Cross-Domain Privacy-Preserving Cooperative Firewall Optimization

Fei Chen
Department of CSE
Michigan State University
East Lansing, MI
feichen@cse.msu.edu

Bezawada Bruhadeshwar
Center for Security Research
Institute of Information Technology
Hyderabad - 500032, India
bezawada@iiit.ac.in

Alex X. Liu
Department of CSE
Michigan State University
East Lansing, MI
alexliu@cse.msu.edu

Abstract—Firewalls have been widely deployed on the Internet for securing private networks. A firewall checks each incoming or outgoing packet to decide whether to accept or discard the packet based on its policy. Optimizing firewall policies is crucial for improving network performance. Prior work on firewall optimization focuses on either intra-firewall or inter-firewall optimization within one administrative domain where the privacy of firewall policies is not a concern. This paper explores inter-firewall optimization across administrative domains for the first time. The key technical challenge is that firewall policies cannot be shared across domains because a firewall policy contains confidential information and even potential security holes, which can be exploited by attackers. In this paper, we propose the first cross-domain privacy-preserving cooperative firewall policy optimization protocol. Specifically, for any two adjacent firewalls belonging to two different administrative domains, our protocol can identify in each firewall the rules that can be removed because of the other firewall. The optimization process involves cooperative computation between the two firewalls without any party disclosing its policy to the other. We implemented our protocol and conducted extensive experiments. The results on real firewall policies show that our protocol can remove as many as 49% of the rules in a firewall whereas the average is 19.4%. The communication cost is less than a few hundred KBs. Our protocol incurs no extra online packet processing overhead and the offline processing time is less than a few hundred seconds.

I. INTRODUCTION

A. Background and Motivation

Firewalls are critical in securing private networks of businesses, institutions, and home networks. A firewall is often placed at the entrance between a private network and the external network so that it can check each incoming or outgoing packet and decide whether to accept or discard the packet based on its policy. A firewall policy is usually specified as a sequence of rules, called Access Control List (ACL), and each rule has a predicate over multiple packet header fields (i.e., source IP, destination IP, source port, destination port, and protocol type) and a decision (i.e., accept and discard) for the packets that match the predicate. The rules in a firewall policy typically follow the first-match semantics where the decision for a packet is the decision of the first rule that the packet matches in the policy. Each physical interface of a router/firewall is configured with two ACLs: one for filtering outgoing packets and the other one for filtering incoming packets. In this paper, we use the terms firewalls, firewall policies, and ACLs interchangeably.

The number of rules in a firewall significantly affects its throughput. Fig. 1 shows the result of the performance test of iptables conducted by HiPAC [1]. It shows that increasing the number of rules in a firewall policy dramatically reduces the firewall throughput. Unfortunately, with the explosive growth of services deployed on the Internet, firewall policies are growing rapidly in size. Thus, optimizing firewall policies is crucial for improving network performance.

B. Limitation of Prior Work

Prior work on firewall optimization focuses on either intra-firewall optimization [7], [15], [16], [17], [18], [19], [20], [21] or inter-firewall optimization [3], [27] within one administrative domain where the privacy of firewall policies is not a concern. Intra-firewall optimization means optimizing a single firewall. It is achieved by either removing redundant rules [15], [17] or rewriting rules [7], [16], [18], [19], [20], [21]. Prior work on inter-firewall optimization requires two firewall policies without any privacy protection, and thus can only be used within one administrative domain. However, in reality, it is common that two firewalls belong to different administrative domains where firewall policies cannot be shared with each other. Keeping firewall policies confidential is very important for two reasons. First, a firewall policy may have security holes that can be exploited by attackers. Quantitative studies have shown that most firewalls are misconfigured and have security holes [25]. Second, a firewall policy often contains private information, e.g., the IP addresses of servers, which can be used by attackers to launch more precise and targeted attacks.

C. Cross-domain Inter-firewall Optimization

To our best knowledge, no prior work focuses on cross-domain privacy-preserving inter-firewall optimization. This paper represents the first step in exploring this unknown space. Specifically, we focus on removing inter-firewall policy redundancies in a privacy-preserving manner. Consider two adjacent firewalls 1 and 2 belonging to different administrative domains Net1 and Net2. Let FW1 denote the policy on firewall 1’s outgoing interface to firewall 2 and FW2 denote the policy on firewall 2’s incoming interface from firewall 1. For a rule r in FW2, if all the packets that match r but do not match any rule above r in FW2 are discarded by FW1, rule r can be removed because such packets never come to FW2. We call rule r an inter-firewall redundant rule with respect to FW1. Note that FW1 and FW2 only filter the traffic from FW1 to FW2; the traffic from firewall 2’s outgoing interface...
Clearly, all the packets that match port, destination port, protocol type, and decision, respectively.

The rules of the two firewall policies \( FW_1 \) and \( FW_2 \), that are used to filter the traffic flowing from CSE to EE, are listed in two tables following the format used in Cisco Access Control Lists. Note that SIP, DIP, SP, DP, PR, and Dec denote source IP, destination IP, source port, destination port, protocol type, and decision, respectively. Clearly, all the packets that match \( r_1 \) and \( r_2 \) in \( FW_2 \) are discarded by \( r_1^* \) in \( FW_1 \). Thus, \( r_1 \) and \( r_2 \) of \( FW_2 \) are inter-firewall redundant with respect to \( r_1^* \) in \( FW_1 \).

If this condition holds for all the predicates computed from rule \( r \), then rule \( r \) is redundant. To efficiently compute these predicates, we convert firewalls to firewall decision diagrams \([17]\). To allow the two firewalls to detect the redundant rules in \( FW_2 \) in a privacy-preserving manner, we develop a protocol so that two firewalls can detect such redundant rules without disclosing their policies to each other.

