyan Yang, C. Martel, S. F. Wu, " On Building the Minimum Number of Tunnels - An Ordered-Split approach to manage IPSec/VPN policies", 9th IEEE/IFIP Network Operations and Management Symposium (NOMS 2004), Seoul, Korea, April 2004.

[2] Yanyan Yang, Z. Fu, S. F. Wu, "BANDS: An Inter-Domain Internet Security Policy Management System for IPSec/VPN", 8th IFIP/IEEE International Symposium on Integrated Network Management (IM 2003), Colorado Springs, Colorado, March 2003.

[3] Z. Fu and S. F. Wu, “Automatic Generation of IPSEC/VPN Policies in an Intra-Domain Environment”, 12th International Workshop on Distributed Systems: Operations & Management (DSOM 2001), October 15-17, 2001, Nancy, France.

[4] Z. Fu, “Automatic Generation of Security Policies”, Technical Report, http://shang.csc.ncsu.edu/secpolicy.pdf.

[5] Z. Fu, S. F. Wu, H. Huang, K. Loh, F. Gong, “IPSec/VPN Security Policy: Correctness, Conflict Detection and Resolution”, IEEE Policy 2001 Workshop, Jan. 2001..

[6] M. Blaze, A. Keromytis, A. Keromytis, L. Sanchez, “IP Security Policy Requirements”, draft-ietf-ipsp-requirements-02.txt, Internet Draft, IPSP Working Group, August 2002.

[7] S. Kent, “IP Encapsulating Security Payload (ESP)”, draft-ietf-ipsec-esp-v3-06.txt, Internet Draft, IPsec Working Group, July 2003.

[8] S. Kent, “IP Authentication Header”, draft-ietf-ipsec-rfc2402bis-04.txt, Internet Draft, IPsec Working Group, July 2003.

[9] Nigel Sheridan-Smith, Tim O Neill, John Leaney, Mark Hunter, "Enhancements to Policy Distribution for Control Flow, Looping and Transactions", 16th IFIP/IEEE Distributed Systems: Operations and Management, Spain, October, 2005.

[10] Bernhard Kempter, Vitalian Danciu, "Generic policy conflict handling using a priori models", 16th IFIP/IEEE Distributed Systems: Operations and Management, Spain, October, 2005.

[11] Wayne Jansen, Tom Karygiannis, Michaela Iorga, Serban Gravila, and Vlad Korolev, “Security Policy Management for Handheld Devices”, The 2003 International Conference on Security and Management (SAM'03), June 2003.

[12] Hu V., D. Frincke, D. Ferraiolo, “The Policy Machine for Security Policy Management”, International Conference on Computational Science (2) 2001.

[13] Devarapalli V., et al, "NEMO Basic Support Protocol", Internet Draft: draft-wakikawa-nemo-basic-02.txt, Work In Progress, December 2003.

[14] Threat Analysis on NEtwork MObility (NEMO) by Souhwan Jung, Fan Zhao,S. Felix Wu,HyunGon Kim, ICICS'2004.

[15] Tzong-Jye Liu, Yuan-Siao Liu, Kuong-Ho Chen1, and Chyi-Ren Dow, "ZERO-Conflict: A Grouping-based Approach for Automatic Generation of IPSec/VPN Security Policies", 17th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management Large Scale Management (DSOM 2006), to appear.

[16] C. L. Chang, Y. P. Chiu and C. L. Lei, "Automatic Generation of Conflict-Free IPSec Policies", 25th IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems (FORTE), Taiwan, October 2005.

[17] Deterlab - Network Security Testbed based on Emulab, http://www.deterlab.net.