FORMAL ANALYSIS OF SECURITY POLICY IMPLEMENTATIONS IN ENTERPRISE NETWORKS

P Bera[1], Pallab Dasgupta[2] and S K Ghosh[1]

[1]School of Information Technology
[2]Department of Computer Science & Engineering
Indian Institute of Technology, Kharagpur, 721302, India.
Email: bera.padmalochan@gmail.com, pallab@cse.iitkgp.ernet.in, skg@iitkgp.ac.in

ABSTRACT

The management of security, operations and services in large scale enterprise networks is becoming more difficult due to complex security policies of the organizations and also due to dynamic changes in network topologies. Typically, the global security policy of an enterprise network is implemented in a distributed fashion through appropriate sets of access control rules (ACL rules) across various interface switches (layer 3 switches) in the network. In such networks, verification of the ACL implementations with respect to the security policies is a major technical challenge to the network administrators. This is difficult to achieve manually, because of the complex policy constraints (temporal access constraints) and the presence of hidden access paths in the network which may in turn violate one or more policy rules implicitly. The inconsistent hidden access paths may be formed due to transitive relationships between implemented service access paths in the network. Moreover, the complexity of the problem is compounded due to dynamic changes in network topologies. In any point of time, the failure of the network interfaces or links may change the network topology as a result alternative routing paths can be formed for forwarding various service packets. Hence, the existing security implementation (distribution of ACL rules) may not satisfy the policies. In this paper, a fault analysis module is incorporated along with the verification framework which as a whole can derive a correct ACL implementation with respect to a given security policy specification and can ensure that a correct security implementation is fault tolerant to certain number of link failures. The verification module can find the correct security implementation and the fault analysis module can find the number of link failures the existing security implementation can tolerate and still satisfy the security policy of the network.

KEYWORDS

LAN, Network Security, Security Policy, Access control lists (ACL), SAT based verification.

1. INTRODUCTION

The services, operations and management of today's organizations (industrial, commercial, academic etc) are becoming increasingly dependent on their enterprise LAN. Usually, these LANs consist of a set of sub-networks or network *zones* (logical group of network elements or entities) corresponding to different departments or sections, connected through various interface switches (typically, Layer-3 switches). The network service accesses between these zones and also with the external network (e.g., Internet) are governed by the *global network security policy* of the organization. The global security policy of the network is defined as a collection of "*permit"* and *"deny"* service access rules across various network zones where the services referred any network applications conforming to TCP/IP protocol. For example, some of the well-known network services are *ssh*, *telnet*, *http* etc. This global security policy is realized by configuring the zone interfaces with appropriate sets of access control lists (ACLs).

The major challenges related to the policy based security management in an enterprise LAN are stated as follows:

- To verify whether the security implementations (distribution of ACLs in the interface switches) conform to the global policy.
- To analyze whether a correct security implementation conforms to the global policy under dynamic changes in network topology.

The first issue states, given the global security policy of the LAN, the switches in the network must be programmed in such a way that the ACL rules in these switches together guarantee correct and precise implementation of the global security policy. In other words, we must guarantee that service paths in the network which violate one or more global policy rules must be denied by the combination of switches on the path through their local ACL rules. As an example, consider a typical enterprise network shown in **Fig.1**. The example network is deployed as hierarchical networking architecture consisting of *Core*, *Distribution* and *Access* layers. Access layer includes two network zones, namely, *ZONE_1* and *ZONE_2*. Moreover, the zones can be partitioned into sub-zones. For example, in **Fig.1**, *ZONE_1* is partitioned into *SUBZONE_11*, *SUBZONE_12* and so on. Here, the *Core* includes three routers (R1, R2 and R3), the distribution network consists of two routers (R4 and R5). The external access to Internet is realized through *PROXY* zone (consist of web proxy servers).

