Modeling and Verification of IPSec and VPN Security Policies

Hazem Hamed, Ehab Al-Shaer and Will Marrero
School of Computer Science, DePaul University, Chicago, USA

Abstract

IPSec has become the defacto standard protocol for secure Internet communications, providing traffic integrity, confidentiality and authentication. Although IPSec supports a rich set of protection modes and operations, its policy configuration remains a complex and error-prone task. The complex semantics of IPSec policies that allow for triggering multiple rule actions with different security modes/operations coordinated between different IPSec gateways in the network increases significantly the potential of policy misconfiguration and thereby insecure transmission. Successful deployment of IPSec requires thorough and automated analysis of the policy configuration consistency for IPSec devices across the entire network.

In this paper, we present a generic model that captures various filtering policy semantics using Boolean expressions. We use this model to derive a canonical representation for IPSec policies using Ordered Binary Decision Diagrams. Based on this representation, we develop a comprehensive framework to classify and identify conflicts that could exist in a single IPSec device (intra-policy conflicts) or between different IPSec devices (inter-policy conflicts) in enterprise networks. Our testing and evaluation study on different network environments demonstrates the effectiveness and efficiency of our approach.

1 Introduction

The Internet Protocol Security architecture (or IPSec) [17] has been proposed by IETF to provide integrity, confidentiality and authentication of data communications over IP networks. The end users can specify the security protocol (AH or ESP) and mode (tunnel or transport) to accommodate the traffic security requirements. IPSec devices typically encrypt and encapsulate the outgoing IP packets, while the receiving devices decapsulate and decrypt the incoming packets in order to verify integrity and authenticity. IPSec operations can be performed either at the traffic source and destination (transport mode) or at intermediate *security gateways* (tunnel mode) in order to allow for source-based or domain-based security, respectively. Due to the flexibility and application transparency of IPSec, it is widely used today as a very cost-effective means to establish Virtual Private Networks (VPNs) or secure channels between corporate networks over the Internet. Users or administrators write the *security policy* at each device interface to define IPSec protection operations for each specific traffic. The IPSec policy consists of lists of rules that designate the traffic to be protected, the type of protection, such as authentication or confidentiality, and the required protection parameters, such as the encryption algorithm [16]. Packets are sequentially matched against the rules until one (*single-trigger*) or more (*multiple-trigger*) matching rules are found [7, 17].

Deploying IPSec policy rules at many hosts and gateways provides incredible flexibility for customizing the appropriate protection mechanisms for different applications and network requirements. However, the lack of automated verification of IPSec security policies significantly increases the potential of policy inconsistency and conflicts allowing for more network vulnerability. Many challenges confront the verification of IPSec policy configuration in enterprise networks. First, the sequential rule matching and multi-trigger semantics make policy verification of single or distributed IPSec policies a very complex and error-prone task, particularly when large number of rules and devices exist. Second, the interaction between different IPSec policies, such as cascaded protection and overlapping tunnels, can lead to inefficient or incorrect data protection. Third, the existence of various action types (*e.g.*, bypass, discard, encrypt/tunnel, authenticate/transport, etc.) poses another challenge when modeling and analyzing IPSec policies. Rule conflicts can occur due to IPSec misconfiguration within a single policy (called *intra-policy conflicts*) or due to the inconsistency between policies in different devices (called *inter-policy conflicts*). These conflicts may result in incorrect operation of IPSec and can lead to serious security threats including transmitting traffic insecurely, dropping legitimate traffic, and allowing undesired traffic into secure networks.

Therefore, successful deployment of IPSec security is highly dependent on the availability of IPSec policy analysis techniques with minimal human intervention. Our contribution in this paper is two-fold. First, we present a generic model that uses Boolean expressions to capture the single-trigger and multi-trigger semantics of a wide range of filtering policies. Second, we introduce a novel framework that uses this model implemented in Ordered Binary Decision Diagrams (OBDDs) to provide comprehensive identification and classification of IPSec policy conflicts. Based on this framework, we develop a set of techniques to discover intra- and inter-policy conflicts in any general IPSec policy configuration. These techniques are implemented in the "Security Policy Advisor" tool, which proved effective in discovering IPSec policy conflicts with acceptable processing and memory overhead.

$$
\begin{aligned}
access\_list &:= access\_rule[\ldots, access\_rule] \\
access\_rule &:= order, filter, action \\
filter &:= protocol, src\_ip, src\_port, dst\_ip, dst\_port \\
action &:= \text{protect} \mid \text{bypass} \mid \text{discard} \\
map\_list &:= map\_rule[\ldots, map\_rule] \\
map\_rule &:= priority, filter, transform\_list \\
transform\_list &:= transform[\ldots, transform] \\
transform &:= sec\_protocol, encaps\_mode, parameters \\
sec\_protocol &:= \text{AH} \mid \text{ESP} \\
encaps\_mode &:= \text{Transport} \mid \text{Tunnel } tunnel\_dst
\end{aligned}
$$

Figure 1. Typical IPSec policy syntax.

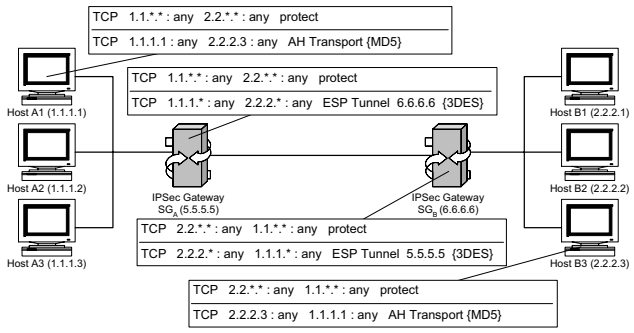

Figure 2. Example of an IPSec configuration.

Although IPSec has been deployed for many years, most of the related research work has been focused on addressing IPSec implementation problems. One related work [10] discovers the conflicts of overlapping IPSec tunnels using a simulation-based technique. Recent work [1] studies the policy conflicts particular to firewalls that are limited to "accept" and "deny" actions. Other related works [9, 15, 21] use a query-based approach to analyze firewall policies. None of the related work used formal methods to comprehensively identify IPSec policy conflicts. Therefore, we consider this work novel and significant not only in the area of IPSec but also for any filtering-based security policy such as firewalls and intrusion detection and prevention systems.

