Automated Synthesis of Distributed Network Access Controls: A Formal Framework with Refinement

Mohammad Ashiqur Rahman and Ehab Al-Shaer
Department of Software and Information Systems
University of North Carolina at Charlotte, United States
Email: {mrahman4, ealshaer}@uncc.edu

Abstract—Due to the extensive use of network services and emerging security threats, enterprise networks deploy varieties of security devices for controlling resource access based on organizational security requirements. These requirements need fine-grained access control rules based on heterogeneous isolation patterns like access denial, trusted communication, and payload inspection. Organizations are also seeking for usable and optimal security configurations that can harden the network security within enterprise budget constraints. In order to design a security architecture, *i.e.*, the distribution of security devices along with their security policies, that satisfies the organizational security requirements as well as the business constraints, it is required to analyze various alternative security architectures considering placements of network security devices in the network and the corresponding access controls. In this paper, we present an automated formal framework for synthesizing network security configurations. The main design alternatives include different kinds of isolation patterns for network traffic flows. The framework takes security requirements and business constraints along with the network topology as inputs. Then, it synthesizes cost-effective security configurations satisfying the constraints and provides placements of different security devices, optimally distributed in the network, according to the given network topology. In addition, we provide a hypothesis testing-based security architecture refinement mechanism that explores various security design alternatives using ConfigSynth and improves the security architecture by systematically increasing the security requirements. We demonstrate the execution of ConfigSynth and the refinement mechanism using case studies. Finally, we evaluate their scalability using simulated experiments.

Index Terms—Security configuration; automatic synthesis; formal modeling; security metrics; isolation.

1 Introduction

Organizational security requirements are becoming very complex due to extensive use of various network services and emerging security threats. In addition, most organizations are not only emphasizing the enforcement of the security requirements but also requiring satisfaction of different business constraints on usability and security deployment cost. Providing a strong security in a network by exploring different security design alternatives while resolving the contention between the security and business constraints is important as well as challenging.

The organizational security requirements are usually ensured by establishing necessary isolation measures between the hosts. There are different isolation patterns which are defined based on different security devices and their capabilities. An isolation pattern signifies the type of security resistance, such as traffic filtering (firewall), trusted communication (IPSec), payload traffic inspection (IDS), and hiding traffic source identity (NAT/Proxy).

Any security design has to satisfy the business constraints of the organization, which are represented mainly in terms of usability and deployment cost. The implementation of isolation measures affects these constraints. While different security devices provide different levels of isolations, their impacts on the usability also often differ. For example, the use of firewall-based access denial gives no usability and the use of IPSec-based isolation pattern often reduce the usability by causing some applications to be inaccessible to a host. Similarly, the placement of security devices in the network according to the chosen isolation measures is crucial for deployment cost. The deployment of a security device incurs cost and each isolation pattern cannot thus correspond to the deployment of one or more security devices. The routing paths for multiple traffic often share one or more links and this sharing makes it possible to deploy necessary security devices such that they are optimally distributed in the network satisfying (*i.e.*, implementing) the security measures. Therefore, it is required to find the best security isolation design at an affordable cost that maintains the security and usability within an expected level.

In this paper, we present an automated framework for the security architecture synthesis using constraint satisfaction checking. We name this framework *ConfigSynth*. The framework takes the network's topology, isolation requirements, and usability and business constraints as inputs, and formulates a model for the security design synthesis. The model is solved using Satisfiability Modulo Theories (SMT) [1] and the solution provides security configurations with various isolation patterns in different segments of the network, as well as physical placements of security devices. ConfigSynth is a novel framework that incorporates

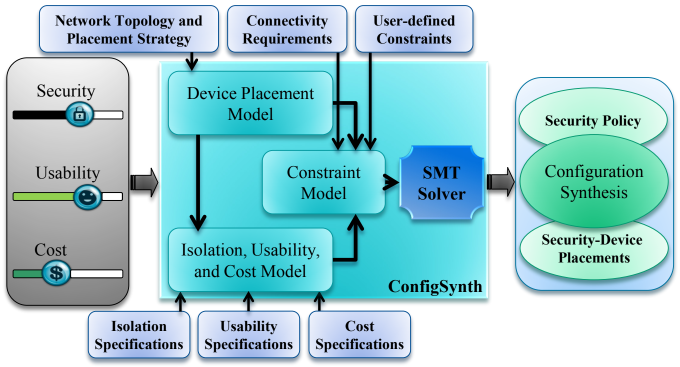


Fig. 1. The architecture of the ConfigSynth framework. The architecture shows the inputs to ConfigSynth and the outputs generated by it.

TABLE 1
Various Notations used in Formal Modeling

Notation	Type	Definition
$g(i,j)$	Boolean	Traffic flow from the host i to the host j under the service g.
$y^k_{i,j}(g)$	Boolean	The k^{th} isolation pattern to be implemented for the flow $g(i,j)$.
$x^d_{i,j}(g)$	Boolean	The d^{th} (type of) security device to be deployed on the routing path of the flow $g(i,j)$.
$L^k_{i,j}(g)$	Number	Isolation score of the k^{th} isolation pattern when it is implemented for the flow $g(i,j)$.
$c_{i,j}(g)$	Boolean	The flow $g(i,j)$ must be allowed.
$a_{i,j}(g)$	Number	Rank or demand of the flow $g(i,j)$.
$b^k_{i,j}(g)$	Number	Usability of the flow $g(i,j)$ when the k^{th} isolation pattern is implemented.
$l^d_{i,j,z,t}$	Boolean	A security device of the type d to be deployed on the link $l_{i,j,z,t}$, *i.e.*, the t^{th} link on the z^{th} routing path from the host i to the host j.

the security device placements in the network topology within the design in order to model the optimal distribution of security devices within the deployment budget. In [2], we presented ConfigSynth and the preliminary results. In this version, we extend the framework by introducing a refinement mechanism that adapts the idea of hypothesis testing to find an improved security architecture in a scalable manner. The framework can be used as a decision support system to create optimal security configurations for a network by exploring different design alternatives. Our evaluation results show that ConfigSynth can efficiently synthesize necessary security configurations for a network with hundreds of hosts.

The rest of this paper is organized as follows: The architecture of ConfigSynth, the corresponding formal models for the security design synthesis, and its implementation are described in Section 2. The security architecture refinement mechanism is presented in Section 3. The evaluation results regarding ConfigSynth and the refinement mechanism are presented in Section 4. The related works are discussed in Section 6. Finally, conclusions are drawn in Section 7.

2 Security Design Synthesis Model

We start this section by presenting the architecture of ConfigSynth. Then we describe the formal modeling of the security design synthesis.

2.1 ConfigSynth Architecture

ConfigSynth follows a top-down security design automation approach in which the high level requirements disaggregated into fine-grained security configurations. The architecture of ConfigSynth is shown in Fig. 1. The synthesis framework follows three main steps: (i) taking necessary inputs, (ii) formally modeling the security architecture synthesis problem according to the inputs, and (iii) encoding the model into SMT logics and solving it using an SMT solver to determine the security architecture.

ConfigSynth takes the following as its main inputs: (i) the network topology, (ii) security (isolation) requirements, and (iii) business (usability and deployment cost) constraints. The tool provides its user with three sliders in order to select the requirements/constraints on the isolation measure taken in the network, the usability of the system, and the cost for deploying necessary security devices. The sliders are scaled from 0 to some upper limit (*e.g.*, 10). The tool also takes partial or complete specifications about the qualities of isolation patterns, the demands of different flows, and the deployment costs of different security devices. The isolation requirements are conditioned on the specifications of different isolation patterns along with their relative order based on their capabilities. ConfigSynth models the functional mapping from each flow to an isolation decision variable. Then, it calculates the overall isolation of the network by accumulating isolation measures between different host pairs under various services.

The usability is modeled based on the connectivity requirements and the ranks of the service flows as provided in the specifications. The connectivity requirements are modeled as a set of rules, where each rule functionally maps a flow to a decision variable. We model impacts of different isolation patterns on the usability. The implementation of an isolation measure is associated with a cost. The cost depends on the security devices (*i.e.*, device types and their numbers) that are required for implementing the isolation patterns. The number of security devices depends on the network topology. ConfigSynth also models different invariant and user-defined constraints on selecting the security configurations. ConfigSynth formalizes the security design synthesis problem as the satisfactions of all the constraints with regards to isolation requirements and business constraints. ConfigSynth solves this security design synthesis problem using Z3, a powerful SMT solver [3]. The solution determines the isolation pattern for each service flow in the network, such that the overall isolation in the network and the usability of the system satisfy the associated requirements/constraints, while the cost for security deployment does not exceed the budget.

2.2 Modeling of Network Topology

ConfigSynth models the network topology as a graph. The network topology is modeled as the tuple $\langle \mathbb{N}, \mathbb{L} \rangle$, where $\mathbb{N}$ is a finite set of network nodes, including hosts ($\mathbb{H}$) and routers ($\mathbb{R}$), and $\mathbb{L}$ is a finite set of links. Each link physically connects two nodes and, thus, $\mathbb{L} \subseteq \mathbb{N} \times \mathbb{N}$. It is assumed that there can be at most a single link between a specific node pair. Each host is identified by an ID (*e.g.*, IP address). A host may execute one or more services, which are accessed by different hosts. A service is denoted using $g \in \mathbb{G}$, where

$\mathbb{G}$ is the set of all services. The term $g(i,j)$ defines the flow between a pair of hosts $\{i,j\}$, where i is the source and j is the destination, under a service g. Two hosts can communicate through multiple services.

Each host pertains a set of running network services. The organizational connectivity requirement is defined on these services. The network topology model will be used in determining the optimal placement of the security devices which has been presented in Section 2.5.2.

2.3 Modeling of Isolation

We define *isolation* as the *restriction* on the connectivity, *i.e.*, network communication. The communication between two hosts can be restricted applying different security devices or systems, such as firewall, IPSec, IDS, NAT, etc. For example, a firewall can be placed to simply block some traffic flows (*i.e.*, complete isolation), while IPSec can be placed to ensure authenticated transmission for the allowed flows (*i.e.*, restriction based on authorization). Both of these devices are required to ensure authenticated and controlled flow of different traffics. In order to formalize isolation, it is required to define different isolation patterns, considering different kinds of security devices, the levels of restrictions they can enforce on the flows, and their impacts on the usability. The objective of isolation requirement is to have fine-grained security measures in the network. Therefore, it is required to devise an appropriate combination of security devices for providing fine-grained security controls.