Our protocol applies to both stateful and stateless firewalls. The main difference between stateful and stateless firewalls is that stateful firewalls maintain a connection table. Upon receiving a packet, if it belongs to an established connection, it is automatically accepted without checking against the rules. Having the connection table or not does not affect our protocol.

E. Key Contributions

We make two key contributions. First, we propose a novel privacy-preserving protocol for detecting inter-firewall redundant rules in one firewall with respect to another firewall. This paper represents the first effort along this unexplored direction. Second, we implemented our protocol and conducted extensive experiments on both real and synthetic firewall policies. The results on real firewall policies show that our protocol can remove as many as 49% of the rules in a firewall whereas the average is 19.4%. The communication cost is less than a few hundred KBs. Our protocol incurs no extra online packet processing overhead and the offline processing time is less than a few hundred seconds.

II. Related Work

A. Firewall Redundancy Removal

Prior work on intra-firewall redundancy removal aims to detect redundant rules within a single firewall \([12]\), \([15]\), \([17]\). Gupta identified backward and forward redundant rules in a firewall \([12]\). Later, Liu et al. pointed out that the redundant rules identified by Gupta are incomplete, and proposed two methods for detecting all redundant rules \([15]\), \([17]\). Prior work on inter-firewall redundancy removal requires the knowledge of two firewall policies and therefore is only applicable within one administrative domain \([3]\), \([27]\).

B. Collaborative Firewall Enforcement in VPN

Prior work on collaborative firewall enforcement in virtual private networks (VPNs) aims to enforce firewall policies over encrypted VPN tunnels without leaking the privacy of the remote network’s policy \([6]\), \([13]\). The problems of collaborative firewall enforcement in VPNs and privacy-preserving inter-firewall optimization are fundamentally different. First, their purposes are different. The former focuses on enforcing a firewall policy over VPN tunnels in a privacy-preserving manner, whereas the latter focuses on removing inter-firewall redundant rules without disclosing their policies to each other. Second, their requirements are different. The former preserves the privacy of the remote network’s policy, whereas the latter preserves the privacy of both policies.

III. System and Threat Models

A. System Model

A firewall \( FW \) is an ordered list of rules. Each rule has a predicate over d fields \( F_1, \ldots, F_d \) and a decision for the packets that match the predicate. Firewalls usually check five fields: source IP, destination IP, source port, destination port, and protocol type. The length of these fields are 32, 32, 16, 16, and 8 bits, respectively. A predicate defines a set of packets
over the $d$ fields, and is specified as $F_1 \in S_1 \land \cdots \land F_d \in S_d$ where each $S_i$ is a subset of $F_i$’s domain $D(F_i)$. A packet over the $d$ fields $F_1, \cdots, F_d$ is a $d$-tuple $(p_1, \cdots, p_d)$ where each $p_i$ $(1 \leq i \leq d)$ is an element of $D(F_i)$. A packet $(p_1, \cdots, p_d)$ matches a rule $F_1 \in S_1 \land \cdots \land F_d \in S_d \rightarrow \text{(decision)}$ if and only if the condition $p_1 \in S_1 \land \cdots \land p_d \in S_d$ holds. Typical firewall decisions include accept, discard, accept with logging, and discard with logging. Without loss of generality, we only consider accept and discard in this paper. We call a rule with the accept decision an accepting rule and a rule with the discard decision a discarding rule. In a firewall policy, a packet may match multiple rules whose decisions are different. To resolve these conflicts, firewalls typically employ a first-match semantics where the decision for a packet $p$ is the decision of the first rule that $p$ matches. A matching set of $r_i$, $M(r_i)$, is the set of all possible packets that match the rule $r_i$ [15]. A resolving set of $r_i$, $R(r_i)$, is the set of packets that match $r_i$ but do not match any rule $r_j$ above $r_i$ $(j < i)$, and $R(r_i)$ is equal to $M(r_i) - M(r_1) \cup \cdots \cup M(r_{i-1})$ [15].

Based on above concepts, we define inter-firewall redundant rules. Given two adjacent firewalls $FW_1$ and $FW_2$, where the traffic flow is from $FW_1$ to $FW_2$, a rule $r$ in $FW_2$ is inter-firewall redundant with respect to $FW_1$ if and only if all the packets in $r$’s resolving set are discarded by $FW_1$.

B. Threat Model

We adopt the semi-honest model in [9]. For two adjacent firewalls, we assume that they are semi-honest, i.e., each firewall follows our protocol correctly but each firewall may try to reveal the policy of the other firewall. The semi-honest model is realistic and well adopted [4], [26]. For example, this model is appropriate for large organizations that have many independent branches as well as for loosely connected alliances composed by multiple parties. While we are confident that all administrative domains follow mandate protocols, we may not guarantee that no corrupted employees are trying to reveal the private firewall policies of other parties. We leave investigation of privacy-preserving firewall optimization in the model with malicious participants to future work.