The rest of this paper is organized as follows. In Section 2 we highlight the main components and present our formal model of the IPSec policy. In Sections 3 and 4, we identify and define IPSec intra-policy and iter-policy conflicts respectively and we present techniques to discover them. In Section 5 we present a usability and performance study of our proposed techniques. In Section 6 we give a summary of the related work. Finally, in Section 7 we present our conclusion and plans for future research.

2 Modeling of Filtering Security Policies

A solid and flexible formal model that is capable of capturing the underlying policy semantics is needed to perform robust policy analysis. In this section, we show the components of IPSec policies, and present a formal model for any general filtering policy including IPSec.

2.1 IPSec policy components

The protection offered by IPSec to certain traffic is based on requirements defined by security policy rules defined and maintained by the system administrator [7, 16]. In general, packets are selected for a packet protection mode based on network and transport layer header information matched against rules in the policy, *i.e.,* transport protocol, source address and port number, and destination address and port number. To define traffic protection rules, the IPSec standard specifies the policy operational guidelines that should be implemented by vendors rather than a specific policy model [17]. In this work, we use a generic policy format that resembles the format used in a wide range of IPSec implementations [6]. This policy model is composed of two lists of packet-filtering rules:

Crypto-access list: consists of ordered filtering rules that specify required actions for packets that match the rule conditions. All traffic is matched against the access rules sequentially until a matching rule is found. The matching rule action is either "protect" for secure transmission, "bypass" for insecure transmission, or "discard" to drop the traffic.

Crypto-map list: consists of prioritized filtering rules that determine the cryptographic transformations required to protect the traffic selected for protection by the access list. A traffic may match multiple rules resulting in applying more than one transformation on the same traffic such that higher priority transformations are applied first.

IPSec policy rules can be written using the syntax shown in Fig. 1. The *access_list* is used to define IPSec protection rules, while the *map_list* is used to define IPSec transformation rules. A *transform* is any cryptographic service that can be used to protect network traffic. These security services are IPSec AH and ESP protocols operating either in transport or tunnel mode along with the cryptographic algorithm and the necessary cryptographic parameters. Fig. 2 shows an example of a typical outbound IPSec policy. The policy at each device is defined in terms of the access-list (upper section) and the map-list (lower section). In our work, we consider that inbound traffic arriving at a device interface is matched against a mirror image of the outbound IPSec policy of this interface,*i.e.,* the inbound policy is similar to the outbound policy after swapping the packet filters for source and destination addresses [6].

2.2 Filtering policy representation

Although our discussion in this section will focus on IPSec filtering policies, the presented framework can be used to model and analyze generic filtering policies.

Definition 1 *An* access policy, $P = R_1, R_2, \ldots, R_n$, *is a sequence of* n *filtering rules that determine the appropriate action performed on any incoming packet.*

Definition 2 *A* filtering rule, R_i, *consists of a set of constraints on a set of* k *filtering fields,* $F = \{f^1, f^2, \ldots, f^k\}$, *together with an action,* act_i, *from the set of all actions,* A.

Each rule can be written in the form:

$$R_i := C_i \rightsquigarrow act_i$$

where C_i is the constraint on the filtering fields that must be satisfied for the action act_i to be triggered. The condition C_i can be represented as a Boolean expression over the filtering field values $fv_i^1, fv_i^2, \ldots, fv_i^k$ as follows:

$$C_i = fv_i^1 \wedge fv_i^2 \wedge \cdots \wedge fv_i^k$$

For IPSec, a filtering field value fv is typically given as a binary expression representing the binary value of a specific IP address (123.45.201.5), a block of IP addresses (123.45.201.*), a specific port number (80 for http) or a range of port numbers (137-139 for netbios). Finally, the actions allowed are simply protect, bypass or discard.

Definition 3 *A* single-trigger access policy *is an access policy where only one action is triggered for a given packet. A* multi-trigger access policy *is an access policy where multiple different actions may be triggered for the same packet.*

IPSec crypto-access rules form a single-trigger access policy. Once a traffic matches a certain rule, its action is triggered and no further matching is performed. This is in contrast to crypto-map rules where a particular traffic may match multiple rules causing multiple actions to be triggered.

2.2.1 Formalization of single-trigger policies

The semantics of a single-trigger policy $P = R_1, R_2, \ldots, R_n$ can be represented as a collection of Boolean expressions, $[[P]] = \{P_{a_1}, P_{a_2}, \ldots, P_{a_m}\}$, one for each possible action $a_m \in A$. The expression for an action should evaluate to *true* for all packets that trigger the action and *false* otherwise. The fact that $[[P]]$ is a single-trigger policy implies that, for any given packet, only one policy expression evaluates to *true*, and all other expressions evaluate to *false*. In general, we can construct a Boolean expression for P_a by using the rule constraints from each rule as follows:

$$P_a = \bigvee_{i \in index(a)} (\neg C_1 \wedge \neg C_2 \wedge \ldots \wedge \neg C_{i-1} \wedge C_i)$$

where $index(a)$ is the set of indices of rules that have a as their action. In other words,

$$index(a) = \{i \mid R_i = C_i \rightsquigarrow a\}.$$

This formula can be understood as saying that a packet will trigger action a if it satisfies the condition C_i for some rule R_i with action a, provided that the packet does not match the condition of any prior rule in the policy.

We express the IPSec crypto-access policy, S, as a single-trigger policy composed of three action expressions, *i.e.*, $[[S]] = \{S_{protect}, S_{bypass}, S_{discard}\}$. Therefore, based on the above formalization, the IPSec protection access policy $S_{protect}$ can then be defined as follows:

$$S_{protect} = \bigvee_{i \in index(protect)} (\neg C_1 \wedge \neg C_2 \wedge \ldots \wedge \neg C_{i-1} \wedge C_i)$$

2.2.2 Formalization of multi-trigger policies

The semantics of a multi-trigger policy $[[P]]$ can also be represented as a collection of Boolean expressions, one for each action allowed by the policy. Like the single-action case, the expression for an action should evaluate to *true* for all packets which trigger the action and to *false* otherwise. Since $[[P]]$ is a multi-trigger policy, more than one action expression may evaluate to *true* for any given packet. Using the rule constraints from each rule, the Boolean expression for any possible action $a \in A$ is constructed as follows:

$$P_a = \bigvee_{i \in index(a)} C_i$$

The formula means that a packet will trigger action a if it satisfies the condition C_i for any rule R_i with action a.