2.3.1 Isolation Patterns

Isolation patterns can be network level, host level, or application level. In this research, we consider the following network level isolation patterns:

- *Access denial*. This is naturally enforced by a firewall.
- *Trusted communication*, *i.e.*, authenticated and encrypted communication. IPSec devices are used to build trusted path (a.k.a. tunnel).
- *Payload inspection*. This is done by an intrusion detection system (IDS).

ConfigSynth allows network administrators to define isolation patterns considering different security devices (*primitive isolation*) and their combinations (*composite isolation*), along with their relative order based on the capabilities and functionalities of the devices. A set of primitive isolation patterns is shown in Table 2. Each pattern is represented using an ID, k. As shown in the table, $k = 1$ denotes '*access denial*' and $k = 2$ for '*trusted communication*', and so on. We formalize the isolation measures (*i.e.*, the *security configurations*) as a set of rules, where each isolation rule is defined as follows:

$$y^k_{i,j}(g), \text{where, } i,j \in \mathbb{H} \text{ and } g \in \mathbb{G}$$

The term $y^k_{i,j}(g)$ indicates that corresponding kth isolation pattern is required to be implemented between the host pair $\{i,j\}$ for service g. Notice that a host can represent a group of hosts if they have the same properties with respect to the operating system, running services, and users, and they reside in the same subnet.

TABLE 2
Network Level Isolation Patterns

Isolation (k)	Isolation Pattern	Decision	Score
1	Access Denial	$y^1_{i,j}(g)$	4
2	Trusted Communication	$y^2_{i,j}(g)$	2
3	Payload Inspection	$y^3_{i,j}(g)$	1
4	Traffic Forwarding through Proxy	$y^4_{i,j}(g)$	1
5	Traffic Forwarding through Proxy with Trusted Communication	$y^5_{i,j}(g)$	3

TABLE 3
Security Devices

Id (d)	Device Name	Primitive Isolation Pattern
1	Firewall	Access Denial
2	IPSec	Trusted Communication
3	IDS	Payload Inspection
4	Proxy	Traffic Forwarding through Proxy

An application of an isolation pattern requires the deployment of one or more security devices. Usually, an isolation pattern is related to a particular type of security device. This one-to-one matching is true for primitive isolation patterns. In case of a composite isolation pattern, it is required to deploy more than one security device. The following equation models the relationship between an isolation pattern and associated security device(s):

$$\forall_{i,j,g}, y^k_{i,j}(g) \Rightarrow x^d_{i,j}(g) \tag{1}$$

Equation (1) specifies that if the kth isolation is selected for the flow $g(i,j)$, the dth (type of) security device is required to be deployed between the host pair $\{i,j\}$ (*i.e.*, on the route of the flow). A particular value of d denotes a particular type of security device. For example, as shown in Table 3, $d = 1$ represents a firewall security device. If kth pattern is a composite isolation pattern, multiple security devices are required to implement the isolation pattern. Hence, in this case, multiple $x^d_{i,j}(g)$s are true. Usually, a security device deployment depends on the isolation pattern only, not on the flows (*i.e.*, i, j, or g). Equation (1) considers this. Table 3 shows a list of network security devices and the associated primitive isolation patterns.

2.3.2 Isolation Calculation

We define the isolation *score* (also named as *rank*) of the kth isolation pattern between a pair of hosts $\{i,j\}$ under the network service g by the parameter $L^k_{i,j}(g)$. The score of an isolation pattern denotes its isolation capability compared to others. The scores are computed based on the relative order of the isolation patterns according to their isolation capabilities. An administrator can provide the relative order explicitly or partial information about the order. A simple formal model is developed based on the given partial order between different isolation patterns. The model generates a complete relative order by assigning a value to each isolation pattern. The value assigned to a pattern denotes its (relative) *isolation score*. The highest value specifies the maximum isolation score. It is plausible to assume the same score (L^k) for a particular isolation pattern irrespective of hosts and services. Table 2 shows an example of relative isolation scores from the following partial information:

$$\forall_{k \neq 1}, L^k < L^1$$

$$(L^2 > L^3) \wedge (L^2 > L^4) \wedge (L^5 > L^2)$$

It is worth mentioning that this scoring of isolation patterns is relative, and security requirements based on this scoring system reflect the same relative meaning.

The decision variables $y^k_{i,j}(g)$, for all k, represent isolation patterns between a pair of hosts $\{i, j\}$ for the flow $g(i, j)$. These decision variables and associated isolation weights $L^k_{i,j}(g)$ are used to formally define the total isolation ($\bar{I}_{i,j}$) of j with respect to the incoming traffic from i. $\bar{I}_{i,j}$ is formalized as follows:

$$\bar{I}_{i,j} = \sum_g \sum_k y^k_{i,j}(g) \times L^k_{i,j}(g)$$

The equation indicates that the isolation between a pair of hosts $\{i, j\}$ is the sum of the isolation measures taken for different services between these hosts.

The isolation of a host depends not only on the hosts that can connect to it, it also depends on the hosts that it can connect to. For example, if a host can connect to the Internet, the host can download malicious content from the Internet and can get infected. However, the impact of such communication is less compared to the communication coming from the other direction. Since the outgoing traffic from j to i is the incoming traffic for i from j, the total isolation $I_{i,j}$ considering both the incoming and the outgoing traffic with respect to j for the pair of hosts $\{i, j\}$ is defined as follows:

$$I_{i,j} = \alpha \bar{I}_{i,j} + (1 - \alpha)\bar{I}_{j,i} \tag{2}$$

Here, α ($0 \leq \alpha \leq 1$) is the weight for the isolation due to the incoming traffic, while $1 - \alpha$ is the weight for the isolation due to the outgoing traffic. The total isolation score of a host j is defined in (3).

$$I_j = \sum_{i \neq j} I_{i,j} \tag{3}$$

Equation (4) represents the overall isolation in the network (*i.e.*, the *network isolation*) considering all of the hosts.

$$I = \sum_i I_i \tag{4}$$

2.4 Modeling of Usability

Business constraints like usability play a significant role in synthesizing usable security configurations in a network. For example, a higher network isolation can provide strong defense, but the network usability might reduce to a level which is unacceptable to the organization. In this subsection, we discuss the formalization of the usability.

2.4.1 Connectivity Requirements

Every organization usually has a number of service flows, which are essential for its successful operation. Each of these *connectivity requirements* represents a flow that must be able to communicate. Connectivity requirements are formalized as a set of rules, where each connectivity rule defines the mapping from a flow (*i.e.*, a tuple of source, destination, and service) to a decision variable c that represents whether the flow is required to be allowed. Each rule is denoted as $c_{i,j}(g)$, where $i, j \in \mathbb{H}$ and $g \in \mathbb{G}$. Here, $c_{i,j}(g)$ is a binary variable. When it is true, it represents that the service flow g must be allowed from i to j. If it is false, then nothing has been specified for this flow, *i.e.*, the flow can either be allowed or denied. CR represents the conjunction of all connectivity requirements.

$$CR \Rightarrow \bigwedge_{i,j,g} c_{i,j}(g) \tag{5}$$

2.4.2 Usability Calculation

The usability of the network depends on the ranks of the service flows between the hosts in the network. The rank of a service flow denotes the demand of the flow. Each service flow $g(i, j)$ is associated with a rank, $a_{i,j}(g)$. These ranks are expected to be given in the form of a relative order by the administrator based on the organizational demand. Partial information can be given, from which a complete relative order can be derived, as it has been shown in the case of the isolation patterns. If no specification is given about the demand of a flow, it receives the default (*i.e.*, the minimum) rank. The usability of a service g running on a host j is formalized by aggregating the usability scores of all the flows corresponding to this service:

$$S_j(g) = \sum_i \sum_k y^k_{i,j}(g) \times b^k_{i,j}(g) \times a_{i,j}(g)$$

The application of an isolation pattern to a flow can affect the usability of the flow. The parameter $b^k_{i,j}(g)$ represents the usability of the flow $g(i, j)$ when the k^{th} isolation pattern is implemented. We assume that the usability depends on the isolation pattern, not on the host-pair (*i.e.*, $b^k_{i,j}(g) = b^k(g)$). The value of $b^k(g)$ can be determined based on the prior knowledge of network security by considering the time or effort required to get a service access under an isolation measure. The valuation of the parameter $b^k(g)$, in the simplest form, can be as follows: the 'access denial' isolation pattern reduces the usability to zero, *i.e.*, $\forall_g, b^1(g) = 0$; while other isolation patterns maintain the same usability, *i.e.*, $\forall_{g,k \neq 1}, b^k(g) = 1$. The usability S_j represents the accumulated usability considering all of the services running on a host j, as follows:

$$S_j = \sum_g S_j(g)$$

Equation 6 adds up the usability scores corresponding to all the hosts, which is ultimately the aggregation of the usability scores for all the service flows in the network This aggregated value represents the overall usability of the network (*i.e.*, the *network usability*).

$$U = \sum_j S_j \tag{6}$$

2.5 Modeling of Deployment Cost

The deployment of a security device incurs costs and an organization often has a budget limit for implementing security measures. The deployment cost is the sum of the prices of the security devices that are required to be deployed in different segments of the network in order to implement necessary isolation patterns between different host-pairs. The number of security devices depends not only on the isolation measures but also on how they are distributed in the topology. The cost cannot be calculated from the isolation measures alone. This is because of the fact that there are usually similar types of isolation between multiple host-pairs, and these host-pairs can share one or more links for communication. In this case, placing a single security device at one of the shared links may ensure the desired isolation. Moreover, if there are more than one routing path between a host-pair, we have to secure all of the alternative paths. Therefore, modeling correct and optimal placements of the security devices is very challenging, considering the network topology, the isolation patterns, and the budget.

2.5.1 Flow Routes

ConfigSynth requires the flow routes between the hosts for the purpose of determining the placements of the security devices satisfying the isolation measures. A *flow route*, $F^z_{i,j}$ is defined as a set of links $\{l_{i,j,z,1}, l_{i,j,z,2}, ...\} \subseteq \mathbb{L}$, that form a path from a source i to a destination j. As multiple routes are possible between a pair of hosts, z indicates the index of a flow route (*i.e.*, the z^{th} route), between the host-pair $\{i, j\}$. The term $|F^z_{i,j}|$ denotes the path length, *i.e.*, the number of hops or links in the path. $F_{i,j}$ denotes all of the flow routes from i to j:

$$F_{i,j} \Rightarrow \bigwedge_z F^z_{i,j}$$

ConfigSynth finds the flow routes for a host pair by applying a path searching algorithm on the network topology.