IV. PRIVACY-PRESERVING INTER-FIREWALL REDUNDANCY REMOVAL

In this section, we present our privacy-preserving protocol for detecting inter-firewall redundant rules in $FW_2$ with respect to $FW_1$. To do this, we first converts each firewall to an equivalent sequence of non-overlapping rules. Because for any non-overlapping rule $nr$, the matching set of $nr$ is equal to the resolving set of $nr$, i.e., $M(nr) = R(nr)$, we only need to compare non-overlapping rules generated from the two firewalls for detecting inter-firewall redundancy. Second, we divide this problem into two subproblems, single-rule coverage redundancy detection and multi-rule coverage redundancy detection, and then propose our privacy-preserving protocol for solving each subproblem. A rule $nr$ is covered by one or more rules $nr_1', \cdots, nr_k'$ $(k \geq 1)$ if and only if $M(nr) \subseteq \bigcup M(nr_i')$. The first subproblem checks whether a non-overlapping rule $nr$ in $FW_2$ is covered by a non-overlapping discarding rule $nr'$ in $FW_1$, i.e., $M(nr) \subseteq M(nr')$. The second subproblem checks whether a non-overlapping rule $nr$ in $FW_2$ is covered by multiple non-overlapping discarding rules $nr_1' \cdots nr_k'$ $(k \geq 2)$ in $FW_1$, i.e., $M(nr) \subseteq M(nr_1') \cup \cdots \cup M(nr_k')$. Finally, after the redundant non-overlapping rules generated from $FW_2$ are identified, we map them back to the original rules in $FW_2$ and then identify the redundant ones.

The problem of checking whether $M(nr) \subseteq M(nr')$ boils down to the problem of checking whether one range $[a, b]$ in $nr$ is contained by another range $[a', b']$ in $nr'$, which further boils down to the problem of checking whether $a \in [a', b']$ and $b \in [a', b']$. Thus, we first describe the privacy-preserving protocol for comparing a number and a range.

A. Privacy-Preserving Range Comparison

To check whether a number $a$ from $FW_2$ is in a range $[a', b']$ from $FW_1$, we use a method similar to the prefix membership verification scheme in [13]. The basic idea is to convert the problem of checking whether $a \in [a', b']$ to the problem of checking whether two sets converted from $a$ and $[a', b']$ have a common element. Our method consists of four steps:

1. **Prefix conversion.** It converts $[a', b']$ to a minimum number of prefixes, denoted as $S([a', b'])$, whose union corresponds to $[a', b']$. For example, $S([11, 15]) = \{1011, 11**\}$.

2. **Prefix family construction.** It generates all the prefixes that contains $a$ including $a$ itself. This set of prefixes is called the prefix family of $a$, denoted as $F(a)$. Let $k$ be the bit length of $a$. The prefix family $F(a)$ consists of $k+1$ prefixes where the $i$-th prefix is obtained by replacing the last $i-1$ bits of $a$ by $\ast$. For example, as the binary representation of 12 is 1100, we have $F(12) = \{1100, 110*, 11**, 11***, 11****\}$. It is not difficult to prove that $a \in [a', b']$ if and only if $F(a) \cap S([a', b']) = \emptyset$.

3. **Prefix numericalization.** It converts the prefixes generated in the previous steps to concrete numbers such that we can encrypt them in the next step. We use the prefix numericalization scheme in [5]. Given a prefix $b_1b_2\cdots b_k*\cdots\ast$ of $w$ bits, we first insert 1 after $b_k$ and $\ast$, and then $1011$ and $1100$. If the prefix does not contain $\ast$, we place 1 at the end of the prefix. For example, $1100$ is converted to $11001$.

4. **Comparison.** It checks whether $a \in [a', b']$ by checking whether $F(a) \cap S([a', b']) = \emptyset$, which boils down to checking whether two numbers are equal. We use commutative encryption to do this checking in a privacy-preserving manner. Given a number $x$ and two encryption keys $K_1$ and $K_2$, a commutative encryption is a function that satisfies the property $(x_{K_1})_{K_2} = (x)_{(K_1)K_2}$, i.e., encryption with key $K_1$ first and then $K_2$ is equivalent to encryption with key $K_2$ first and then $K_1$. Example commutative encryption algorithms are the Pohlig-Hellman Exponentiation Cipher [22] and Secure RPC Authentication (SRA) [23]. In our scheme, each domain chooses a private key. Let $K_1, K_2$ be the private keys chosen by $Net_1$ and $Net_2$, respectively. To check whether number $v_1$ from $Net_1$ is equal to number $v_2$ from $Net_2$ without disclosing the value of each number to the other party, $Net_1$ can first encrypt $v_1$ using key $K_1$ and sends $(x)_{K_1}$ to $Net_2$; similarly, $Net_2$ can first encrypt $v_2$ using key $K_2$ and sends $(x)_{K_2}$ to $Net_1$. Then, each party checks whether $v_1 = v_2$ by checking whether $((v_1)_{K_1})_{K_2} = ((v_2)_{K_2})_{K_1}$. Note that $((v_1)_{K_1})_{K_2} = ((v_2)_{K_2})_{K_1}$ if and only if $v_1 = v_2$. If $v_1 \neq v_2$, neither party can learn anything about the numbers being compared. Fig. 3 illustrates the process of checking whether 12 from $FW_2$ is in the range $[11, 15]$ from $FW_1$. 
There is exactly one node that has no incoming edges. This is an acyclic and directed graph that has five properties. (a) $F W$ terminal

No two nodes on a decision path have the same label. Each $v$ node has a label, denoted $L(v)$. If $v$ is a non-terminal node, then $L(v) \in \{F_1, \ldots, F_d\}$. If $v$ is a terminal node, then $L(v)$ is a decision. (c) Each edge $e$, $u \rightarrow v$, is labeled with a non-empty set of integers, denoted $I(e)$, where $I(e) \subseteq D(F(u))$. (d) The set of all outgoing edges of a node $v$, denoted $E(v)$, satisfies two conditions: (a) consistency: $I(e) \cap I(e') = \emptyset$ for any two distinct edges $e$ and $e'$ in $E(v)$; (b) completeness: $\bigcup_{e \in E(v)} I(e) = D(F(v))$. (e) A directed path from the root to a terminal node is called a decision path. No two nodes on a decision path have the same label. Each path in the FDD corresponds to a non-overlapping rule. A full-length ordered FDD is an FDD where each decision path all fields appear exactly once and in the same order. For ease of presentation, we use the term “FDD” to denote “full-length ordered FDD”. An FDD construction algorithm, which converts a firewall policy to an equivalent FDD, is presented in [14]. Fig. 4(b) shows the FDD constructed from Fig. 4(a).