We express the IPSec crypto-map policy, T, as a set of expressions each of which represents the condition that triggers a specific transform, *i.e.*, $[[T]] = \{T_{t_1}, T_{t_2}, ..., T_{t_m}\}$ where T_{t_m} could be for example $T_{\text{ESP-tunnel}}$, $T_{\text{AH-transport}}$ and so on. Thus, in general, the traffic transformation policy T_t that triggers a certain transform t can be represented as follows:

$$T_t = \bigvee_{i \in index(t)} C_i$$

2.2.3 Policy representation using OBDDs

The main objective of our policy representation as Boolean expressions is to formalize and facilitate policy analysis and rule conflict identification. Using Boolean expressions allows us to use ordered binary decision diagrams (OBDDs) [4] in our analysis techniques. OBDDs provide a canonical and concise representation for Boolean expressions and support all the common Boolean operations. Their usefulness can be highlighted by pointing out that two policies which are syntactically different (they have different rules) but are semantically equivalent (they exhibit identical behavior) will have the same OBDD representation. Policies can then be built up, combined, and compared using Boolean operations on the OBDDs representing them.

3 IPSec Intra-policy Analysis

In this section, we use our formal model to identify all possible types of conflicts that may exist in the policy of a single IPSec device. These conflicts may exist between rules in the crypto-access or crypto-map lists. We also prove that our conflict analysis is comprehensive.

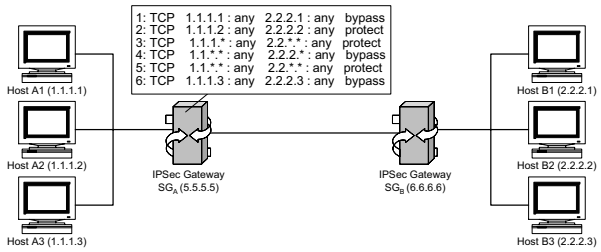

Figure 3. Example for IPSec intra-policy crypto-access list conflicts.

3.1 Classifying and discovering access-list conflicts

The ordering of access rules is crucial in determining IPSec access policy semantics. This is because the packet filtering process is performed by sequentially matching the packet against filtering rules until a match is found. If filtering rules are disjoint, the ordering of the rules is insignificant. However, it is very common to have filtering rules that are inter-related, *i.e.*, exactly matched, inclusively matched or correlated [2]. In this case, if the related rules are not carefully ordered, some rules may never get triggered because of other rules, resulting in an incorrect policy. In this section, we classify different conflicts that may exist among the rules in an access policy and then describe a technique for discovering these conflicts.

We apply our policy model on IPSec crypto-access lists as a special case of the single-trigger policy. The policy expression S_{a_i} represents a policy that incorporates rule R_i where $a_i = act_i$, while S'_{a_i} represents the policy with R_i excluded.

Intra-policy shadowing A rule is shadowed when every packet that could match this rule is matched by some preceding rule with a different action. Subsequently, the shadowed rule will never be activated. Based on our OBDD representation, a rule is shadowed if the policy expression for the rule action does not change when this rule is removed, and the rule is not implied in the modified policy expression. Formally, rule R_i is shadowed if the following condition is true:

$$[(S'_{a_i} \Leftrightarrow S_{a_i}) = true] \text{ and } [(C_i \Rightarrow S'_{a_i}) \neq true] \quad (1)$$

The first condition means that the policy semantics does not change after removing the rule, while the second condition means that the rule condition is not included in the policy semantics[1]. As an example for this case, rule 6 is shadowed by rule 5 in Fig. 3. Shadowing is a critical conflict because the shadowed rule never takes effect. This might cause a desired traffic to be discarded or an undesired traffic to be bypassed.

Intra-policy redundancy A rule is redundant when every packet that could match this rule is matched by some other rule that have a similar action, such that if the redundant rule is removed, the security policy will not be affected. In OBDD representation, a rule is redundant if the policy expression for the rule action does not change when this rule is removed, and the rule is implied in the modified policy expression. Formally, rule R_i is redundant if the following condition is true:

$$[(S'_{a_i} \Leftrightarrow S_{a_i}) = true] \text{ and } [(C_i \Rightarrow S'_{a_i}) = true] \quad (2)$$

Referring to Fig. 3, rule 2 is redundant to rule 3. Redundancy is considered a critical conflict because a redundant rule adds to the size of the filtering rule list, increasing the search time and space requirements of the packet filtering process.

Intra-policy correlation A rule is in correlation with another rule if they have different filtering actions, and the preceding rule matches some packets that match the following rule and vice versa. Using OBDDs, a rule is in correlation with another rule if the policy changes when this rule is removed, and this rule is not fully implied in the modified policy expression. Formally, correlation exists if the following condition is true:

$$[(S'_{a_i} \Leftrightarrow S_{a_i}) \neq true] \text{ and } [(C_i \Rightarrow \neg S'_{a_i}) \neq true] \quad (3)$$

Rule 3 is in correlation with Rule 4 in Fig. 3. The two rules with this ordering imply that all traffic that is coming from 1.1.1.* and going to 2.2.2.* is protected. However, if their order is reversed, the same traffic will be bypassed. Correlation is considered a potential conflict because the relative order of the correlated rules directly affects the policy semantics.

Intra-policy exception A rule is an exception of a following rule if they have different actions, and the following rule is a superset match. Based on OBDDs, a rule is an exception of a general rule if the policy changes when this rule is removed, and the rule is implied in the complement of the modified policy expression. Formally, exception exists if the following condition is true:

$$[(S'_{a_i} \Leftrightarrow S_{a_i}) \neq true] \text{ and } [(C_i \Rightarrow \neg S'_{a_i}) = true] \quad (4)$$

Rule 1 is an exception of rule 3 in Fig. 3. These two rules imply that all the traffic coming from the address 1.1.1.* will be protected, except the traffic coming from 1.1.1.1. Exception is considered a non-critical conflict because it is often desired to make exceptions of some general rule, and it is usually the case for rules that are only related to the default rule [2]. However, it is important to identify rule exceptions because they partially change the policy semantics and can lead to violation of the policy requirements.