2.5.2 Device Placements and Cost Calculation

Equation (1) specifies the security devices which are required to employ an isolation pattern. The placements of the security devices on the flow routes are modeled from these specifications. If an isolation pattern, *e.g.*, 'access denial', is selected for the traffic from a host i to a host j, then it is required to block the traffic through all possible flow routes between $\{i, j\}$. Equation (1) specifies a firewall to be deployed for implementing an 'access denial' isolation pattern. Hence, there should be a firewall deployed at least on a link of each flow route. We formalize the placement of a security device d for a particular pair of hosts as follows:

$$\forall_g \; x^d_{i,j}(g) \Rightarrow \forall_z \exists_t l^d_{i,j,z,t} \tag{7}$$

In the equation, $l^d_{i,j,z,t}$ represents that a security device of type d is deployed on the link $l_{i,j,z,t}$. It can be noted that if there is a security device, *e.g.*, firewall, on the flow route for a host pair, this does not imply that the flow access between the pair is denied. It is denied only if the 'access denial' isolation pattern is specified for the host pair.

The placement of an IPSec device requires special modeling which is different from that of the security devices like firewall and IDS. The 'trusted communication' isolation pattern usually requires encrypted communication (*i.e.*, tunnel) to take place throughout the unsecured or untrusted part of the routing path. Moreover, to ensure an encrypted tunnel between a host pair, it is required to deploy two IPSec devices, one at the source side (start of the tunnel) and another at the destination side (end of the tunnel). A network administrator needs to specify the guidelines for placing the IPSec gateways. The administrator can specify the maximum number of hops (*i.e.*, the number of links) from the end-hosts that can be outside of the tunnel. For example, it can be specified that the source-gateway and the destination-gateway should be deployed within two hops from the source and the destination, respectively. We model this as a conjunction of three clauses:

$$\begin{aligned}\forall_g \; x^2_{i,j}(g) \Rightarrow & \forall_z (|F^z_{i,j}| > (2 \times T)) \wedge \\ & (\exists_t (l^2_{i,j,z,t} \wedge (t \leq T)) \wedge \\ & \exists_{t'} (l^2_{i,j,z,t'} \wedge ((|F^z_{i,j}| - t') \leq T)))\end{aligned}$$

Here, T denotes the maximum number of hops that can be outside of the tunnel. Hence, the second clause of the equation limits the number of hops between the source and the source-gateway within T. The third clause ensures the same between the destination and the destination-gateway. Now, if the flow route between the source and the destination does not have a hop length larger than $2T$, it is not possible to deploy 'trusted communication' between this pair of hosts. The first clause establishes this constraint.

For the deployment of the security devices, the deployment cost is computed as the summation of the costs of all of the devices deployed in different links. We define C_d as the average deployment cost of the security device d. Now, if l^d denotes whether a security device d is deployed on the link $l \in \mathbb{L}$, the total deployment cost C is computed as follows:

$$C = \sum_{l \in \mathbb{L}} \sum_d l^d \times C_d, \text{where } l^d \Rightarrow \exists_{i,j,z,t}, l^d_{i,j,z,t} \tag{8}$$

2.6 Modeling Constraints

ConfigSynth synthesizes security configurations by solving a number of constraints. In the following, we discuss these constraints in different categories.

2.6.1 Threshold Constraints

In ConfigSynth, we have three generic threshold-based constraints in selecting the security measures (*i.e.*, isolation patterns) on the network flows.

$$TC : (I \geq Th_I) \wedge (U \geq Th_U) \wedge (C \leq Th_C) \tag{9}$$

In the equation, Th_I, Th_U and Th_C represent the slider values, *i.e.*, the constraints on the network isolation, usability, and deployment cost, respectively. The network isolation and the network usability must be greater than or equal to their respective threshold values, Th_I and Th_U. The deployment cost must also be within the budget, Th_C. Since each slider value is selected from a scale of 0 to some upper limit (*e.g.*, 10), the slide values are adjusted with respect to the maximum possible values of corresponding metrics. For example, if I^{max} is the maximum possible network isolation, V is the corresponding slider value, and V^{max} is the slider's upper limit, then Th_I is $I^{max}V/V^{max}$.

2.6.2 Invariant Constraints on Isolation Selections

There are different invariant constraints in ConfigSynth which ensure the consistency between functional behaviors of the isolation patterns and the business requirements.

$$IIC_1 : y^k_{i,j}(g) \Rightarrow \forall_{\bar{k} \neq k} \neg y^{\bar{k}}_{i,j}(g)$$

$$IIC_2 : c_{i,j}(g) \Rightarrow \neg y^1_{i,j}(g)$$

The constraint IIC_1 states that only one isolation pattern can be selected for a flow. The constraint IIC_2 ensures that if 'access denial' is chosen as the isolation pattern for a flow from i to j, there cannot be any connectivity requirement for that flow. IIC is the conjunction of all these constraints.

TABLE 4
Assistance on Choosing Sliders' Values by ConfigSynth

Isolation score = 10 : Usability score = 0 No flow is allowed to communicate. Each host is isolated from other hosts.
Isolation score = 0 : Usability score = 10 No isolation measure is taken on any flow.
Isolation score = 8.2 : Usability score = 1.8 Each flow is protected by 'access denial' except connectivity requirements.
. . . .
Isolation score = 5 : Usability score $\leq$ 5 1/2 of the flows (50%) are protected by 'access denial'.
Isolation score 5 : Usability score $\leq$ 7.5 1/4 of the flows (25%) are protected by 'access denial', 1/4 of the flows (25%) are protected by 'trusted communication'.
. . . .

Algorithm 1 Systematic Analysis of UNSAT Result

```
if Solver returns UNSAT then
    Get the UNSAT-CORE $\mathcal{U}$.
    $\mathbb{A}$ is the set of all combinations of assumptions in $\mathcal{U}$.
    for Each combination of assumptions, $\mathcal{A} \in \mathbb{A}$ do
        Remove the assumptions in $A$ from the query $Constr$.
        if Solver returns SAT then
            Get the Model, $\mathcal{M}$.
            Print each $Th_{A_i}$ associated to the value of $A_i \in \mathcal{A}$, as found in $\mathcal{M}$.
        end if
    end for
end if
```

2.6.3 User-defined Isolation Policy Constraints

User-defined constraints represent organizational requirements. Some examples are as follows:

$$UIC_1 : g(i,j) \wedge (g = \text{SSH}) \Rightarrow \neg y^2_{i,j}(g)$$

$$UIC_2 : \neg y^1_{i,\hat{j}}(g) \Rightarrow (\bar{i} = \text{Internet}) \wedge y^1_{i,i}(g)$$

$$UIC_3 : c_{i,j}(g) \wedge (g = \text{WEB}) \Rightarrow \neg y^2_{i,j}(g)$$

An organizational policy (UIC_1) may state that IPSec should not be deployed for a pair of hosts in the case of SSH (Secure Shell)-based communication. The isolation requirement for a particular type of flow can be defined by stating that access will be allowed from a source i to a specific destination $\hat{j}$ under the service g, if the Internet is not allowed to connect to i. This is modeled in UIC_2. The organizational policy may require that no web service should be protected by the 'trusted communication' isolation pattern (UIC_3), while the flow is already specified to be allowed as a connectivity requirement. We use UIC to denote all such user-defined constraints.

2.7 Implementation of ConfigSynth

The main objective of our configuration synthesis problem is to maximize the security in the network by satisfying various security requirements as well as the organization's business constraints. Thus, the synthesis problem is formalized as the satisfaction of the constraint, ($Constr$), which is the conjunction of all of the constraints as follows:

$$Constr \Rightarrow CR \wedge TC \wedge IIC \wedge UIC \qquad (10)$$

The satisfaction to this constraint provides necessary security configurations, *i.e.*, isolation patterns between different host pairs ($y^k_{i,j}(g)$s), along with the placements of necessary security devices (l^ds).

2.7.1 SMT Encoding and Query Formulation

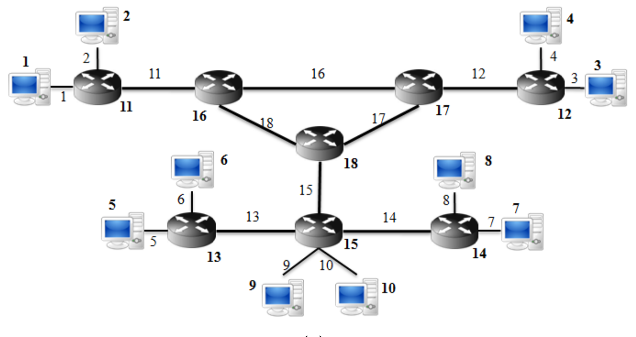
We implement our model by encoding the system configuration and the constraints into SMT logics [1]. For encoding the formalizations of the network topology, device configurations, traffic modeling, and the security and business properties, we use mainly two types of terms: *boolean* and *integer*. Boolean terms are used for encoding the boolean configuration parameters and decision variables, such as isolation patterns and device placements. The remaining parameters are modeled as integer terms. In our modeling, we represent a host using an integer ID, which is not necessarily in IP address format, since no IP address-based computation is required in this model. Each service is also encoded as an integer value (as an ID specifying a protocol-port pair). ConfigSynth takes the system configurations, requirements and constraints from a text file (*input file*).

Choices for Sliders' Values: An administrator applies constraints on the network isolation, usability, and deployment cost by selecting the associated sliders. Each slider has a scale, *e.g.*, from 0 to 10. A particular choice for a slider, especially in the cases of isolation and usability, may not give an exact understanding of the expected behavior. For example, in the case of the isolation slider, the maximum slider value(*i.e.*, 10) represents that each host is isolated from the rest (*i.e.*, the 'access denial' isolation pattern is applied on each flow). On the other hand, the slider value of zero represents that no isolation measure is taken in the network. However, the level of security meant by a particular selection of the slider other than the maximum and minimum is hard to envisage. Since a particular network isolation can be achieved using different security configurations, depending on the selection of the usability requirement, ConfigSynth assists its users by providing some simple but useful example scenarios, so that they can have an understanding of the slider values with respect to the security configurations.