Net2 reduces the FDD’s size by merging isomorphic subgraphs. An FDD $f$ is reduced if and only if it satisfies two conditions: (a) no two nodes in $f$ are isomorphic; (b) no two nodes have more than one edge between them. Two nodes $v$ and $v'$ in an FDD are isomorphic if and only if $v$ and $v'$ satisfy one of the following two conditions: (a) both $v$ and $v'$ are terminal nodes with identical labels; (b) both $v$ and $v'$ are non-terminal nodes and there is a one-to-one correspondence between the outgoing edges of $v$ and the outgoing edges of $v'$ such that every two corresponding edges have identical labels and they both point to the same node. Fig. 4(c) shows the FDD reduced from the FDD in Fig. 4(b).

Net1 extracts non-overlapping discarding rules. We do not need to consider the non-overlapping accepting rules from $FW_1$ because the packets accepted by $FW_1$ are passed to $FW_2$. Note that a non-overlapping rule from $FW_2$ that is covered by these discarding rules from $FW_1$ is redundant. Fig. 4(d) shows the discarding non-overlapping rules extracted from the reduced FDD in Fig. 4(c).

Net3 further encrypts these prefixes with $K_2$ and sends them back to $Net_1$ as shown in Fig. 4(i).

Net1 reconstructs the non-overlapping discarding rules.
from the double encrypted prefixes because Net₁ knows which prefix belongs to which field of which rule.

(9) For each reconstructed non-overlapping rule, Net₁ assigns a distinct random index to it. These indices are used for Net₂ to identify the redundant non-overlapping rules from FW₂. For example, in Fig. 6(a), Net₁ assigns its three rules with three random indices: 27, 13, and 45. The detailed discussion is in Section IV-D.

C. Processing Firewall FW₂

In order to compare two firewalls in a privacy-preserving manner, Net₂ and Net₁ convert firewall FW₂ to d sets of double encrypted numbers, where d is the number of fields. The conversion of FW₂ includes five steps:

(1) Net₂ converts FW₂ to an equivalent all-match FDD. All-match FDDs differ from FDDs on terminal nodes. In an all-match FDD, each terminal node is labeled with a decision. For rule set of rule sequence numbers, whereas in an FDD each terminal node is labeled with a decision. For rule rᵢ (1 ≤ i ≤ n), we call i the sequence number of rᵢ. The set of rule sequence numbers labeled on a terminal node consists of the sequence numbers of all the rules that overlaps with the decision path ending with this terminal node. Given a decision path P, (v₁e₁, v₂e₂, ..., vₙ₋₁eₙ₋₁, vₙeₙ), the matching set of P is defined as the set of all packets that satisfy F(v₁)∧F(e₁)∧...∧F(vₙ)∧F(eₙ) = I(eₙ). We use M(P) to denote the matching set of P. More formally, in an all-match FDD, for any decision path P : (v₁e₁, v₂e₂, ..., vₙ₋₁eₙ₋₁, vₙeₙ), if M(P) ∩ M(rᵢ) ≠ ∅, then M(P) ⊆ M(rᵢ) and i ∈ F(vₙ₊₁). Fig. 4(b) shows the all-match FDD generated from Fig. 4(a). Considering the terminal node of the fourth path in Fig. 4(b), its label {2, 4} means that M(P) ⊆ M(r₂), M(P) ⊆ M(r₄), M(P) ∩ M(r₁) = ∅, and M(P) ∩ M(r₃) = ∅. An all-match FDD not only represents a firewall in a non-overlapping fashion but also represents the overlapping relationship among rules. The reason of converting FW₂ to an all-match FDD is that later Net₂ needs the rule sequence numbers to identify inter-firewall redundant rules in FW₂.

(2) Net₂ extracts all non-overlapping rules from the all-match FDD. Fig. 5(c) shows the non-overlapping rules extracted from Fig. 5(b). For each range [a, b] of a non-overlapping rule, Net₂ generates two prefix families F(a) and F(b). Fig. 5(d) shows the result from Fig. 5(c).

(3) For every field Fᵢ, Net₂ unifies all prefix families of all the non-overlapping rules into one prefix set and permutes the prefixes. Considering the first field F₁ in Fig. 5(d), Net₂ unifies F(0), F(2), F(3), F(5), F(6), and F(15) to the first prefix set in Fig. 5(e). The benefits of this step are similar to those of Step (5) in FW₁’s conversion. In terms of efficiency, it avoids encrypting and sending the duplicate prefixes for each field, and hence, significantly reduces the computation and communication costs. In terms of security, Net₁ cannot reconstruct the non-overlapping rules of FW₂, because it Net₁ does not know which prefix belongs to which rule in FW₂. However, Net₂ knows such information, which will be used to identify redundant non-overlapping rules later. Note that, the ordering of the fields cannot be permuted because Net₁ needs to perform comparison of the prefixes with only those prefixes from the corresponding fields.

(4) Net₂ numerizes and encrypts the prefixes using its private K₂, and sends to Net₁. Fig. 5(f) shows the prefixes.

(5) Net₁ further encrpytes these prefixes using its key K₁.