Theorem 1 *The intra-policy access-list conflict conditions (Cases 1-4) are complete in the sense that every rule in a policy must satisfy one of the conflict conditions.*

[1]The result of an OBDD operation evaluates either to *true* for a satisfying assignment, *false* for a non-satisfying assignment or a *predicate* representing the expression resulting from an incomplete assignment.

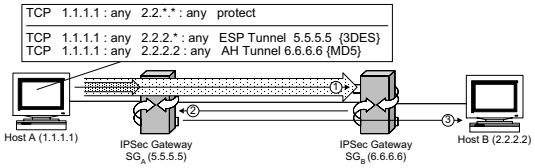

Figure 4. Example for IPSec intra-policy overlapping-session conflicts.

Proof sketch The logical disjunction of the four conditions is equivalent to *true*. The complete proof can be found in [13].

To discover intra-policy access-list conflicts, the analysis is performed for every possible action in the access policy; *i.e.*, protect, bypass and discard. For every rule in the access list that has the same action a, we build the policy BDD (S_a) as well as the test BDD (S'_a). If both BDDs are identical, the rule is either redundant or shadowed based on testing if the rule is implied in S'_a. If the BDDs are not identical and the rule implication test is true, then the rule is an exception of a more general rule, otherwise it is correlated to another rule in the policy. To find the related rule, every rule in the policy is sequentially matched against this rule and the first matching rule is reported. The full discovery algorithm with detailed description are presented in [13].

3.2 Classifying and discovering map-list conflicts

In this section, we identify the rule conflicts that may exist in a single IPSec crypto-map list and result in security policy violation or unnecessary traffic protection.

3.2.1 Intra-policy overlapping-session conflicts

IPSec allows nesting multiple IPSec sessions on the same traffic from a source to different remote peers. In this case, in order to construct correct nesting, the traffic must be delivered to the closer peer first and then to the farther peer. In other words, in the map-list, the rule priority of the farther peer should be higher than the rule priority of the closer peer. This is mandatory because if the traffic is decapsulated at the farther peer first, it will be transmitted to the closer peer in the opposite direction, resulting in transmitting the traffic back to the destination without protection. The example in Fig. 4 illustrates this situation. In this example, two map rules apply to the traffic flowing from A to B. The first rule encapsulates the traffic in a tunnel to SG_A, then the second rule re-encapsulates the traffic in another tunnel to SG_B. The traffic is first received and decapsulated by SG_B and then forwarded back to SG_A. SG_A decapsulates the traffic and forwards it to B as clear text. Notice that this conflict can only occur with two tunnel transforms or with a transport transform followed by a tunnel. Other rule combinations will send the traffic to the same destination node. Also notice that if the nested sessions terminate at the same end point, the rule ordering is not required because all decapsulation will be performed at the same node.

Formally, the intra-policy overlapping-session conflict occurs when the following condition is true for any two tunnel-mode map-list rules R_i and R_j:

$$[(C_i \wedge C_j) \neq false] \text{ and } (i < j) \text{ and}$$
$$\text{Location}(R_i[tunnel\_dst]) < \text{Location}(R_j[tunnel\_dst]) \quad (5)$$

The first condition expresses the fact that the two rules must match some common traffic, and the other conditions verify that the tunnel end-point of the preceding rule comes before the tunnel end-point of the following rule in the path from $R_i[src\_ip]$ to $R_i[dst\_ip]$. A similar condition holds for any transport rule followed by a tunnel rule, but using $R_i[dst\_ip]$ instead of $R_i[tunnel\_dst]$. Later in Section 4.2.1, we prove that these conditions are comprehensive.

To discover intra-policy overlapping-session conflicts, we search for the rules that match the same traffic and satisfy the conflict conditions. The topology of the network can be encoded using OBDDs in a manner similar to what is done in symbolic model checking [5]. The location of every node can be encoded using OBDDs such that the locations of any two nodes can be retrieved and compared relative to a certain node. We start the analysis by finding every two rules that partially or completely overlap, thus matching the same traffic. A path conflict is reported if one rule specifies a tunnel transform terminating at a further point than the end point of a preceding overlapping rule. The full discovery algorithm and a detailed description of the technique are provided in [13].

3.2.2 Intra-policy multi-transform conflicts

IPSec also allows for multiple transforms to be applied to the same traffic simultaneously. This gives the user the flexibility to combine different IPSec protection methods to achieve the traffic security goals. However, some of these combinations provide weak protection, such as applying ESP transport after AH transport because ESP transport does not provide IP header protection. Moreover, other combinations may not improve traffic protection but cause performance overhead, such as applying AH tunnel followed by AH transport [3].

Formally, the intra-policy multi-transform conflict occurs when the following conditions are true for any two map-list rules R_i and R_j:

$$[(C_i \wedge C_j) \neq false] \text{ and } (i < j) \text{ and}$$
$$\text{Strength}(R_i[transform]) > \text{Strength}(R_j[transform])$$
$$\text{Location}(R_i[tunnel\_dst]) \geq \text{Location}(R_j[tunnel\_dst]) \quad (6)$$

Here we introduce the *transform strength* concept as the level of protection the transform provides for a particular traffic. For flexibility, the strength of any transform t_i can be user-defined, and we refer to it as Strength(t_i). If Strength(t_i)>Strength(t_j), then the transform t_i provides better protection than t_j, and vice versa. The first condition expresses the fact that the two rules must match some common traffic, and the second find if a weaker transform is applied on a stronger one. The third condition verifies that the tunnel end-point of the preceding rule comes after the tunnel end-point of the following rule in the path from $R_i[src\_ip]$ to $R_i[dst\_ip]$.

A similar condition holds for any transport rule followed by a tunnel rule, but using $R_i[dst\_ip]$ instead of $R_i[tunnel\_dst]$. If the third condition is not true, the conflict reduces to the overlapping-session conflict described earlier in Section 3.2.1. We prove that these conditions are comprehensive later in Section 4.2.2.