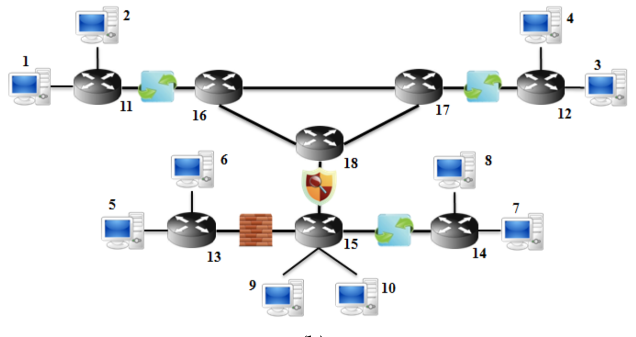
According to the given network topology, security requirements, and business constraints, ConfigSynth presents a number of security plans and the corresponding isolation and usability scores. Table 4 shows an example of such an assistance, considering the example referred to Section 2.8. An isolation score of 8.2 and a usability score of 1.8 can be achieved when each flow is protected by 'access denial', except connectivity requirements. An isolation score of 5 and a usability score no more than 7.5 can be achieved when a quarter of the flows are protected using 'access denial' and another quarter of flows are protected using 'trusted communication'. Therefore, when the isolation and usability requirements are 7.5 and 2.5, respectively, it is evident that the isolation measures cannot be all access denial, while a larger number a flows than a half need to be protected by 'access denial and 'trusted communication'.

2.7.2 Synthesis Result Analysis

ConfigSynth uses Z3 SMT solver to check the verification constraint (*Constr*), which provides a satisfiable (SAT) result if all constraints are satisfied. The SAT result provides a SAT instance, which represents the value assignments to the parameters of the model. According to our objective, we require the assignments of the following variables: (i) the



(a)



(b)

Fig. 2. (a) An example network for synthesizing security configurations and corresponding security device placements. An ID is assigned to each of the hosts, routers, and links. (b) The solution to the example problem, *i.e.*, the placements of necessary security devices.

security configurations, *i.e.*, the isolation variable ($y^k_{i,j}(g)$), between each pair of hosts in the network, and (ii) the security device placements in the topology, *i.e.*, the device placement variable (l^d) for each link. The security configurations and the device placements are printed in a text file (*output file*) for the user. ConfigSynth also provides a graphical representation of the output (*i.e.*, the network with the placements of the security devices) to its user. Fig. 2 shows snapshots of such graphical representations (the network before and after the synthesis).

UNSAT result and its analysis: If there is any disagreement or inconsistency between the constraints, the SMT solver gives an unsatisfiable (UNSAT) result. In order to find the constraints that lead to the unsatisfied result, it is required to get the UNSAT-CORE. The UNSAT-CORE can be achieved through the use of *hard* and soft clauses or constraints [3], [4]. In an SMT formula, a clause is hard when it is must be satisfied, while a clause is soft when it may be satisfied. Soft clauses are often specified as *assumptions* in Z3. The SMT formula can be verified for all or some assumptions. If the result is UNSAT, then the list of assumptions that are unsatisfied are returned in the UNSAT-CORE. The constraints that we take as hard clauses are connectivity requirements (CR), invariant constraints (IIC), and user-defined constraints (UIC). In contrast, we consider threshold constraints (Equation 9) as assumptions.

By performing a systematic analysis of the UNSAT-CORE, ConfigSynth shows the constraints that are required to be tuned or modified in order to satisfy the model. We follow Algorithm 1 to show all possible closed solutions. In this way, ConfigSynth helps in identifying inconsistencies in the constraints and provides satisfiable choices for the constraints, which is an added support to network administrators for synthesizing the best security specifications.

TABLE 5
Input (Partial) to the Example

```
# Number of Security Devices
3 # 1 for Firewall, 2 for IPSec, and 3 for IDS, while 0 for None
# Isolation Specifications (partial orders)
2 # Device, Device, Comparison (1 for =, 2 for >, and 3 for >=)
1 2 2
2 3 2
# Usability if an isolation pattern is applied
0 2 3
# Cost of each isolation device (in thousand dollars)
20 18 15
# Number of Hosts and Routers
10 8
# Links
18
1 11
2 11
. . . . . . . . .
# Connectivity Requirements (each row for a host, which ends with 0)
3 0 # The flow from Host 1 to Host 3 must be allowed
4 0
1 2 0
2 0
3 4 0
3 4 0
1 2 0
1 0
0
1 0
# Sliders' Values (Isolation 0-10, Usability 0-10, Cost in thousand dollars)
6 5 90
```

TABLE 6
Selected Isolation Patterns for the flows in the Example

Destination	Sources Classified according to Selected Isolation Patterns			
Host	Access Denial	Trusted Communication	Payload Inspection	No Isolation
1	5, 6	3, 4, 7, 8	9, 10	2
2	5, 6	3, 4, 7, 8	9, 10	1
3	—	1, 2, 7, 8	6, 9, 10	4, 5
4	—	1, 2, 7, 8	9, 10	3, 5, 6
5	1, 2, 3, 4, 7, 8, 9, 10	—	—	6
6	1, 2, 3, 4, 7, 8, 9, 10	—	—	5
7	5, 6	1, 2, 3, 4, 9, 10	—	8
8	5, 6	1, 2, 4, 9, 10	3	7
9	5, 6	7, 8	1, 2, 3, 4	10
10	5, 6	7, 8	1, 2, 3, 4	9

2.8 A Case Study

Fig. 2(a) shows a small network for which an optimal security design will be synthesized based on the given input file as shown in Table 5. In this example, the connectivity requirements are considered as a list of allowed services between different hosts. In order to keep the example simple, we consider only three primitive isolation patterns (*i.e.*, 'access denial', 'trusted communication', and 'payload inspection'). We also assume a single flow type (*i.e.*, a single service) between each pair of hosts. ConfigSynth gives a SAT result for this example. From the resultant SAT instance, we find the necessary isolation patterns along with the necessary device placements. Fig. 2(b) shows the placements of the security devices. Table 6 shows the isolation patterns. For an instance, the first row of the table specifies the isolation patterns that are selected on the incoming traffics toward the host 1.

3 Security Architecture Refinement

ConfigSynth provides a security architecture satisfying the security requirements and business constraints, as we have seen in the last section. We can compute the optimal solution by running the ConfigSynth framework many times

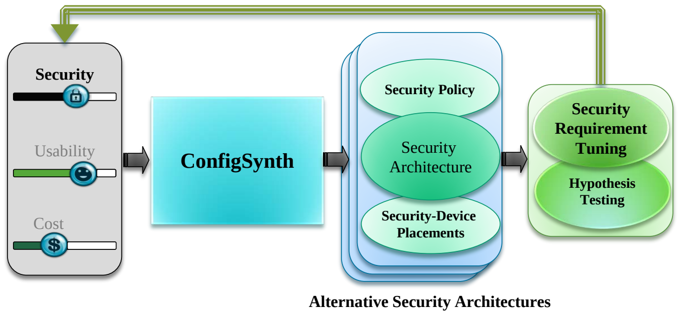

Security
Usability
Cost
ConfigSynth
Security Policy
Security Architecture
Security-Device Placements
Alternative Security Architectures
Security Requirement Tuning
Hypothesis Testing

Fig. 3. The security architecture refinement framework.

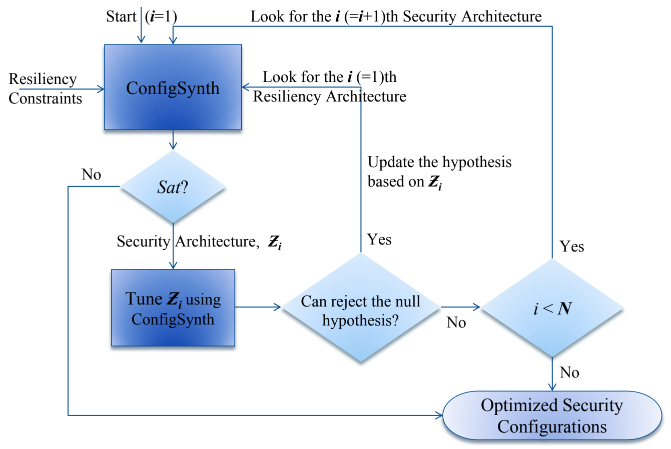

Start (i=1)
Look for the i (=i+1)th Security Architecture
Resiliency Constraints
ConfigSynth
Look for the i (=1)th Resiliency Architecture
Update the hypothesis based on Z_i
No
Sat?
Security Architecture, Z_i
Yes
Yes
Tune Z_i using ConfigSynth
Can reject the null hypothesis?
No
i < N
No
Optimized Security Configurations

Fig. 4. The process flow diagram for the security architecture refinement.

by changing the isolation requirement, *e.g.*, by running a binary search algorithm within a range of maximum and minimum isolation requirements. Unfortunately, from evaluation results (refer to Section 4), we have found that if the isolation requirement is very close to the optimal value that can be achieved for the given business constraints, the time for executing the ConfigSynth framework turns out to be significantly high, compared to the rest of the cases. In such a case, the time to conclude whether there is a solution or not is found to be notably high even when the number of hosts is as small as 30 hosts. Moreover, if the network is in a clean state, *i.e.*, there is no given isolation measure as well as security device placement, synthesizing a satisfiable security architecture needs a long time as the search space is large. On the other side, if there is already a partial security architecture, the time to improve the architecture by adding further security measures is significantly short. Therefore, we adopt the concept of statistical hypothesis testing for tuning or improving the security architecture toward finding a security architecture closer to the optimal one. Fig. 3 shows the corresponding extended architecture of ConfigSynth.

In this section, we develop a mechanism for the security architecture refinement based on the hypothesis testing-based analysis. We also present a case study demonstrating the security architecture refinement.

3.1 Hypothesis Testing-Based Refinement

In this refinement process, our objective is to disprove the null hypothesis, which we take as "there is no better security architecture within the given business constraints other than the known best security architecture." Thus, in order to reject this null hypothesis, we need to prove the alternative hypothesis, *i.e.*, there are security architectures significantly better than the known best security architecture, which satisfy the given business constraints as well. In this hypothesis testing, the standard error is considered as 5% of the isolation provided by the known best security architecture. Let us define N as the number of alternative security architectures that will be used to verify our null hypothesis. We would like to reject the null hypothesis if we find just one better security architecture from the N number of architectures. The reason of this idea is to increase the refinement efficiency with respect to time, because each search/synthesis for a security architecture consumes a substantial time.

We find that if we consider the significant level as 5% and the standard deviation as 5% of the maximum isolation, then if N is taken as 25, the null hypothesis can be rejected if and only if one security architecture is found that offers an isolation significantly better than the known best isolation. If the known best isolation is 5, then the increase in isolation needs to be around 10% of the known best isolation. It is worth mentioning that, for a larger known best isolation value, the increase requirement can be smaller in order to reject the null hypothesis if there is a single security architecture that provides the increased isolation irrespective of the others in the sample set. In the case of a smaller known best isolation, a larger increase requirement is needed to reject the null hypothesis. In Section 3.1.1, we present the calculation process with regards to the hypothesis testing considering the above mentioned parameters, showing the basis of this refinement mechanism. In this mechanism, the potential of Type II error is unknown when we have N number of alternative security architectures, as we do not know the maximum number of alternative security architectures. However, if the number of architectures is less than N, there is no Type II error.