D. Single-Rule Coverage Redundancy Detection

After processing the two firewalls, Net₁ has a sequence of double encrypted non-overlapping rules obtained from FW₁ and d sets of double encrypted numbers obtained from FW₂. Let (F₁ ∈ S₁) ∧ ... ∧ (Fₖ ∈ Sₖ) → discard denote a double encrypted rule, where Sᵢ is a set of double encrypted numbers. Let S₁, ..., Sₖ denote the d sets of double encrypted numbers from FW₂. Fig. 6(a) shows the double encrypted non-overlapping rules generated from Fig. 4 and Fig. 6(b) shows the double encrypted numbers generated from Fig. 5. For each field Fᵢ (1 ≤ i ≤ d) and for each number a in Sᵢ, Net₁ checks whether there exists a double encrypted rule (F₁ ∈ S₁) ∧ ... ∧ (Fₖ ∈ Sₖ) → discard such that a ∈ Sᵢ. If rule rᵢ satisfies this condition, then Net₁ associates the rule index i with a. As there maybe multiple rules that satisfy this condition, eventually Net₁ associates a set of rule indices with a. If no rule satisfies this condition, Net₁ associates an empty set with a. Considering the number (010111, K₁), the only rule with index 13 contains it because (010111, K₁) = (010111, K₁); thus, Net₁ associates (010111, K₁) with {13}. Finally, Net₁ replaces each number in S₁, ..., Sₖ with its corresponding set of rule indices, and sends them to Net₂.
Upon receiving the sets from $Net_1$, for each prefix family, $Net_2$ finds the index of the rule that overlaps with the prefix family. For a non-overlapping rule $nr$ from $FW_2$, if all its prefix families overlap with the same discarding rule $nr'$ from $FW_1$, $nr$ is covered by $nr'$ and hence, $nr$ is redundant. For example, in Fig. 6(d), $nr_1$ is redundant, because $F(0), F(2), F(0)$ and $F(6)$ overlap with rule 27 from $FW_1$. Similarly, $nr_2$ is redundant. Note that $F(v)$: $j_1, ..., j_k$ denotes that $F(v)$ overlaps with non-overlapping rules $j_1, ..., j_k$ from $FW_1$.

E. Multi-Rule Coverage Redundancy Detection

To detect multi-rule coverage redundancy, $Net_2$ keeps track of partial overlapping between its non-overlapping rules and those from $FW_1$. A non-overlapping rule $nr$ from $FW_2$ is partial overlapping with a non-overlapping rule $nr'$ from $FW_1$ if and only if $M(nr) \cap M(nr') \neq \emptyset$. Let $[a_1, b_1] \cap \cdots \cap [a_d, b_d]$ and $[a_1', b_1'] \cap \cdots \cap [a_d', b_d']$ denote the predicates of $nr$ and $nr'$, respectively. The condition $M(nr) \cap M(nr') \neq \emptyset$ holds if and only if for any field $F_k$ $(1 \leq k \leq d)$, one of the three conditions holds: (1) $a_k \in [a_k', b_k']$; (2) $b_k \in [a_k', b_k']$; (3) $a_k, b_k \in [a_k', b_k']$. The first two conditions are easy to check for $Net_2$ based on the privacy-preserving range comparison scheme in Section IV-A. Checking the third condition can be done by reversing the roles of $Net_1$ and $Net_2$ in the single-rule coverage redundancy detection. Considering the comparison result in Fig. 6(d), $Net_2$ knows that $nr_4$ is partial overlapping with two rules 27 and 13 in Fig. 5(j). Using this information, for each non-overlapping rule $nr$ from $FW_1$ that overlaps with multiple non-overlapping rules from $FW_1$, $Net_2$ asks $Net_1$ to compute a new set of rules by combining these non-overlapping rules from $FW_1$. For example, after receiving the list with indices 27 and 13, $Net_1$ computes a new rule $nr'_{new}$: $F_1 \in [0, 7] \land F_2 \in [13, 15] \rightarrow d$ by combining these two rules, $F_1 \in [0, 4] \land F_2 \in [13, 15] \rightarrow d$ and $F_1 \in [5, 15] \land F_2 \in [5, 15] \rightarrow d$. If no new rule exists, $nr$ is not multi-rule coverage redundant. Otherwise, both parties run the single-rule redundancy detection protocol again to check whether $nr$ is covered by a new rule. If true, rule $nr$ is multi-rule coverage redundant. For example, after comparing $nr_4$ with $nr'_{new}$, $Net_2$ can identify that $nr_4$ is redundant.

Note that, if the protocol is executed with the same keys in both single-rule and multi-rule redundancy detection, one firewall could learn considerable information of the other by comparing the new reconstructed firewall with that generated in the single-rule redundancy detection. To avoid this problem, $Net_1$ and $Net_2$ choose two new keys $K_1'$ and $K_2'$ in the multi-rule redundancy detection. Further note that in the multi-rule redundancy detection, the number of rules to be compared could be much smaller than those in the single-rule redundancy detection. Thus, the computation and communication costs are much less than those in the single-rule redundancy detection.

F. Identification and Removal of Redundant Rules

After single-rule and multi-rule coverage redundancy detection, $Net_2$ identifies the redundant non-overlapping rules in $FW_2$. Next, $Net_2$ needs to identify which original rules are inter-firewall redundant. As each path in the all-match FDD of $FW_2$ corresponds to a non-overlapping rule, we call the paths that correspond to the redundant non-overlapping rules redundant paths and the remaining paths effective paths. For example, in Fig. 7, the dashed paths are the redundant paths that correspond to $nr_1, nr_2$ and $nr_3$ in Fig. 5(c), respectively. Finally, $Net_2$ identifies redundant rules based on Theorem 4.1.

**Theorem 4.1**: Given firewall $FW_2' : \{r_1, \ldots, r_m\}$ with no intra-firewall redundancy and its all-match FDD, rule $r_i$ is inter-firewall redundant with respect to $FW_1$ if and only if two conditions hold: (1) there is a redundant path whose terminal node contains sequence number $i$; (2) there is no effective path whose terminal node contains $i$ as the smallest element.