Multi-transform intra-policy conflicts can be easily discovered by searching the map policy for rules that contain conflicting transforms. We start by building the OBDDs for map-list entries that include any two conflicting transforms. Then we get the intersection OBDD that represents the overlap condition where both transforms are applicable. For every map-list rule that intersects with the overlap condition, we check if it provides the weaker protection. In this case, a conflict is reported for this rule provided that the rule end point satisfies the location condition. The full discovery algorithm with detailed description are presented in [13].

4 IPSec Inter-policy Analysis

In this section, we identify all policy conflicts that may exist between any two different IPSec devices. These include conflicts between rules of the crypto-access lists or crypto-map lists in different IPSec devices. We also prove the comprehensiveness of our conflict analysis.

4.1 Classifying and discovering access-list conflicts

Because of the decentralized nature inherent to the IPSec security policy, the potential of conflicts between policies in different IPSec devices is significantly high. Even if every IPSec device policy in the network does not contain any of the intra-policy conflicts described in Section 3, conflicts could exist between policies of different IPSec devices. For example, an upstream device might protect a traffic that is bypassed by a downstream device or vice versa. In the first case, the traffic will be dropped at the upstream peer because SA negotiation will fail. In the second case, the traffic will be dropped at the downstream peer because it will not be able to perform the decapsulation. In this section, we first define the conflicts that may exist between an upstream and a downstream access policy, and then we describe a technique to discover these conflicts.

In this discussion, S^k resembles the access policy of an IPSec device D_k that exists along a certain network path. Also in our discussion, for simplicity and without loss of generality, we analyse the policies of only two devices on any network path, the upstream device (D_u) and the downstream device (D_d). However, the analysis can be performed iteratively on every two IPSec devices in the network in order to verify the IPSec policies in the entire network. Based on these assumptions, we can now formally define IPSec inter-policy conflicts as follows.

Inter-policy shadowing Traffic is shadowed if the upstream policy S^u blocks some traffic permitted by the downstream policy S^d. Formally, inter-policy shadowing exists if the following condition is true:

$$[(S^u_{discard} \wedge \neg S^d_{discard}) \vee (S^u_{protect} \wedge \neg S^d_{protect})] \neq false \quad (7)$$

This expression represents the filtering condition that results in shadowing some traffic by D_u. The first term represents the traffic discarded by D_u but permitted by D_d while the second term represents the traffic that requires protection by D_u but is not protected by D_d. In the second case, SA negotiation fails and the traffic is discarded at the upstream device. Rule 2 in SG_A and rule 2 in SG_B show an example of inter-policy shadowing. Shadowing is considered a critical conflict because it prevents the traffic desired by some nodes from flowing to the end destination.

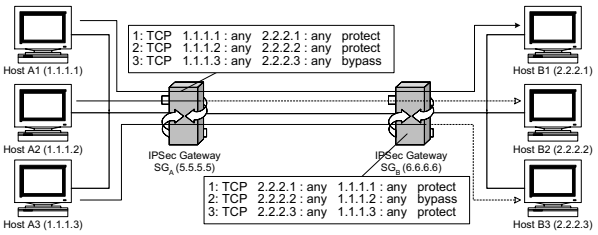

Figure 5. Example for IPSec inter-policy crypto-access list conflicts.

Inter-policy spuriousness Traffic is spurious if the upstream policy S^u permits some traffic blocked by the downstream policy S^d. Formally, inter-policy spuriousness exists if the following condition is true:

$$[(S^u_{bypass} \wedge \neg S^d_{bypass}) \vee (S^u_{protect} \wedge S^d_{discard})] \neq false \quad (8)$$

This expression represents the filtering condition that results in spurious traffic flowing to D_d. The first term represents the traffic permitted by D_u but not permitted by D_d, while the second term represents the traffic protected and permitted by D_u but discarded by D_d. The second case applies when intermediate downstream IPSec devices are not configured to bypass the protected traffic. Rule 3 in SG_A and rule 3 in SG_B show an example of inter-policy spuriousness. Spuriousness is a critical conflict because it allows unwanted traffic to flow into the network, increasing the network vulnerability to various network attacks such as port scanning, denial of service, etc.

Theorem 2 *The inter-policy access-list conflict conditions (Cases 7,8) are complete in the sense that any policy inconsistency between two IPSec devices must satisfy one of the conflict conditions.*

Proof sketch For any packet, an upstream device may perform one of three actions: bypass, protect, or discard. For that same packet, a downstream device may perform one of the same three actions. These means there are nine possible combinations. The only combinations that do not satisfy one of

the conflict conditions are combinations where the upstream device and the downstream device perform the same action. Clearly, these cases are not conflicts [13].

To discover these conflicts, we analyze the upstream outbound access policy against the downstream inbound access policy[2]. First, we construct the BDD for each of the conflict conditions defined above. Each rule in the upstream policy is checked if it intersects with any of the conflict conditions. If an intersection is found, we look for the corresponding rule in the downstream policy. Again, we match every rule in the downstream policy against the conflict condition until we find an intersecting rule. If the downstream rule also matches the upstream rule, then the discovered conflict is reported along with the involved rules. The full discovery algorithm with detailed description are presented in [13].

4.2 Classifying and discovering map-list conflicts

In this section, we identify the conflicts that may occur between rules in different IPSec crypto-map lists and result in security policy violation or unnecessary traffic protection.

4.2.1 Inter-policy overlapping-session conflicts

IPSec allows applying nested sessions on the same traffic at different points on the traffic path to multiple remote peers. Similar to the intra-policy case, the traffic must be transferred to the closer peer first and then to the farther peer. In other words, the packets should be decapsulated in reverse order of their encapsulation at subsequent points on the traffic path, otherwise unprotected traffic is transmitted to the destination. The example in Fig. 6 illustrates this case. In this example, two IPSec sessions are used to protect the traffic flowing from A to B. The sessions start at A and SG_A and encapsulate the traffic in tunnels terminating at SG_B and SG_C respectively. The traffic is first received and decapsulated by SG_C and then forwarded back to SG_B. SG_B decapsulates the traffic and forwards it to B as clear text. Notice that this conflict can occur with either two tunneled transforms, or a transport transform followed by a tunnel. Other rule combinations are not feasible because IPSec transport sessions cannot be initiated at intermediate security gateways.