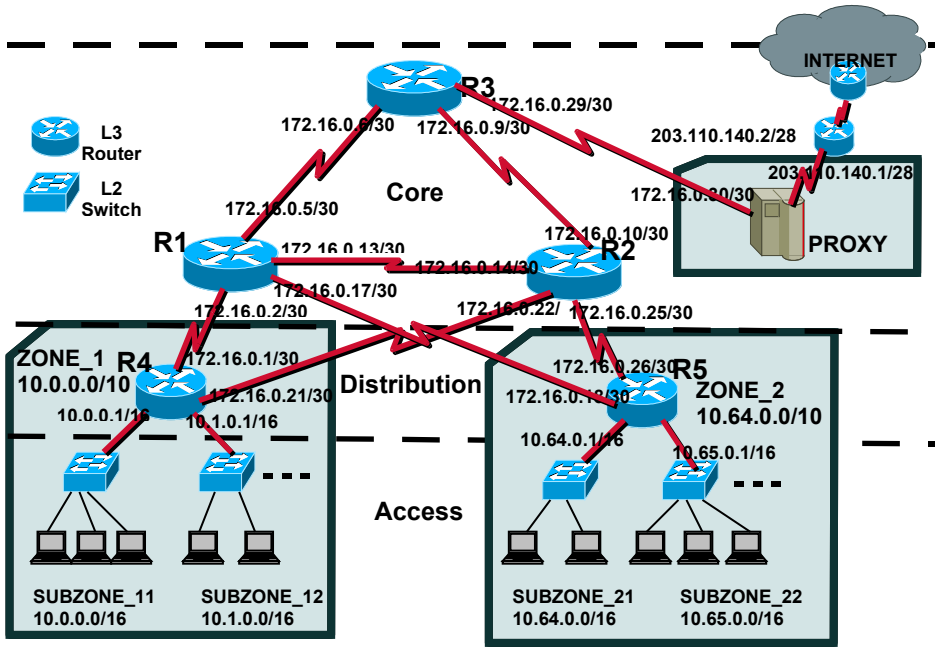

Fig.1 A Typical Enterprise Network

Now, consider the following global security policy for the network: ***"Internet (http) access is NOT allowed from the ZONE_1"***. This policy rule must be implemented by programming the ACL rules in the routers interfaces. The first problem addressed in this paper is the task of verifying that a given implementation is correct. The verification problem is more complex than it appears at first glance. Firstly the security policies may be temporal, such as an organization has the requirement that the *http* access to be blocked between 0900-1700 hours in weekdays from a particular network zone. Therefore it is required to verify correctness at the various windows of time. Secondly, network services may be combined to define service access paths which are indirect or *hidden*. For example, even if direct http access from the *ZONE_1* to the *PROXY* in **Fig.1** is blocked (say, by placing an appropriate deny rule in the router, R4), one can *ssh/telnet* from *ZONE_1* to *ZONE_2* (which may be allowed from *ZONE_1*) and access Internet

(which may be allowed for *ZONE_2*). Therefore, even though there is no direct http access path from *ZONE_1*, the actual intent of preventing internet access from that zone is not guaranteed. The combination of network services that can create hidden access paths for a specific service needs to be known for verification. This constitutes a significant amount of domain knowledge in the verification problem. In order to accommodate new types of services and thereby new combinations of network services, our decision framework provides formalism for specifying *hidden access rules*. In our previous work [24], a formal verification framework has been proposed to analyze the security policy implementations in an enterprise network considering these problems.

In our earlier work [24], the security analysis was performed considering the network topology to be static. However, in any point of time the failure of the interfaces may change the network topology which is a common circumstance in an enterprise network. Hence, more challenging problem is to analyze a correct security implementation under network topology changes. In that case, the network service packets may be routed through alternative paths in the network hence the distribution of ACL rules may not satisfy the security policy. This paper presents a graph based network access model for analyzing the robustness of a correct ACL implementation under arbitrary link failures. Basically, it finds the maximum number of link failures the security implementation can tolerate and still satisfy the global policy of the network. The proposed *fault analysis* methodology can be layered at the bottom of the verification framework to extract a correct implementation under various network link failures.