3.1.1 Selection of Parameter Values in Algorithm 2

We adopt the one-sided test in our hypothesis testing mechanism. As we look for better security architecture in the refinement process, we consider right-sided test. That is, alternative hypothesis specifies the set of values strictly greater than the critical value. Here, we briefly present the process of hypothesis testing for a particular scenario with regards to the chosen parameters above. This illustration will prove the correctness of this particular selection of values for those parameters.

The Hypothesis Testing Process:

- According to the isolation scale of 0 to 10, known best isolation, $I = 5$
- Sample size, *i.e.*, the number of alternative security architectures, $N = 25$
- Standard deviation, $D = 0.05$
- Standard Error, $E = D/\sqrt(N) = 0.05 \ / \ 5 = 0.01$
- Assume, each of $N-1$ security architectures provides isolation I in average, while the rest one pro-

Algorithm 2 Systematic Refinement of Security Architecture

```
Require: $Synth$ is the synthesis model (i.e., the ConfigSynth framework) as of Equation 10.
Require: $I^{mean}$ is the mean isolation.
Require: Null Hypothesis ($H_0$): There is no security architecture possible providing higher isolation score than $I^{mean}$.
Require: $N$ is the number of alternative security architectures, i.e., the sample size.
Require: $I^{min}$ is the minimum isolation requirement for a security architecture to be a member of the sample set. $I^{min} \leq I^{mean}$.
Require: $Th_I$ is initialized with $I^{min}$.
Require: $X$ (e.g., 10% of $I^{mean}$) is the minimum increase in the isolation score provided by a security architecture in order to reject the null hypothesis.
Require: $I^{max}$ is the maximum isolation provided by a security architecture after improvement.
Require: $\mathcal{Z}^{Best}$ is the known best security architecture.
 1: for $i = 1$ to $N$ do
 2:   if Solver returns SAT then
 3:     Get the model, $\mathcal{M}$.
 4:     From $\mathcal{M}$ fetch the complete security architecture, $\mathcal{Z}$ (it includes the isolation measures for all the flows and security device placements for all the links).
 5:     From $\mathcal{M}$ fetch the applied security, $\bar{\mathcal{Z}}$ (it specifies the flows with positive isolation measures and the links with one or more device placements).
 6:     Update $Synth$ by adding $\neg\mathcal{Z}$.
 7:     Push or save $Synth$ in the memory.
 8:     Set $I^{max}$ to $I$ as it is obtained from $\mathcal{M}$.
 9:     Update $Synth$ by adding $\bar{\mathcal{Z}}$.
10:     Increase $Th_I$ with a small value $Y$, such that $Y \leq X$,
11:     while Solver returns SAT do
12:       Get the model, $\mathcal{M}'$.
13:       From $\mathcal{M}'$ fetch the complete security architecture, $\mathcal{Z}'$.
14:       From $\mathcal{M}'$ fetch the security security, $\bar{\mathcal{Z}}'$.
15:       Pop or retrieve $Synth$ from the memory, that we saved in Steps 7 or 17.
16:       Update $Synth$ by adding $\neg\mathcal{Z}$.
17:       Push $Synth$ in the memory.
18:       Update $Synth$ by adding $\bar{\mathcal{Z}}$.
19:       Set $I^{max}$ to $I$ as it is obtained from $\mathcal{M}$.
20:       Update $Synth$ by adding $\mathcal{Z}$.
21:       $Th_I = Th_I + Y$.
22:     end while
23:     if $I^{max} \geq I^{mean} + X$ then
24:       Reject $H_0$.
25:       Update $I^{mean}$ with $I^{max}$.
26:       Update $\mathcal{Z}^{Best}$ with $\mathcal{Z}'$.
27:       Reinitialize $i$ with 1.
28:     end if
29:     Pop $Synth$ from the memory, that we saved in Steps 7 or 17.
30:   end if
31:   Return $\mathcal{Z}^{Best}$.
32: end for
```

vides I' isolation significantly greater than I. Let I' = I + 10% of I. Then, I' = (5 + 0.5) = 5.5

- Thus, the average isolation provided by the security architectures, $\bar{I}$ = (5 × 24 + 5.5) / 25 = 5.02
- Test Statistic, z = (1.004I - I) / E = 0.4I
- Significance Level, α = 5%
- At this significance level, from the table of z-scores, Critical Value (CV) = 1.645
- Since z > CV, the null hypothesis is rejected.

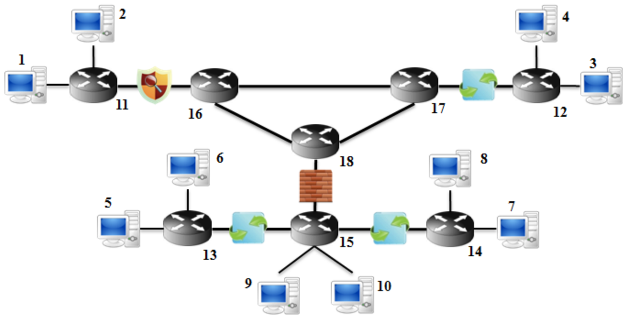


Fig. 5. The placements of security devices after the refinement.

TABLE 7
Selected Isolation Patterns for the flows in the Example

Destination	Sources Classified according to Selected Isolation Patterns			
Host	Access Denial	Trusted Communication	Payload Inspection	No Isolation
1	5, 6, 9	–	3, 4, 7, 8, 10	2
2	5, 6, 8, 9	–	3, 4, 7, 9, 10	1
3	7, 8, 9, 10	5, 6	1, 2	4
4	7, 8, 9, 10	5, 6	1, 2	3
5	1, 2, 3, 4	7, 8, 9, 10	–	6
6	1, 2, 4	3, 7, 8, 9, 10	–	5
7	1, 3, 4	5, 6, 9, 10	2	8
8	1, 2, 3, 4	5, 6, 9, 10	–	7
9	1, 2, 3, 4	5, 6, 7, 8	–	10
10	1, 3, 4	5, 6, 7, 8	2	9

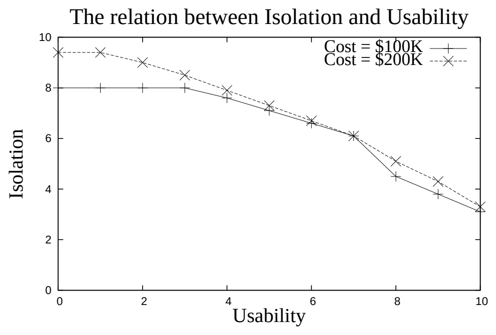
As we stated before, for a larger I, I' can be smaller, while for a smaller I, a larger I' is required, to find a security architecture providing an increased isolation greater or equal to I' and so to reject the null hypothesis.

3.2 Refinement Mechanism

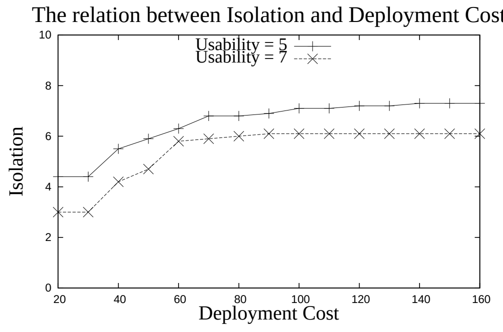
According to the hypothesis testing-based refinement idea discussed in the previous subsection, we devise a mechanism as presented in Algorithm 2. The corresponding process flow diagram is shown in Fig. 4. The mechanism starts with a null hypothesis specifying an isolation value (I^{mean}) which is the isolation provided by the known best security architecture. If the null hypothesis is rejected, then we update the null hypothesis with the isolation of the security architecture that beats the null hypothesis and start the hypothesis testing from the beginning according to the updated hypothesis. The refinement process continues until we fail to reject the null hypothesis.

3.3 A Case Study

In this case study, we use the same network that we have used in Section 2.8. The corresponding inputs are similar to Table 5. Therefore, the objective of the refinement process is to find the optimal security architecture that provides the best isolation considering isolation 6 as the minimum security requirement and satisfying the business constraints. After the execution of Algorithm 2, we receive the best security architecture, which provides isolation 6.63. The placement of security devices is shown in Fig. 5 and the security policy (*i.e.*, the isolation measures) is presented in Table 7. If we compare Table 7 with Table 6, we see that a greater number of traffics are denied access, while trusted communication is selected for most of the allowed traffic flows. This selection of isolation patterns requires better placements of security devices, which is reflected in Fig. 5.

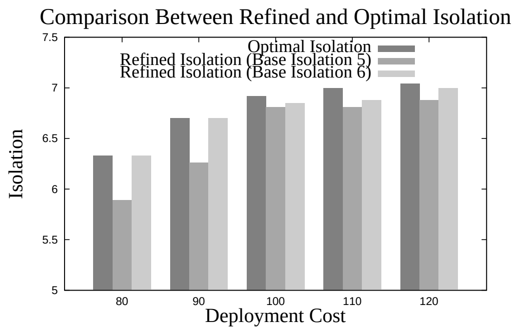


(a)

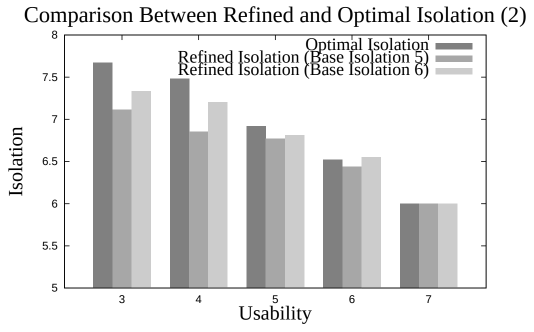


(b)

Fig. 6. (a) The maximum possible isolation with respect to the usability constraint considering a fixed cost constraint (*i.e.*, $200K) and (b) the maximum possible isolation with respect to the deployment cost constraint considering a fixed usability constraint (*i.e.*, 5).



(a)



(b)

Fig. 7. A comparison between the optimized isolation provided by the refinement mechanism and the optimum isolation with respect to (a) various deployment cost and (b) various usability constraints.