Formally, the inter-policy overlapping-session conflict occurs when the following condition is true for any two tunnel-mode map-list rules R_i^u in the upstream device, and R_j^d in the downstream device:

$$\begin{gathered} R_i^u[src\_ip] \subseteq R_j^d[src\_ip] \text{ and} \\ R_i^u[tunnel\_dst] \subseteq R_j^d[dst\_ip] \text{ and} \\ \text{Location}(R_i^u[tunnel\_dst]) < \text{Location}(R_j^d[tunnel\_dst]) \end{gathered} \quad (9)$$

The first two conditions express the fact that the traffic that matches the upstream rule also matches the downstream rule. The last condition verifies that the tunnel end-point of the upstream rule comes before the tunnel end-point of the upstream rule in the path from $R_i^u[src\_ip]$ to $R_i^u[dst\_ip]$. A similar condition holds for any transport rule followed by a tunnel rule, but using $R_i^u[dst\_ip]$ instead of $R_i^u[tunnel\_dst]$ in the above condition.

[2]Recall that, in general, the inbound policy of an IPSec device is a mirror image of the outbound policy.

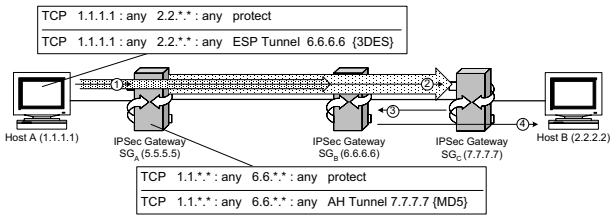

Figure 6. Example for IPSec inter-policy overlapping-session conflicts.

Theorem 3 *The overlapping-session map-list conflict conditions (Cases 5,9) are complete in the sense that any security violation must satisfy one of the conflict conditions.*

Proof sketch Nested IPSec sessions can start/terminate at the same node or at different subsequent nodes on the traffic path. Therefore, to create two nested IPSec sessions, we have six possible scenarios. Only two of these scenarios correspond to the decapsulation being performed in the wrong order. The conflict conditions are the formal descriptions of exactly these two cases [13].

A procedure similar to the one presented in Section 3.2.1 can be used to discover overlapping-session conflicts. For every upstream device D_u and downstream device D_d, we analyze the map-list rules that match the same traffic in both devices but terminate at different end points. Every outbound map-list rule in D_u is checked against all the inbound map-list rules of D_d. In tunnel mode rules, the source and destination filters are replaced by the tunnel end-points to resemble the resulting packet header. If any two rules overlap, and the downstream rule specifies a tunnel terminating at a farther point than the end-point of the upstream rule, a session conflict is reported. The full discovery algorithm and a detailed description of the technique are provided in [13].

4.2.2 Inter-policy multi-transform conflicts

IPSec also allows intermediate nodes to apply traffic protection on already protected traffic. However, this might be unnecessary and can cause extra overhead particularly if the new protection is weaker than the existing one. For example, applying an AH tunnel on traffic already encapsulated in an ESP tunnel does not improve the security protection [3]. Formally, the inter-policy overlapping-session conflict occurs when the following condition is true for any two tunnel-mode map-list rules R_i^u in the upstream device, and R_j^d in the downstream device:

$$R_i^u[src\_ip] \subseteq R_j^d[src\_ip] \text{ and}$$
$$R_i^u[tunnel\_dst] \subseteq R_j^d[dst\_ip] \text{ and}$$
$$\text{Strength}(R_i^u[transform]) > \text{Strength}(R_j^d[transform])$$
$$\text{Location}(R_i^u[tunnel\_dst]) \geq \text{Location}(R_j^d[tunnel\_dst]) \quad (10)$$

Similarly, the same condition holds for any transport rule followed by a tunnel rule, but using $R_i^u[dst\_ip]$ instead of $R_i^u[tunnel\_dst]$ in the above condition.

Theorem 4 *The multi-transform map-list conflict conditions (Cases 6,10) are complete in the sense that any unnecessary protection between a pair of map-list rules must satisfy one of the conflict conditions.*

Proof sketch The two rules could be from the same policy or from different policies. If a pair of rules do not satisfy either of the conditions, then one or more of the following are true: (1) there are no packets that match both rules, (2) the protection offered by the earlier transform is weaker than the protection offered by the later transform, and (3) the second transformation is undone at a further point on the traffic path than the first transformation. In the first two cases, the second transformation does provide some added protection. The third case is clearly an overlapping-session conflict [13].

Similar to the approach presented in Section 3.2.2, to discover this conflict we build the OBDDs for map list entries that include two conflicting transforms in two different IPSec devices. Then we get the intersection OBDD that represents the traffic condition where both transforms are applicable. For every rule in the downstream map-list, we verify that it does not provide the weaker protection when the location condition is satisfied. The full discovery algorithm with detailed description are presented in [13].

5 Usability and Performance Evaluation

We implemented the techniques described in Sections 3 and 4 in a software tool called the "Security Policy Advisor" or SPA. The SPA was developed using the Java programming language and BuDDy, an OBDD package implemented in Java [18]. In this section, we present our evaluation of the usability and the performance of the IPSec policy analysis techniques described in this paper.

To assess the practical value of our techniques, we first used the SPA tool to analyze real IPSec policy rules in our university network as well as in some local industrial networks in the area. In many cases, the SPA has shown to be effective by discovering many policy conflicts that were not discovered by human visual inspection. We made an attempt to quantitatively evaluate the practical usability of the SPA by conducting a set of experiments that consider the level of network administrator expertise. In this experiment, we created two IPSec policy exercises and recruited 38 network administrators with varying level of expertise in the field (7 experts, 12 intermediates and 19 beginners) to complete each exercise. The exercise included writing IPSec access list and map list rules based on a set of access-control policy requirements for 9 interconnected networks with 12 IPSec security gateways (intermediate and end-point gateways). We then used the SPA tool to analyze the rules and count different types of conflicts. The experiment results in Fig. 7 show the percentage of persons who introduced various types of conflicts in their IPSec policy configuration.

Experience	Access-list	Overlapping-session	Multi-transform
	Intra-Policy Conflicts		
Expert (7)	14%	14%	0%
Intermediate (12)	42%	33%	8%
Beginner (19)	84%	63%	16%
Conflict type %	19%	9%	7%
	Inter-Policy Conflicts		
Expert (7)	29%	14%	14%
Intermediate (12)	50%	33%	17%
Beginner (19)	90%	53%	16%
Conflict type %	38%	16%	11%

Figure 7. The percentage of administrators who created conflicts in the IPSec policy.