Proof: Let $\{P_1, \ldots, P_m\}$ denote all paths in $FW_2$’s all-match FDD. According to the theorems in [15], [17], the resolving set of each rule $r_i$ $(1 \leq i \leq n)$ in firewall $FW_2$ satisfies the condition $R(r_i) = \cup_{k=1}^m M(P_{j_k}) (1 \leq k \leq m)$, where $P_{j_k}, \ldots, P_{j_{k'}}$ are all the paths whose terminal nodes contain $i$ as the smallest element. Based on the definition of inter-firewall redundant rules in Section III-A, rule $r_i$ is inter-firewall redundant if and only if all the packets in $\cup_{k=1}^m M(P_{j_k})$ are discarded by $FW_1$. Thus, each path $P_{j_k}$ $(1 \leq k \leq l)$ is a redundant path. In other words, all the paths $P_{j_1}, \ldots, P_{j_l}$ whose terminal nodes contain $i$ as the smallest element are redundant paths.

Considering redundant paths in Fig. 7, $Net_2$ identifies that $r_1$ and $r_2$ are inter-firewall redundant with respect to $FW_1$.

**Theorem 4.2**: The privacy-preserving inter-firewall redundancy removal protocol is a complete inter-firewall redundancy removal scheme.

Proof: Suppose that our proposed protocol is not a complete scheme and hence it cannot detect all inter-firewall redundant rules in $FW_2$. Assume that rule $r_i$ is inter-firewall redundant in $FW_2$ but it is not detected by our protocol. According to Theorem 4.1, $R(r_i) = \cup_{k=1}^m M(P_{j_k}) (1 \leq k \leq m)$ and $P_{j_1}, \ldots, P_{j_l}$ are redundant paths in $FW_2$’s all-match FDD. Thus, some paths in $\{P_{j_1}, \ldots, P_{j_l}\}$ cannot be identified as redundant paths by our protocol. This conclusion violates the fact that our protocol can identify all the redundant paths in $FW_2$’s all-match FDD.
V. FIREWALL UPDATE AFTER OPTIMIZATION

If FW_1 or FW_2 changes after inter-firewall optimization, the inter-firewall redundant rules identified by the optimization may not be inter-firewall redundant anymore. In this section, we discuss our solution to address firewall update. There are five possible cases under this scenario.

(1) Net_1 changes the decisions of some rules from discard to accept in FW_1. In this case, Net_1 needs to notify Net_2 that which non-overlapping rules (indices of these rules) from FW_1 are changed. Using this information, Net_2 checks if there were any rules in FW_2 that were removed due to these rules, and then adds the affected rules back into FW_2.

(2) Net_1 changes the decisions of some rules from accept to discard in FW_1. In this case, Net_2 can run our cooperative optimization protocol again to identify more inter-firewall redundant rules in FW_2.

(3) Net_2 changes the decisions of some rules in FW_2. In this case, neither party needs to take actions because the inter-firewall redundancy detection does not consider the decisions of the rules in FW_2.

(4) Net_1 adds or removes some rules in FW_1. In this case, since the resolving sets of some rules in FW_1 may change, a rule in FW_2 that used to be inter-firewall redundant maybe not redundant anymore. It is important for Net_2 to run our cooperative optimization protocol again.

(5) Net_2 adds or removes some rules in FW_2. Similar to the fourth case, since the resolving sets of some rules in FW_2 may change, it is important for Net_2 to run our protocol again.

VI. SECURITY AND COMPLEXITY ANALYSIS

A. Security Analysis

To analyze the security of our protocol, we first describe the commutative encryption and its properties. Let Key denote a set of private keys and Dom denote a finite domain. A commutative encryption f is a computable function f: Key x Dom → Dom that satisfies the following four properties.

(1) Secrecy: For any x and key K, given (x)_K, it is computationally infeasible to compute K. (2) Commutativity: For any x, K_1, and K_2, we have (x)_K_1 K_2 = (x)_{K_2 K_1}. (3) For any x, y, and K, if x ≠ y, we have (x)_K ≠ (y)_K. (4) The distribution of (x)_K is indistinguishable from the distribution of x. Note that, for ease of presentation, in the rest of this section, any x, y, or z is an element of Dom, any K, K_1, or K_2 is an element of Key, and (x)_K denotes f(x, K).

In the conversion of FW_1, for each non-overlapping rule nr’ from FW_1, let V_Fj(nr’) denote the prefix set for the field F_j, e.g., in Fig. 4(e), V_Fj(nr’) denotes {00**, 0100}. In the conversion of FW_2, let T_Fj denote the prefix set for the field F_j after removing duplicate prefixes, e.g., in Fig. 5(e), T_Fj denotes the first prefix set. Our cooperative optimization protocol essentially compares V_Fj(nr’) and T_Fj in a privacy-preserving manner. We have the following theorem.

Theorem 6.1: If both parties Net_1 and Net_2 are semi-honest, after comparing two sets V_Fj(nr’) and T_Fj using our protocol, Net_1 learns only the size |T_Fj| and the intersection V_Fj(nr’) ∩ T_Fj, and Net_2 learns only the size |V_Fj(nr’)|. The protocol guarantees that neither party learns any information about the other party’s set.

Proof: According to the theorems in multi-party secure computation [2], [8], if we can prove that the distribution of the Net_1’s view of our protocol cannot be distinguished from a simulation that uses only V_Fj(nr’), V_Fj(nr’) ∩ T_Fj, and |T_Fj|, then Net_1 cannot learn anything else except V_Fj(nr’) ∩ T_Fj, and |T_Fj|. Notice that Net_1’s view of our protocol is the information that Net_1 gains from FW_2.