The rest of the paper is organized as follows. Section 2 describes the related work on network security policy specification, security analysis tools and models. In section 3, the proposed verification and fault analysis framework has been described. The verification of security policy implementations along with the experimental results has been presented in section 4. Section 5 describes the fault analysis of security implementations.

2. Related Work

The research works on network security analysis and policy configurations can be broadly classified into three categories: (a) network firewall analysis algorithms and tools; (b) security policy specification languages; and (c) network security and fault analysis using formal approaches. However, none of these addresses the issue of *hidden access path analysis*. Moreover, most of the works do not consider the dynamic network topology changes in their analysis. Though the present work focuses on formal verification and fault analysis of security policy implementations, a brief overview of all the categories has been presented in this section.

Existing literatures on firewall analysis primarily concentrate on inconsistency and redundancy checks but most of those are not formally verified. Tools that allow user queries for firewall analysis and management include Firmato [1] and Lumeta [2]. These tools can specify an abstract network access control policy and firewall rules which satisfy that policy but lacks in incorporating temporal constraints and hidden rule analysis. Eronen and Ziting [3] have implemented an expert system for inconsistency detection. Al-Shaer and Hamed [4] worked on the Firewall Policy Advisor. But both of these tools can handle a simple set of policy constraints. The work by Guttman et al. [9] focuses on high level modelling of firewall and network configurations that satisfy a given policy but the policy specifications are more general.

Researchers proposed different high level network security policy languages, namely, HLFL [6], Firmato [1], FLIP [7] etc. The high level firewall language (HLFL) project is an approach to translate the high level firewall rules into useful rules for IPChains, Netfilter, and many others. The approach does an automatic translation, but it lacks important features such as detecting and preventing the conflicts in the firewall rules. FLIP, most recently proposed high level conflict free firewall policy language for enforcing access control based security and ensuring seamless configuration management. In FLIP, security policies are defined as high level service oriented

goals, which can be translated automatically into access control rules to be distributed to appropriate enforcement devices. But still it does not address temporal access constraints and hidden access path analysis. A high level language, namely, Extended Security Policy Specification Language (ESPSL) has been proposed as a part of this framework. The unique feature of the language is simple constructs for specifying temporal policy rules.

There are few works on network security analysis using formal approaches. The FIREMAN Toolkit [5] is an example to detect inconsistencies and redundancies in network of firewalls. The set of all possible requests is formulated and model checking is used to divide the set into those which are accepted, those which are rejected, and those for which no rule applies. The tool can handle large set of firewall rules since it uses an efficient BDD representation. The Network Policy Enforcement tool [11] is one of the more recent tools in this line of work. Ritchey and Amman [12] shows how model checking can be used to analyze network vulnerabilities. Another recent work is proposed by Matousek, Rysavy, Rab, and Sveda [13] on formal model for network wide security analysis. They model the network topology with changing link states and deploy bounded model checking of network security properties using SAT-based decision procedure. However, this work is unable to ensure whether a correct security implementation is fault tolerant under arbitrary link failures. Also, they enumerate all possible link state combinations in a network for their analysis which may lead to state explosion problem for large scale networks. Matsumoto and Bouhoula [16] propose a SAT based approach for verifying firewall configurations with respect to security policy requirements. However, the notion of *hidden access paths* and formalizing the verification problem in their presence has not been addressed earlier. Again, a complete framework for analyzing the correctness of the security policy implementations under network topology changes has not been proposed earlier.