The results show clearly that even the expert administrators created policy conflicts. A total of about 29% of experts created intra-policy and inter-policy conflicts. This figure is even much higher for intermediate and beginner administrators. The table also shows the average ratio of each conflict type relative to the total number of discovered conflicts. The results clearly indicate that access-list conflicts (38%) dominate the misconfiguration errors made by administrators.

In the second phase of our evaluation study, we conducted a number of experiments to measure the performance and the scalability of policy conflict discovery under different filtering policies and network sizes. Our experiments were performed on a Pentium PIII 600MHz processor with 512MB RAM.

To study the performance of the intra-policy conflict discovery techniques, we produced three sets of policy rules that reflect different space and processing requirements. The first set includes rules that have IP source and destination address ranges to resemble the best case scenario. In the second set, each rule has fully specified IP addresses for the source and destination, representing the worst case scenario. The third set includes rules that are randomly selected from the two previous sets in order to represent the average case scenario and resembles a realistic IPSec policy. We used the SPA tool to run the intra-policy analysis technique on each set using various sizes of rule sets (10-100 rules). In each case, we measured the processing time and memory space needed to produce the policy analysis report. The processing time results we obtained are shown in Fig. 8-(a). Set 1 shows the least processing time, Set 3 is expected to have the highest processing, and Set 2 shows a moderate processing time. Even in the worst case scenario (Set 3), the processing time looks very reasonable; approximately 20-220 ms for 10-100 rules respectively. The memory space needed in the analysis is plotted in Fig. 8-(b). In the worst case, only 56 kbytes are needed to create the OBDDs used to analyze a policy of 100 rules.

For evaluating the performance of the inter-policy conflict

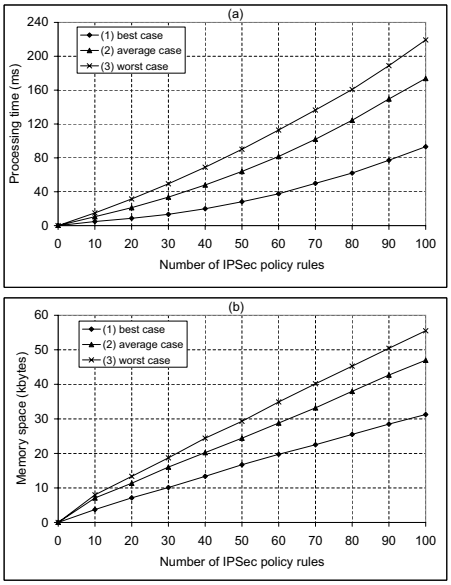

Figure 8. Intra-policy conflict discovery (a) processing time, (b) memory space.

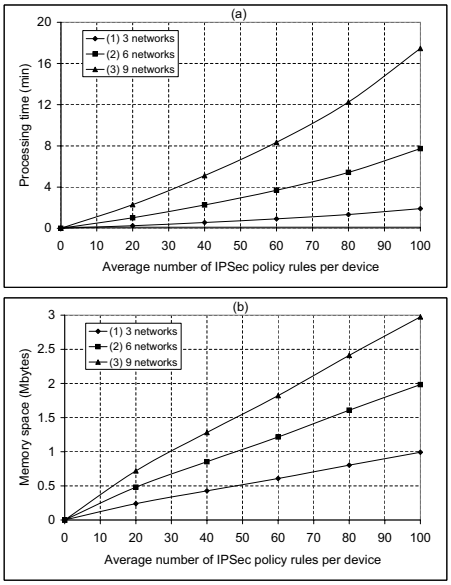

Figure 9. Inter-policy conflict discovery (a) processing time, (b) memory space.

discovery techniques, we conducted a similar experiment on a network of IPSec devices resembling a realistic IPSec configuration. The topology is composed of several local networks connected globally to the Internet. Each network contains one IPSec security gateway and 30 IPSec enabled hosts protected by the gateway. Each host can establish IPSec sessions with 60% of the hosts in other networks. We created three instances of the topology each with an different number of interconnected networks: 3 networks, 6 networks, and 9 networks. For each IPSec node, we installed a random set of IPSec rules to protect the traffic flowing to other networks. We then used the SPA to run the inter-policy analysis technique on every pair of interacting IPSec nodes in each topology with a varying number of policy rules (10-100 rules). For each topology, we measured the total processing time and memory space required to perform policy analysis. The processing time results are shown in Fig. 9-(a). We noticed that when the analysis is performed on a small number of networks, the processing time ranges from 40 seconds to 2 minutes. However, as more networks are involved in the analysis, the policy conflict discovery requires quadratically increasing processing time ranging from 1 to 18 minutes depending on the rule complexity. Fig. 8-(b) shows the memory space used in the analysis. The plot reflects very reasonable memory requirements (less than 3 Mbytes) even for the large network.

6 Related Work

Since IPSec was published as an IETF draft in the late nineties, it has gained much attention from networking vendors and research institutions. However, most of these efforts were mainly focused on IPSec implementation issues, the reason why the IPSec standard is still under continuous review by the IETF community. A few research works were published in the specific area of IPSec policy conflict analysis, however, a significant amount of work has been reported in the area of firewall and policy-based security management. In this section, we shall focus our study on two aspects of related work: (1) packet filter modeling, and (2) conflict discovery and policy analysis.

Several models have been proposed for packet filtering rules. A humble attempt to use OBDDs to model firewall rules was presented in [15]. This work focused on the hardware implementation of firewalls and did not define a formal model for filtering policies or present a framework for conflict detection. Moreover, this approach was limited on firewall-specific rules that use only “accept” and “deny” actions. Other models use specialized data structures like hash tables, bucket filters, binary tries and geometric structures [12]. Because these models were designed particularly to optimize packet classification in high-speed networks, they are too complex to use for filtering policy analysis. Interval diagrams are used in [11] to compact firewall rules. However, this approach can only be used with non-overlapping rules, which significantly restricts the practical use of this technique to analyze general filtering policies.