As the algorithm executes, we observe 4 updates of the null hypothesis. The null hypothesis starts with the known best isolation 6. That is, the hypothesis specifies that there is no security architecture that provides isolation greater than 6 as well as satisfies the business constraints. This null hypothesis is rejected by the first alternative security architecture as we find a security architecture with isolation 6.07. Thus, the null hypothesis is updated with this isolation 6.07, *i.e.*, there is no security architecture which provides isolation greater than 6.07 as well as satisfies the business constraints. Based on this updated null hypothesis, the algorithm starts from the beginning with the empty set of security alternatives. This null hypothesis is rejected by the 2nd alternative security architecture. We receive isolation around 6.40 according to this architecture. Therefore, the current null hypothesis is rejected, while the null hypothesis is renewed with isolation 6.40. This null hypothesis is rejected, as we find that the 16th security architecture provides isolation 6.44. The null hypothesis is updated according to this new isolation. This null hypothesis is rejected again. It is rejected by the 12th alternative security architecture as it provides isolation 6.63. Thus, the null hypothesis is updated again with isolation 6.63. This time, we cannot reject this updated null hypothesis, as none of the 25 alternative security architectures can provide isolation greater than 6.63.

4 Evaluation

Here, we first present the analysis on the relationships between the isolation, usability, and deployment cost. Then, we present the performance (*i.e.*, *scalability*) analysis of the tool. We ran our experiments on different test networks.

4.1 Analysis of the Relationships between Isolation, Usability, and Deployment Cost Constraints

In this analysis, we ran a number of experiments considering the same network topology as shown in Fig. 2(a). The impact of the network usability constraint on the network isolation is shown in Fig. 6(a) under two deployment cost constraints. We found that with the increase of the usability constraint, the maximum possible isolation reduces. However, due to the connectivity requirements, the isolation cannot be more than a particular point, although the usability constraint is very low. At the lower values of the usability constraint, the rate of the isolation decrease is less in comparison to that at the higher values of the constraint.

The deployment cost constraint has an impact on the isolation. Fig. 6(a) shows that in the case of the higher cost constraint (*i.e.*, $200K), a higher isolation is achieved compared to the case of the lower cost constraint (*i.e.*, $100K). A higher cost allows ConfigSynth to deploy more security devices, particularly IPSec devices in these experiments, which helps in increasing the isolation. We also found that with the increase of the usability constraint (*i.e.*, from 0 to 7), the difference between the maximum possible isolation values in both of the cases reduces. At the usability value 7, we found that the isolation difference sharply increases, then slowly reduces. The reasons behind this behavior are that different security devices have different prices and different impacts on the usability (refer to Section 2). Even different deployment aspects influence the deployment cost. For example, IPSec-based security usually requires deployment of two IPSec gateways close to the end hosts, which are at the boundary of the core network. This does not allow many hosts to share these gateways for implementing the 'trusted communication' isolation pattern. As a result, IPSec-based security incurs a higher deployment cost compared to that of the firewall- or IDS-based security.

Fig. 6(b) shows the relationship between the isolation and the deployment cost more adequately, considering two different usability constraints. We changed the cost constraint and observed the maximum possible isolation. Obviously, in the case of the lower usability constraint (*i.e.*, 5), the isolation is higher compared to the case of the higher usability constraint (*i.e.*, 7). We also found that after a certain level, it is not possible to increase the isolation, despite increasing the deployment budget. This is due to the usability constraint. To increase the isolation after a certain point, it is required to use the highly scored isolation patterns (*e.g.*, 'access denial'), which at the same time reduce the usability.

4.2 Analysis of the Isolation Optimization Capability of Refinement Mechanism

We compared the optimized isolation provided by the refinement mechanism with the optimum isolation found by applying the brute-force method. We considered a random network of 50 hosts and 10 routers, to execute the refinement process (Algorithm 2) for receiving the optimized isolation and compare this value with the optimum one. The results are shown in Fig. 7(a) and Fig. 7(b), where we varied the deployment cost and usability, respectively for two different values of the base isolation requirement, 5 and 6. In the first case, the usability constraint was kept at 5, while in the second case the cost constraint was fixed at \$100. As shown in Fig. 7(a), we observed that the improved isolation received after the refinement process is very close ($<$ 10%) to the optimum value, and when the base isolation requirement is high, the refined isolation becomes closer. We observed similar characteristics in Fig. 7(b). However, with the increase in the base isolation requirement, the execution time of ConfigSynth also increases rapidly, which ultimately increases the running time of the refinement process.

4.3 Scalability Analysis

Here we present the evaluation results demonstrating the scalability of ConfigSynth and the refinement mechanism.

4.3.1 Methodology

We evaluated the scalability of ConfigSynth by analyzing the *time* and *memory* required for synthesizing the configurations by varying the problem size and the constraints. The problem size depends mainly on the number of flows, since the synthesis problem considers the isolation pattern for each flow. The number of flows mostly depends on the number of hosts. We ran ConfigSynth in a machine running Windows 7 OS. The machine was equipped with an Intel Core i3 Processor and a 4 GB memory. The test networks were randomly generated where the number of hosts ranged from 50 to 1000 and that of routers from 8 to 20. The number of services between a pair of hosts ranged from 1 to 3 (*i.e.*, maximally 3 flows). Each of the isolation and usability constraints was a value between 0 and 10, representing a normalized score as 0 for no isolation/usability while 10 for complete isolation/usability.

4.3.2 Performance of ConfigSynth

Impact of the Problem Size: Fig. 8(a) and Fig. 8(b) show the model synthesis time with respect to the problem size. In the first analysis, we considered two different scenarios. In one scenario, the volume of the connectivity requirements is 10% of all the flows possible between the hosts. In the other scenario, the percentage is 20%. In this analysis, we varied the problem size with respect to the number of hosts and the corresponding results are shown in Fig. 8(a). We observed a quadratic increase in the analysis time with respect to the number of hosts. This is due to the fact that the problem size depends on the number of possible flows in the network. The number of flows is $O(N^2)$, where N is the number of hosts and the number of services is constant. The volume of the connectivity requirements also increases with the increase in the number of flows. This increase in the problem size requires a verification of more constraints. As a result, the running time is increased to over $O(N^2)$. In the second analysis, we varied the core network by changing the number of routers in two different connectivity requirements. The results are presented in Fig. 8(b). In this case, since the number of hosts in the network remains the same, there is no increase in the number of flows. However, due to the increase in the number of routers, the core network becomes larger, where the hosts are more distributed and more links turn out as the candidates for security device placements. As a result, an increased search is required to find a satisfiable model, which increases the synthesis time.

We analyzed the impact of the volume of the connectivity requirements (*i.e.*, the percentage of all flows that are in the connectivity requirements) on the synthesis time. The results are presented in Fig. 8(c). Though the number of total flows in the network remains unchanged in this case, the increase in the number of connectivity requirements adds more constraints which decreases the possible options for a satisfiable model. Hence, the synthesis time increases.

Impact of the Tight and Relaxed Constraints: We analyzed the impact of the tight or relaxed constraints on the model synthesis time. Tightening (relaxing) the network isolation or usability constraint means to increase (decrease) the associated constraint value. On the other hand, tightening (relaxing) the deployment cost constraint means to decrease (increase) the constraint value. The analysis results are shown in Fig. 9(a) and Fig. 9(b) varying the isolation constraint and the deployment cost constraint, respectively. In these analyses, we considered a fixed number of hosts (300) and a fixed volume of connectivity requirements (10% of all flows) in two different network usability constraints (3 and 5 in a scale of 10). We observed that the execution time increases significantly with the increase of the network isolation constraint (see Fig. 9(a)). This is due to the reason that increasing the isolation constraint reduces the number of possible solutions to the model with respect to a particular usability constraint and a specific deployment budget. As a result, usually an increased search (*i.e.*, a longer time) is required to find a solution. After a certain value of the isolation constraint (*i.e.*, 3 in Fig. 9(a)), when the requirement is already tight, a small increase in the constraint increases the synthesis time sharply.

We also observed almost similar behavior in the case

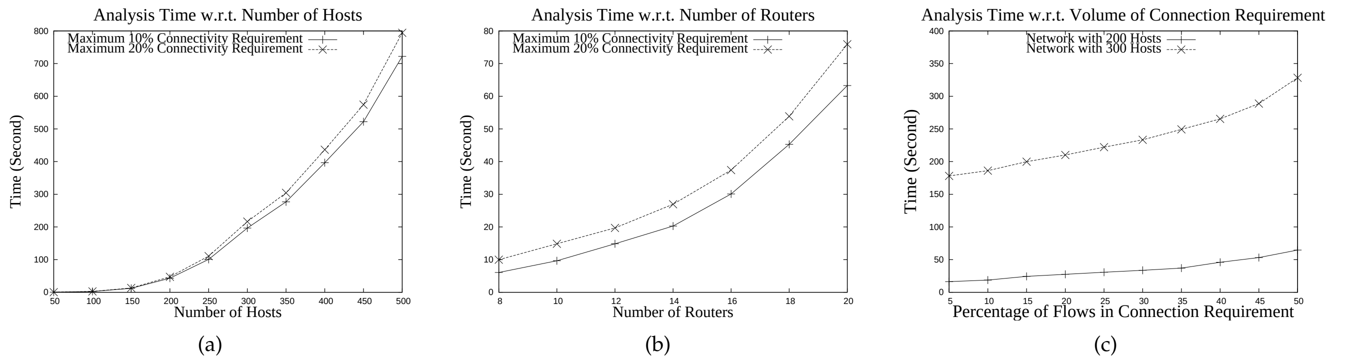


Fig. 8. The security architecture synthesis time with respect to: (a) the number of hosts, (b) the number of routers, and (c) the volume (in percentage) of the connectivity requirements.

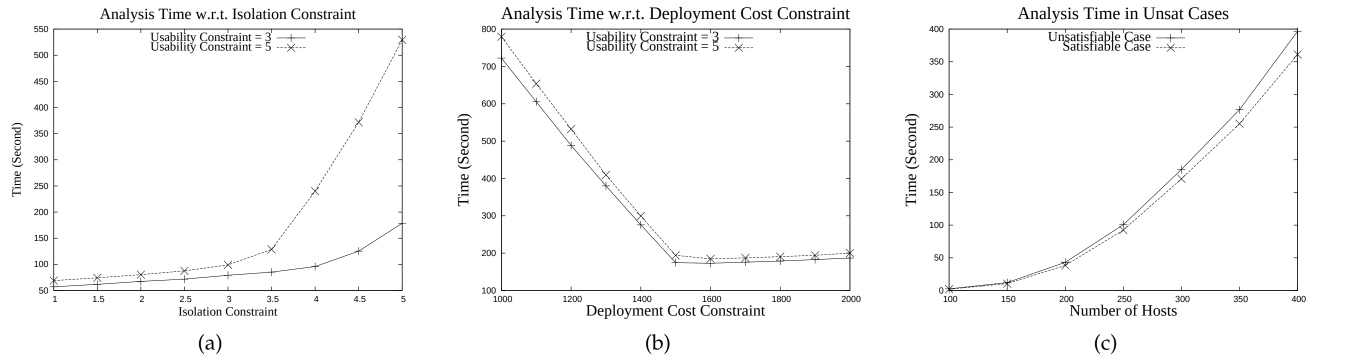


Fig. 9. (a) The impact of the isolation constraint on the model synthesis time, (b) the impact of the deployment cost constraint on the model synthesis time and (c) the comparison between the satisfiable and unsatisfiable cases with respect to the model synthesis time.

of the deployment cost constraint (see Fig. 9(b)). In this case, the higher the budget is, the more satisfiable options are available. Hence, the synthesis time decreases with the increase of the budget. We observed that after a certain increase in the budget ($1500K), the synthesis time does not decrease further. The number of potential satisfiable models does not increase any more despite increasing the budget.