3. Proposed Verification and Fault Analysis Framework

The proposed verification and fault analysis framework shown in **Fig.2** primarily focuses on the following issues:

- Conflict-free specification and modelling of the global security policy of an enterprise LAN using the proposed policy specification language, ESPSL.
- Extraction of the implementation model from the distributed access control (ACL) implementation in the network;
- Hidden access path analysis on the ACL implementation model;
- Reduction of policy and ACL implementation models into boolean functions and formulation of a QSAT (satisfiability of quantified Boolean formula) problem;
- Solving the satisfiability problem using an efficient QBF SAT solver;
- Fault analysis on a correct ACL implementation model under network link failures.

The framework consists of two interlinked modules: 1) Verification Module 2) Fault Analysis Module. The Verification module models the global security policy in an enterprise LAN using a policy specification language; then extracts an implementation model from the distributed ACL implementation corresponding to the policy; finally verifies the ACL implementation model with the policy model using boolean satisfiability analysis (SAT). The fault analysis module takes a correct ACL implementation with the underlying network topology and finds the maximum number of network link failures it can tolerate to satisfy the global policy.

A policy specification language, namely, ESPSL (Extended Security Policy Specification Language) has been proposed for modelling the global security policy. The inter-rule conflicts between the policy rules are removed by maintaining a complete rule order in the rule set. The conflict-free policy specification represents the policy model, M_P.