Without loss of generality, we only prove that Net_1 learns only the size |T_Fj| and the intersection V_Fj(nr’) ∩ T_Fj. The simulator for Net_1 uses key K_1 to create a set from V_Fj(nr’) and V_Fj(nr’) ∩ T_Fj as follows:

\[ Y_S = \{(x_1)_{K_1}, \ldots, (x_m)_{K_1}, z_{m+1}, \ldots, z_n \} \]

where \( z_{m+1}, \ldots, z_n \) are random values generated by the simulator and they are uniformly distributed in the finite domain Dom. According to the theorems in [2], Net_1 cannot distinguish the distribution of \( Y_S \)'s elements from that in:

\[ Y_R = \{(x_1)_{K_1}, \ldots, (x_m)_{K_1}, (z_{m+1})_{K_1}, \ldots, (z_n)_{K_1} \} \]

The Net_1’s view of our protocol corresponds to \( Y_R \). Therefore, the distribution of the Net_1’s view of our protocol cannot be distinguished from this simulation.

Next, we analyze the information learned by Net_1 and Net_2. After implementing our protocol, Net_1 knows the converted firewalls of FW_1 and FW_2, e.g., Fig. 4(j) and Fig. 5(g), and Net_2 knows the comparison result, e.g., Fig. 6(d). On Net_1 side, for each field F_j (1 ≤ j ≤ d), it knows only |T_Fj| and \( V_Fj(nr')\setminus T_Fj \), and it cannot reveal the rules of FW_2 for two reasons. First, in \( V_Fj(nr')\setminus T_Fj \), the prefix of each rule that is generated from many different numbers. For example, if a prefix of IP addresses (32 bits) \( b_1 b_2 \ldots b_k \) can be generated from \( 232−k \) different IP addresses. Second, even if Net_1 finds the number for a prefix in \( V_Fj(nr')\setminus T_Fj \), Net_1 doesn’t know which rule in FW_2 contains that number. On Net_2 side, it only knows that the prefix x in \( F(x) \) belongs to which non-overlapping rules in FW_1. But such information is not enough to reveal the rules in FW_1.

B. Complexity Analysis

Let \( n_1 \) and \( n_2 \) be the number of rules in two adjacent firewalls FW_1 and FW_2, respectively, and \( d \) be the number of fields in both firewalls. For simplicity, we assume that the numbers in different fields have the same length, say \( w \) bits.

We first analyze the computation, space, and communication costs for the conversion of FW_1. Based on the theorem in [14], the maximum number of non-overlapping rules generated from the FDD is \((2n_1−1)^d\). Each non-overlapping rule consists of \( d \) w-bit intervals and each interval can be converted to at most \( 2w−2 \) prefixes. Thus, the maximum number of prefixes generated from these non-overlapping rules is \( d(2w−2)(2n_1−1)^d \). Note that the total number of prefixes cannot exceed \( 2^w+1 \) because Net_1 puts all prefixes into one set. Thus, the computation cost of encryption by Net_1 is \( \min(d(2w−2)(2n_1−1)^d, 2^w+1) \). Therefore, for the conversion of FW_1, the computation cost of Net_1 is \( \min(O(dw^2), O(n_1^d + 2^w)) \), the space cost of Net_1 is \( O(dw^2) \), the communication cost is \( \min(O(dw^2), O(2^w)) \), and the computation cost of Net_2 is \( \min(O(dw^2), O(2^w)) \).

Similarly, for the conversion of FW_2, the computation cost of Net_2 is \( \min(O(n_2^d + 2^w)) \), the space cost of Net_2 is \( O(dw^2) \), the communication cost is \( \min(O(n_2^d), O(2^w)) \), and the computation cost of Net_1 is \( \min(O(dw^2), O(2^w)) \).
VII. EXPERIMENTAL RESULTS

We evaluate the effectiveness of our protocol on real firewalls and evaluate the efficiency of our protocol on both real and synthetic firewalls. We implemented our protocol using Java 1.6.0. Our experiments were carried out on a PC running Linux with 2 Intel Xeon cores and 16GB of memory.

A. Evaluation Setup

We conducted experiments over five groups of two real adjacent firewalls. Each firewall examines five fields, source IP, destination IP, source port, destination port, and protocol. The number of rules ranges from dozens to thousands. In implementing the commutative encryption, we used the Pohlig-Hellman algorithm [22] with a 1024-bit prime modulus and 160-bit encryption keys. To evaluate the effectiveness, we conducted our experiments over these five groups of adjacent firewalls. To evaluate the efficiency, for two firewalls in each group, we measured the processing time, the comparison time, and the communication cost of both parties.

Due to security concerns, it is difficult to obtain a large number of real adjacent firewalls. To further evaluate the efficiency, we generated a large number of synthetic firewalls based on Singh et al.’s method [24]. The synthetic firewalls also examine the same five fields as real firewalls. The number of rules in the synthetic firewalls ranges from 200 to 2000, and for each number, we generated 10 synthetic firewalls. To measure the efficiency, we first processed each synthetic firewall as $FW_1$ and then measured the processing time and communication cost of two parties. Second, we processed each synthetic firewall as $FW_2$ and measured the processing time and communication cost. Third, we measured the comparison time for every two synthetic firewalls. We did not evaluate the effectiveness of our protocol on synthetic firewalls because they are generated randomly and independently without considering whether two firewalls are adjacent or not.

B. Methodology

In this section, we define the metrics to measure the effectiveness of our protocol. Given our firewall optimization algorithm $A$, and two adjacent firewalls $FW_1$ and $FW_2$, we use $A(FW_1, FW_2)$ to denote a set of inter-firewall redundant rules in $FW_2$. Let $|FW|$ denote the number of rules in $FW$ and $|A(FW_1, FW_2)|$ denote the number of inter-firewall redundant rules in $FW_2$. To evaluate the effectiveness, we define a redundancy ratio $\beta(A(FW_1, FW_2)) = \frac{|A(FW_1, FW_2)|}{|FW_2|}$. This ratio $\beta(A(FW_1, FW_2))$ measures what percentage of rules are inter-firewall redundant in $FW_2$.