Performance in the Unsatisfied Cases: In the cases of very tight constraints (*e.g.*, very high values for isolation and usability constraint or low values for the cost constraint), there may not be any satisfiable model. In these cases, the SMT solver takes slightly longer time to give the unsatisfiable (*UNSAT*) results compared to the time required in the satisfiable cases. Because, in an unsatisfiable case, the SMT solver requires verifying all possible ways to conclude that there is no solution satisfying all of the given constraints. Fig. 9(c) shows the comparison between the satisfiable and unsatisfiable cases with respect to the synthesis time.

Memory Requirement: We evaluated the memory requirement for executing ConfigSynth with respect to the SMT solver [3]. We varied the number of hosts to understand the impact of the problem size on the memory requirement. The memory requirement actually specifies the memory required for modeling the synthesis problem, which is the sum of the memory for modeling the system parameters and that for modeling the constraints. The analysis results are shown in Table 8 in two different scenarios of the network isolation constraint. In the first scenario, the isolation constraint is 3 (in a scale of 10), while in the second scenario, this is 5. We observed that the memory requirement increases quadratically ($O(N^2)$) with the increase in the number of hosts. The table shows that the memory requirement in the second scenario is larger than the memory requirement in the first scenario. If the isolation constraint is high, the solver requires searching more options for a satisfiable solution, which incurs more memory.

TABLE 8
The memory requirement (MB) w.r.t. problem size

Hosts	Scenario 1	Scenario 2
100	1.93	2.39
200	6.71	6.59
400	30.48	41.72
600	113.99	160.70
800	376.21	532.89

4.3.3 *Performance of ConfigSynth Refinement Framework*

Fig. 10(a) and Fig. 10(c) show the execution time of the ConfigSynth refinement mechanism with respect to the number of hosts and the number of routers, respectively. In the first case, where we varied the problem size with respect to the number of hosts, we considered two different sizes of connectivity requirements. In one scenario, the volume of the connectivity requirements is 10% of all possible flows, while in the second scenario, the percentage is 20%. The execution times are shown in Fig. 10(a). We observed that the refinement time increases rapidly with the number of hosts. This is because the time for executing the ConfigSynth framework, as we have already seen in Fig. 8(a), follows a quadratic order. Moreover, the refinement process needs to execute ConfigSynth for many times. However, the overall time of executing the refinement mechanism is not simply the multiplication of the number of times the ConfigSynth framework is executed. This is because the security architecture is often improved based on an initial security architecture with a relaxed isolation requirement, and the execution

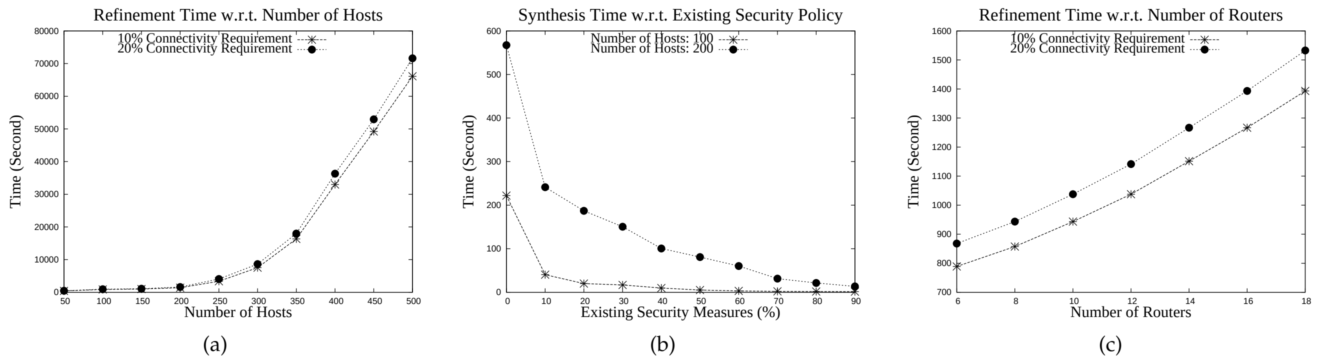


Fig. 10. The security architecture refinement mechanism execution time with respect to: (a) the number of hosts, (b) the existing security, and (c) the number of routers.

time of ConfigSynth in such a case is significantly smaller compared to that of ConfigSynth in a clean state. Fig. 10(b) justifies this argument, which represents the execution time of ConfigSynth with respect to the size of security configurations that are already existing. We observed significant decrease in the synthesis time even when only 10% of the security configurations preexist. It is also worth mentioning that the isolation, attained through the refinement process, most often cannot be achieved within the execution time of the refinement process, if the same isolation is given as the initial isolation requirement.

In the second case, the core network is varied by changing the number of routers. We again considered the same two different connectivity requirements. The results are presented in Fig. 10(c). As we have explained in Section 4.3.2, due to the increase in the number of routers, the core network becomes larger and more links turn out as the candidates for security device placements. Therefore, the synthesis time increases, so does the execution time of the refinement mechanism increases.

5 Discussion

This section briefly discusses the limitations of this work with respect to the utility and the scalability of ConfigSynth.

5.1 Utility

The synthesis of the security architecture depends on three metrics: isolation, usability, and deployment cost. These metrics are formally defined, which include the relation between one another and the calculation of each metric's score. Each score is compared with an associated constraint, which is a quantitative value and set using a slider (*e.g.*, a value between 0 to 10). A security administrator requires to set the isolation, usability, and deployment cost constraints by selecting the corresponding sliders. As we discussed in Section 2.7.1, these slider values, especially that correspond to isolation and usability constraints, cannot provide a clear understanding of the security or usability status. Moreover, as there is no exact comparison of the security measures, a precise scoring of these measures is quite impossible. Although it is possible to order them according to the pairwise relative performance comparisons, this can only provide an abstract understanding of the security. Therefore, an exact assessment of the security measures with regards to the isolation capability will increase the efficacy of ConfigSynth. A similar argument is true for the usability scores.

5.2 Scalability

Our evaluation results show that the time and memory requirements of ConfigSynth increase quadratically with the increase in the problem size. However, a synthesis problem with 500 hosts (*i.e.*, several thousands of flows) needs 800 seconds and 100 MB memory to find a satisfiable solution when the given network is in a clean state. Apparently, it may seem that this number of hosts is small compared to a large enterprise network. However, in most of the large networks, usually many of the hosts exhibit similar properties. They are running the same OS, services, and even operated by the same level of users (*e.g.*, a student lab in a university or a customer service center in an organization). They usually reside under the same subnet. The security configuration for such a group is expected to be the same. Therefore, this group can easily be assumed as a single host. Moreover, in an enterprise network the overall number of services running on the hosts are also limited. Thus, our model is adequately efficient for an enterprise network.

If there are some security measures including security devices already deployed, the execution time is found to be significantly smaller (refer to Section 4.3.3). Therefore, the rectification of the security design of a large enterprise network, which is mostly the case for an existing network, can readily be done with the help of ConfigSynth. The proposed security architecture refinement mechanism is developed leveraging this incremental design concept and it can find the near-optimal security architecture in a plausible time (refer to Section 4.3.3). A parallel execution of the iterative steps of the refinement mechanism would significantly increase the scalability of the security architecture design process. This idea remains as a future extension to this work.

6 Related Work

Throughout the last decade, security policy misconfigurations have been studied extensively in literature [5], [6], [7], [8], [9], [10], [11]. In these works, the formal definition of security configuration anomalies with respect to the deployment of network security devices, mainly firewalls, have been presented. Different algorithms were also proposed to discover configuration inconsistencies. Unlike this static and offline analysis of network security, Khurshid et al. recently proposed VeriFlow, a runtime and fast security verification framework [12]. This framework leverages Software Defined Networking (SDN) layers and verifies the new forwarding entries with the security constraints by placing itself between the controller and the forwarding devices. FlowGuard is another security analysis framework proposed for SDN-based dynamic networks [13]. This framework is capable of verifying the accuracy of the firewall policy while the networking states are frequently changing. It can also perform resolution of firewall policy violations. These works follow the traditional bottom-up approach of analyzing existing security policies, which can only help in trial and error-based design of security architectures, but cannot be used to automatically synthesize policies based on security and business requirements.

Several researches have been done on attack graph-based security configuration analysis. Sheyner et al. proposed a symbolic model checking-based mechanism for automated generation and analysis of attack graphs [14]. A similar research was performed in [15], [16] where a declarative logic-based approach was used to develop an efficient attack-graph generation tool, named MulVAL. Few works have been proposed to find security hardening measures using attack graphs by limiting potential attack propagations [17], [18]. In [19], the authors model the selection of security hardening measures to minimize the residual damages in a predefined attack graph within a certain budget. In another work, a technique was proposed to place IDS sensors and prioritize IDS alarms using attack graph analysis such that the IDS sensors are placed to cover all these paths [20]. Lippman et al. proposed a tool named NetSPA in [21] for ensuring defense in depth in a network. NetSPA uses attack graphs to measure the network security by drawing predictive graphs and provides an ordered or prioritized list of defense recommendations based on the effects of different vulnerability patches on mitigating the identified attack graphs. Homer and Ou proposed a network security management framework in [22] by analyzing attack graphs. This framework first uses MulVAL to generate all possible attack paths, creates SAT-based Boolean logics relating configurations with the attacker's actions, usability constraints, and finally solves the model to find the network reconfiguration plan (*e.g.*, vulnerability patching, access blocking, etc.). Each security measure is associated with a cost, which is minimized using MinCostSAT. However, these works do not consider the placement of network security devices (*e.g.*, firewall, IPSec, IDS, etc., altogether) which is crucial for implementing security measures within the deployment budget. Moreover, these works need to generate all attack graphs to find necessary security configurations.