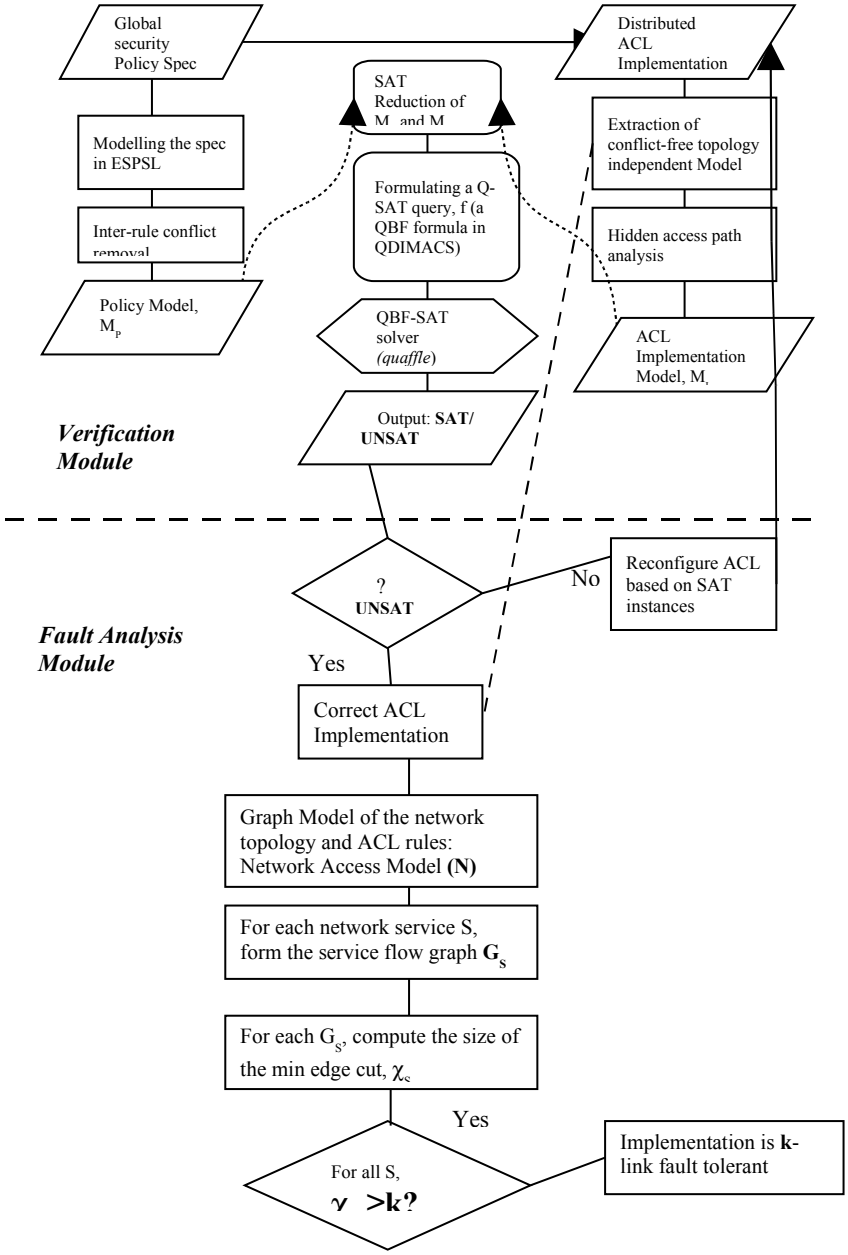

Fig.2 Proposed Verification and Fault Analysis Framework

An implementation model M_I is extracted from the distributed ACL implementation corresponding to the global policy for the network. It typically consists of set of ACL rules distributed across various network interfaces. The hidden access path analysis procedure incorporates extra rules in the implementation model based on the transitive relationships between various service access paths. The inter-rule conflicts and topology information is removed from the ACL rule base. For the purpose of verification, M_P and M_I are reduced into boolean models. It requires reduction of the rule components and their inter relations into boolean functions. The hidden access path analysis procedure is implemented by reducing the hidden rules into a set of quantified boolean clauses which makes the procedure efficient and fast. After boolean model reduction, a Q-SAT query is formulated as $f = M_P \oplus M_I$, and is represented in QDIMACS CNF standard [17]. Finally, the module solves the satisfiability of the query, *f*, using *quaffle* QBF SAT solver [15] [18]. The output from the SAT solver indicates

SAT/UNSAT, where, SAT result indicates the ACL implementation is incorrect with respect to the policy specification. Thus the verification module helps the administrator to debug the ACL rule distribution towards a correct ACL implementation.

The Fault Analysis module takes a correct ACL implementation model (correct distribution of ACL rules in the network interfaces) and the underlying network topology and finds the maximum number of link failure it can tolerate so that it still conforms the global security policy. This module stems from modelling the network topology and ACL rules as a *network access model* and then formation of the *service flow graphs* (G_S) under each network service considering the "permit"/"deny" ACL rules associated to the network access model. Then the module finds the *min-cut*, χ_S (minimum sized edge cut) for each service flow graph G_S. Finally, it finds the $\chi_N = \min(\chi_S)$ (smallest *min-cut* of all the service flow graphs) which represents the *fault tolerance* value corresponding to the correct ACL implementation under the network.

4. Verification of Security Policy Implementation

The security policy verification module consists of three sub-modules. 1) Policy specification module 2) ACL Implementation module 3) QSAT based Verification module. The Policy specification component models the global security policy in a high level language, ESPSL and produces a conflict-free security policy model (M_P). The implementation module takes the distributed ACL implementation corresponding to the global security policy as an input and extracts a conflict free topology independent implementation model (M_I). The verification module reduces the policy and implementation models into boolean clauses. It should be noted that the *hidden access path analysis* procedure is incorporated in the implementation model which typically adds new quantified rules in the boolean implementation model M_I. After reducing the models into boolean clauses, a Q-SAT query is formulated by taking exclusive-OR between the models, i.e., $f = M_P \oplus M_I$ and represented in QDIMACS CNF standard [17]. Finally, the module solves the satisfiability of the query, f, using *quaffle* QBF Solver [18].

4.1. Policy Specification Module

The security policy of a network defines a set of parameterized functional rules on flow of packets between different network zones. The specification language must express the complex security constraints correctly. As a part of the verification framework, a language, namely Extended Security Policy Specification Language (ESPSL) has been proposed. In the following section, the various constructs of the proposed language are described. Some of the language constructs described in our earlier work [24] has been modified in this paper.

4.1.1. Network Topology Specification

The ESPSL has the following constructs to describe the network topology.