C. Effectiveness and Efficiency on Real Firewall Policies

Table I shows the redundancy ratios for 5 real firewall groups. Column 1 shows the names of five real firewall groups. Columns 2 and 3 show the names of firewalls $FW_1$ and $FW_2$, respectively. Column 4 shows the number of rules in firewall $FW_2$. Fig. 8 shows the processing time and communication cost of two parties $Net_1$ and $Net_2$ when processing $FW_1$; Fig. 9 shows the processing time and communication cost of the two parties when processing $FW_2$. Fig. 12 shows the comparison time of each group.

| Group         | $FW_1$ | $FW_2$ | $|FW_2|$ | redundancy ratio |
|---------------|--------|--------|---------|-----------------|
| Econ          | Econ1  | Econ2  | 129     | 17.1%           |
| Host          | Host1  | Host2  | 139     | 17.1%           |
| Wan           | Wan1   | Wan2   | 511     | 1.0%            |
| Ath           | Ath1   | Ath2   | 830     | 14.4%           |
| Comp          | Comp1  | Comp2  | 3928    | 14.4%           |

TABLE I

REDUNDANCY RATIOS FOR 5 REAL FIREWALL GROUPS

Our protocol achieves significant redundancy ratio on four firewall groups. For 5 real firewall groups, our protocol achieves an average redundancy ratio of 19.4%. Particularly, for the firewall group Host, our protocol achieves 49.6% redundancy ratio, which implies that almost half of rules in Host2 are inter-firewall redundant rules. For firewall groups Econ, Ath, and Comp, our protocol achieves 14.4%-17.1% redundancy ratios, which implies that about 15% of rules in $FW_2$ are redundant in these three groups. Only for one firewall group Wan, our protocol achieves 1.0% redundancy ratio. From these results, we observed that most adjacent real firewalls, have many inter-firewall redundant rules. Thus, our protocol could effectively remove inter-firewall redundant rules and significantly improve the network performance.

Our protocol is efficient for processing and comparing two real firewalls. When processing $FW_1$ in the 5 real firewall
groups, the processing time of $Net_1$ is less than 2 seconds and the processing time of $Net_2$ is less than 1 second. When processing $FW_2$ in those real firewall groups, the processing time of $Net_1$ is less than 4 seconds and the processing time of $Net_2$ is less than 10 seconds. The comparison time of two firewalls is less than 0.07 seconds. The total processing time of two parties is less than 15 seconds, which demonstrates the efficiency of our protocol.

Our protocol is efficient for the communication cost between two parties. When processing firewall $FW_1$ in the 5 real firewall groups, the communication cost from $Net_1$ to $Net_2$ and that from $Net_2$ to $Net_1$ are less than 60 KB. Note that the communication cost from $Net_1$ to $Net_2$ and that from $Net_2$ to $Net_1$ are the same because $Net_1$ and $Net_2$ encrypt the same number of values and the encrypted values have the same length, i.e., 1024 bits in our experiments. When processing $FW_2$ in those real firewall groups, the communication cost from $Net_2$ to $Net_1$ is less than 300 KB. The total communication cost between two parties is less than 500 KB, which can be sent through the current network (e.g., DSL network) around 10 seconds.

**D. Efficiency on Synthetic Firewall Policies**

For the synthetic firewalls, Fig. 10 and Fig. 11 show the average processing time and communication cost of two parties $Net_1$ and $Net_2$ for processing $FW_1$ and $FW_2$, respectively. Fig. 13 shows the average comparison time for every two synthetic firewalls. Note that the vertical axis of two figures 10(a) and 11(a) are in a logarithmic scale.

Our protocol is efficient for processing and comparing two synthetic firewalls. When processing the synthetic firewalls as $FW_1$, the processing time of $Net_1$ is less than 400 seconds and the processing time of $Net_2$ is less than 5 seconds. When processing the synthetic firewalls as $FW_2$, the processing time of $Net_1$ is less than 400 seconds and the processing time of $Net_2$ is less than 20 seconds. The comparison time of two synthetic firewalls is less than 4 seconds.

**Our protocol is efficient for the communication cost between two synthetic firewalls.** When processing the synthetic firewalls as $FW_1$, the communication cost from $Net_1$ to $Net_2$ and that from $Net_2$ to $Net_1$ grows linearly with the number of rules in $FW_1$, and both costs are less than 450 KB. Similarly, when processing synthetic firewalls as $FW_2$, the communication cost from $Net_2$ to $Net_1$ grows linearly with the number of rules in $FW_2$, and the communication cost from $Net_2$ to $Net_1$ is less than 1600 KB.

**VIII. CONCLUSIONS AND FUTURE WORK**

In this work, we identified an important problem, cross-domain privacy-preserving inter-firewall redundancy detection. We propose a novel privacy-preserving protocol for detecting such redundancy. We implemented our protocol in Java and conducted extensive evaluation. The results on real firewall policies show that our protocol can remove as many as 49% of the rules in a firewall whereas the average is 19.4%.

There are many special cases that could be explored based on our current protocol. For example, there may be hosts or Network Address Translation (NAT) devices between two adjacent firewalls. Our current protocol cannot be directly applied to such cases. Extending our protocol to these cases could be an interesting topic, and requires further investigation.

**Acknowledgment**

This material is based in part upon work supported by National Science Foundation under Grant Numbers CNS-1017598.

**REFERENCES**