Risk analysis and security hardening have also received a lot of attention from the researchers. In [23], the authors present a methodology to model the composition of vulnerabilities as attack graphs obtainable from topological vulnerability analysis (TVA) system. By analyzing attack graph, they explore different concepts and issues on a metric to quantify potential attacks. Singhal and Xou describe the security metrics to compute the overall risk in an enterprise system in [24]. They present an attack graph-based system architecture and security metrics for an enterprise network in order to quantify the overall risk, which are essential for the decision makers in taking sensible security management. In [25], Ahmed et al. presented a framework of security metrics that objectively quantifies security risk factors considering both the network security policy and the services running on hosts. In [26], we presented a generic formal model that can assess the network risk by considering transitive reachability for understanding potential attack paths. We also presented an automated firewall policy generation mechanism by modeling the problem as a constraint satisfaction framework. However, none of these works address the problem of automatic security architecture design considering security and business constraints.

The research on the security design synthesis is in a premature stage. ConfigAssure is a requirement solver presented in [27]. The tool takes security requirements and configuration variables as inputs and produces the values of the configuration variables as outputs that make the requirements true. ConfigAssure requires complete and well defined properties and it can not reason about the optimal configuration based on isolation, usability and deployment cost. In [28], FADES (Formal Analysis and Design approach for Engineering Security) is proposed with the objective of bridging the gap between formal requirements and design for security requirements. In the works [29], [30], Zhang and Al-Shaer presented procedural approaches of generating firewall configurations. In [30] they also considered the device deployment cost. However, these works only describe generation of firewall policy configurations and do not consider different isolation measures (*i.e.*, firewalls, IPSec, IDS, etc.) in the context of security requirements, usability satisfaction, and deployment cost constraints. In addition, these works cannot do the optimal placements of security devices in the network. Hence, none of the above works generates a security design architecture considering the security requirements and business constraints exploring various security design alternatives in determining satisfiable security configurations, which is the major thrust of this research work.

7 Conclusion

In this paper, we have presented an automated framework, ConfigSynth, for synthesizing correct and cost-effective network security configurations. It formally models the network topology, hosts and possible traffic flows, security requirements in terms of isolation, placements of necessary security devices distributed in the network, and the organizational business constraints in terms of usability and deployment cost, along with different invariant and user-defined constraints. Then, the framework formalizes the security design synthesis problem as the conjunction of all the requirements and constraints. It solves the problem using an efficient SMT solver that results in an optimal network security design along with optimal placements of security devices. We have also developed a refinement mechanism utilizing ConfigSynth that adapts the idea of hypothesis testing to find an improved security architecture in a scalable manner. We evaluated ConfigSynth as well as the refinement mechanism in different synthetic networks and found that the proposed solutions scale reasonably well with respect to the problem size. In future, we would like to extend our model in order to incorporate host and application level isolation patterns. We would also like to address especial cases like the security against denial of service attacks, which needs a mixed use of security devices.

In our future work, we would like to develop a formal framework for the automated design of resilient networks considering diversity, redundancy, and recoverability.

References

[1] L. de Moura and N. Bjørner, "Satisfiability modulo theories: An appetizer," in *Brazilian Symposium on Formal Methods*, 2009, pp. 23–36.

[2] M. Rahman and E. Al-Shaer, "A formal approach for network security management based on qualitative risk analysis," in *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, May 2013, pp. 244–251.

[3] L. de Moura and N. Bjrner, "Z3: An efficient smt solver," in *Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2008, pp. 337–340.

[4] A. Morgado, F. Heras, M. Liffiton, J. Planes, and J. Marques-Silva, "Iterative and core-guided maxsat solving: A survey and assessment," *Constraints*, vol. 18, no. 4, pp. 478–534, Oct. 2013.

[5] E. Al-Shaer and H. Hamed, "Discovery of policy anomalies in distributed firewalls," in *Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, March 2004, pp. 2605–2616.

[6] H. Hamed, E. Al-Shaer, and W. Marrero, "Modeling and verification of ipsec and vpn security policies," in *Network Protocols, 2005 13th IEEE International Conference on*, Nov 2005, pp. 259–278.

[7] L. Yuan, H. Chen, J. Mai, C.-N. Chuah, Z. Su, and P. Mohapatra, "Fireman: a toolkit for firewall modeling and analysis," in *IEEE Symposium on Security and Privacy*, May 2006, pp. 199–213.

[8] C. C. Zhang, M. Winslett, and C. A. Gunter, "On the safety and efficiency of firewall policy deployment," in *IEEE Symposium on Security and Privacy*, 2007.

[9] E. Al-Shaer, W. Marrero, A. El-Atawy, and K. Elbadawi, "Network configuration in a box: towards end-to-end verification of network reachability and security," in *IEEE International Conference on Network Protocols*, Oct 2009, pp. 123–132.

[10] P. Bera, S. Ghosh, and P. Dasgupta, "Policy based security analysis in enterprise networks: A formal approach," *IEEE Transactions on Network and Service Management*, vol. 7, no. 4, pp. 231–243, December 2010.

[11] M. Rahman and E. Al-Shaer, "A declarative approach for global network security configuration verification and evaluation," in *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, May 2011, pp. 531–538.

[12] A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P. B. Godfrey, "Veriflow: Verifying network-wide invariants in real time," in *the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2013, pp. 15–27.

[13] H. Hu, W. Han, G.-J. Ahn, and Z. Zhao, "Flowguard: Building robust firewalls for software-defined networks," in *the 3rd Workshop on Hot Topics in Software Defined Networking*, 2014, pp. 97–102.

[14] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing, "Automated generation and analysis of attack graphs," in *IEEE Symposium on Security and Privacy*, 2002, pp. 273–.

[15] X. Ou, S. Govindavajhala, and A. W. Appel, "Mulval: A logic-based network security analyzer," in *Proceedings of the 14th Conference on USENIX Security Symposium*, 2005.

[16] X. Ou, W. F. Boyer, and M. A. McQueen, "A scalable approach to attack graph generation," in *Proceedings of the 13th ACM Conference on Computer and Communications Security*, 2006, pp. 336–345.

[17] S. Noel, S. Jajodia, B. O'Berry, and M. Jacobs, "Efficient minimum-cost network hardening via exploit dependency graphs," in *Computer Security Applications Conference, 2003. Proceedings. 19th Annual*, Dec 2003, pp. 86–95.

[18] I. Kotenko and M. Stepashkin, "Attack graph based evaluation of network security," in *10th IFIP International Conference on Communications and Multimedia Security*, 2006, pp. 216–227.

[19] R. Dewri, N. Poolsappasit, I. Ray, and D. Whitley, "Optimal security hardening using multi-objective optimization on attack tree models of networks," in *14th ACM Conference on Computer and Communications Security (CCS)*, 2007, pp. 204–213.

[20] S. Noel and S. Jajodia, "Attack graphs for sensor placement, alert prioritization, and attack response," in *Cyberspace Research Workshop of Air Force Cyberspace Symposium*, 2007.

[21] R. Lippmann, K. Ingols, C. Scott, K. Piwowarski, K. Kratkiewicz, M. Artz, and R. Cunningham, "Validating and restoring defense in depth using attack graphs," in *IEEE Military Communications Conference (MILCOM)*, Oct 2006, pp. 1–10.

[22] J. Homer and X. Ou, "Sat-solving approaches to context-aware enterprise network security management," *IEEE JSAC Special Issue on Network Infrastructure Configuration*, vol. 27, no. 3, pp. 315–322, Apr. 2009.

[23] L. Wang, A. Singhal, and S. Jajodia, "Measuring the overall security of network configurations using attack graphs," in *21st Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, 2007, pp. 98–112.

[24] A. Singhal and X. Ou, "Techniques for enterprise network security metrics," in *5th Annual Workshop on Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies*, 2009, pp. 25:1–25:4.

[25] M. S. Ahmed, E. Al-Shaer, M. Taibah, and L. Khan, "Objective risk evaluation for automated security management," *Journal of Network and Systems Management*, vol. 19, no. 3, pp. 343–366, Sep. 2011.

[26] M. A. Rahman and E. Al-Shaer, "A formal approach for network security management based on qualitative risk analysis," in *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, May 2013, pp. 244–251.

[27] S. Narain, G. Levin, S. Malik, and V. Kaul, "Declarative infrastructure configuration synthesis and debugging," *Journal of Network and Systems Management*, vol. 16, no. 3, pp. 235–258, Sep. 2008.

[28] R. Hassan, S. Bohner, S. El-Kassas, and M. Hinchey, "Integrating formal analysis and design to preserve security properties," in *42nd Hawaii International Conference on System Sciences*, Jan 2009, pp. 1–10.

[29] B. Zhang, E. Al-Shaer, R. Jagadeesan, J. Riely, and C. Pitcher, "Specifications of a high-level conflict-free firewall policy language for multi-domain networks," in *Proceedings of the 12th ACM Symposium on Access Control Models and Technologies*, 2007, pp. 185–194.

[30] B. Zhang and E. Al-Shaer, "On synthesizing distributed firewall configurations considering risk, usability and cost constraints," in *Network and Service Management (CNSM), 2011 7th International Conference on*, Oct 2011, pp. 1–8.

Mohammad Ashiqur Rahman received his PhD in Computing and Information Systems from the University of North Carolina at Charlotte (UNC Charlotte), USA, in 2015. Earlier, he received his BSc and MSc degrees in Computer Science and Engineering from Bangladesh University of Engineering and Technology (BUET), Dhaka, in 2004 and 2007, respectively. He joined the Department of Computer Science at Tennessee Tech University as an Assistant Professor in August 2015. His primary research interests include security analysis and automation, risk analysis and security hardening, and security policy verification and optimal management. His research area covers cyber security and management for both general networks as well as cyber physical systems. He has already published over 30 peer-reviewed journals and conference papers.

Ehab Al-Shaer is a Professor and the Director of the Cyber Defense and Network Assurability (CyberDNA) Center in the College of Computing and Informatics at University of North Carolina Charlotte. He received his MSc and Ph.D. in Computer Science from the Northeastern University (Boston, MA) and Old Dominion University (Norfolk, VA) in 1998 and 1994, respectively. His primary research areas are network security, security management, fault diagnosis, and network assurability. Prof. Al-Shaer edited/co-edited more than 10 books and book chapters, and published about 200 refereed journal and conferences papers in his area. Prof. Al-Shaer also served as a Conference/Workshop Chair and Program Co-chair for a number of well-established conferences/workshops in his area including IM 2007, POLICY 2008, ANM-INFOCOM 2008, ACM CCS 2009-2010. He served as a member in the technical programs and organization committees for many IEEE and ACM conferences.