A variety of approaches have been proposed in the area of policy conflict analysis. The most significant attempt for IPSec policy analysis is proposed in [10]. The technique simulates IPSec processing by tracking the protection applied on the traffic in every IPSec device and reports a conflict if the IPSec policy is violated. Although this approach can discover IPSec policy violations in a certain simulation scenario, there is no guarantee that it discovers every possible violation that may exist. In addition, the proposed technique only discovers IPSec conflicts resulting from incorrect tunnel overlapping, but do not address the other types of conflicts that we study in

this paper. Our previous work in firewall policy analysis [2, 1] as well as the work in [19] were a significant advance in the area. However, these works were limited to firewall policy analysis and cannot be easily extended for IPSec. Both [8] and [14] provide algorithms for detecting and resolving conflicts among general packet filters. However, they only detect what we defined as correlation conflict because it causes ambiguity in packet classifiers. Other research work goes one step forward by offering query-based tools for firewall policy analysis [9, 15, 21]. Even though these tools can be extended to run queries to analyze IPSec policies, they cannot provide predefined and automated conflict discovery. Moreover, they have limited practical usability as they require high user expertise to write the proper queries to identify different policy problems. Other work in this area addresses general management policies rather than filtering policies [20]. Although this work is very useful as a general background, it cannot be directly used for IPSec conflict discovery.

Therefore, based on our search, we could not find any previous work offering a comprehensive conflict analysis framework for IPSec policies using formal verification techniques.

7 Conclusions and Future Work

Although the IPSec standard provides various flexible data protection schemes for IP networks, configuring IPSec policies manually can be extremely complex and error-prone, particularly in enterprise networks. An exhaustive analysis of policy rules in all IPSec gateways is required to discover policy conflicts and avoid serious network security threats like insecure transmission and flooding attacks. IPSec security, like any other technology, requires proper management support, including automatic conflict analysis and verification, in order to provide the required security services.

In this paper, we attempt to bridge this gap by presenting (1) a new formal model that covers the semantics of a wide range of filtering policies including IPSec, and (2) a sound and complete framework for analyzing IPSec policy conflicts. The verification framework utilizes OBDDs, a well-known powerful verification tool that is widely used in many fields, to represent IPSec policies and derive solid formulation of policy conflicts. Based on this framework, we developed techniques for identifying rule conflicts in IPSec policies of a single device or across multiple inter-connected devices. Our approach is sufficiently general to be used for verifying many other filtering-based security policies such as firewalls, intrusion detection systems and access control devices. We show that our implementation of these techniques in a tool called the "Security Policy Advisor" is very effective in checking real-life IPSec policies. For example, our tool was able to discover conflicts in IPSec policies that were overlooked by up to 29% of expert network administrators in our experiment. Our experiments have also shown that the average processing time in intra- and inter-policy conflict discovery is very reasonable for off-line analysis in many network configurations.

There is much more research to pursue in the automation of security policy management. Our future research plan includes online discovery and recovery of conflicts created as a result of policy updates, and discovery of conflicts between IPSec devices and other security devices like firewalls and NATs.

References

[1] E. Al-Shaer and H. Hamed. Discovery of policy anomalies in distributed firewalls. In *IEEE INFOCOM'04*, March 2004.

[2] E. Al-Shaer and H. Hamed. Modeling and management of firewall policies. *IEEE eTransactions on Network and Service Management (eTNSM)*, 1(1), April 2004.

[3] S. Bellovin. Problem areas for the IP security protocols. In *The 6^{th} Usenix UNIX Security Symposium*, July 1996.

[4] R. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[5] J. Burch, E. Clarke, K. McMillan, D. Dill, and J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Journal of Information and Computation*, 98(2), 1992.

[6] Cisco Systems. Configuring IPSec network security. In *Cisco IOS Security Configuration Guide, Release 12.2*, 2003.

[7] N. Doraswamy and D. Harkins. *IPSec: The New Security Standard for the Internet, Intranets, and Virtual Private Networks*. Prentice Hall PTR, second edition, March 2003.

[8] D. Eppstein and S. Muthukrishnan. Internet packet filter management and rectangle geometry. In *The 12^{th} ACM Symposium on Discrete Algorithms (SODA'01)*, January 2001.

[9] P. Eronen and J. Zitting. An expert system for analyzing firewall rules. In *The 6^{th} Nordic Workshop on Secure Systems*, November 2001.

[10] Z. Fu, F. Wu, H. Huang, K. Loh, F. Gong, I. Baldine, and C. Xu. IPSec/VPN security policy: Correctness, conflict detection and resolution. In *Policy'2001 Workshop*, January 2001.

[11] M. Gouda and X. Liu. Firewall design: Consistency, completeness, and compactness. In *The 24^{th} IEEE Int. Conference on Distributed Computing Systems (ICDCS'04)*, March 2004.

[12] P. Gupta and N. McKeown. Algorithms for packet classification. *IEEE Network*, 15(2):24–32, 2001.

[13] H. Hamed, E. Al-Shaer, and W. Marrero. Design and implementation of security policy advisor tools. Technical Report CTI-TR-05-010, DePaul University, May 2005.

[14] B. Hari, S. Suri, and G. Parulkar. Detecting and resolving packet filter conflicts. In *IEEE INFOCOM'00*, March 2000.

[15] S. Hazelhurst, A. Attar, and R. Sinnappan. Algorithms for improving the dependability of firewall and filter rule lists. In *IEEE Workshop on Dependability of IP Applications, Platforms and Networks*, June 2000.

[16] K. Jason, L. Rafalow, and E. Vyncke. IPsec configuration policy information model. IETF RFC-3585, August 2003.

[17] S. Kent and R. Atkinson. Security architecture for the internet protocol. IETF RFC-2401, November 1998.

[18] J. Lind-Nielsen. The buddy obdd package. http://www.bdd-portal.org/buddy.html.

[19] A. Liu and M. Gouda. Complete redundancy detection in firewalls. In *The 19^{th} Annual IFIP Conference on Data and Applications Security*, August 2005.

[20] E. Lupu and M. Sloman. Conflict analysis for management policies. In *IFIP/IEEE International Symposium on Integrated Network Management (IM'97)*, May 1997.

[21] A. Mayer, A. Wool, and E. Ziskind. Fang: A firewall analysis engine. In *IEEE Symposium on Security and Privacy (SSP'00)*, May 2000.