Zone: A zone is a logical unit consisting of workstations, servers or other systems in the network, usually refers to a particular section of an organization. It is represented by IP address block(s) and it referred by the IP address block(s) or by a symbolic name. Further, a zone can be partitioned into multiple disjoint sub-zones. For example *ZONE_1* can be defined as follows.

```
Zone ZONE_11 [10.0.0.0-10.0.255.255]; Zone ZONE_12 [10.1.0.0-10.1.255.255];
Zone ZONE_1 [ZONE_11, ZONE_12];
```

Router: Routers are interconnecting Layer 3 switches for connecting various sub-networks. A router can be connected to a network zone or another router. It consists of set of interfaces.

Interface: An interface is the connecting link between a zone and a router or multiple routers. Each interface is identified by a unique IP address.

```
Interface L12 [172.16.0.13]; Interface L13 [172.16.0.5]; Interface L14 [172.16.0.2];
Interface L15 [172.16.0.17]; Router R1 [L12, L13, L14, LR15];
```

4.1.2. Network Service and Policy Rule Specification

The ESPSL has the following constructs to specify the network services and policy rules.

Service: Network service is defined by a network protocol and a predicate associated with it. Each predicate defines the service port numbers or port ranges.

Service http = TCP [port = 80]; Service ssh = TCP [port>20 AND port<23];

Policy Rule: A policy rule defines a service access path between a source and a destination zone under some constraints (optional). The static rules do not include temporal access constraints, whereas temporal rules include such constraints. ESPSL models only time dependent temporal constraints which can be combination of day and time range specifications.

Permit ssh(ZONE_1, ZONE_2);

Deny http(ZONE_1, PROXY) [const = week_day(0800-1700)];

4.2. ACL Implementation Module

The global security policy of an enterprise LAN is implemented through a set of access control rules (ACL) applied to various interfaces of the access switches (or routers) in a distributed manner. There are various device specific standards for specifying access control rules e.g. *Cisco standard ACL* [8]. Most of the standards are logically similar in the context of implementing basic security policy of a network. *Cisco standard ACL* has some extra features to represent temporal constraints and is widely used in large scale networks. In our approach, a model is extracted from the distributed ACL implementation corresponding to the global security policy of a network. This process involves following phases; (a) Translating ACL rules into service flow rules (b) Resolving inter-rule conflicts and topology dependency (c) Hidden access path analysis.

4.2.1. Translating ACL Implementation into Service Flow Rule base

This phase translates the distributed ACL implementation into a service flow rule base which is stored in a 3-level index structure. Each service flow rule consists of rule header and its functional clause. Rule header component holds binding information of the rule to an ACL group and a router interface. Functional clause holds the components of each ACL rule. The specifications of the functional components are similar to our policy model. The service flow rule structure is shown in **Table 1**.

Table 1. Service Flow Rule Structure

<F_Rule>:: <Rule_header> <Functional_clause>
<Rule_header>:: <router_id> <Interface_bind>
<Interface_bind>::<interface_id><acl_gr_no>
<Functional_clause>:: <action><service>(<src_IP>,<dest_IP>)[<const>]
<Service>:: <protocol> <port no | service_name>
<Action>:: permit | deny

A 3-level index structure is used to store the functional components of various ACL rules along with their bindings to ACL groups and router interfaces. First two level of the index holds the rule header and the leaf level holds the functional components. Each leaf level node can hold components of multiple rules under the same ACL group or router interface. Each leaf level node is represented as a linked list of structured nodes where each structure node holds the functional components of a single ACL rule. The service flow rule base generated from this phase does not change the semantics of the distributed ACL implementation.

4.2.2. Resolving Inter-rule Conflicts and Network Topology Dependency

The proposed SAT based verification framework aims at verifying the distributed ACL implementation with the global policy specification through reduction of the rule bases into a set of boolean clauses. So, it is required to represent the distributed ACL rules into a single ACL rule base which is inter-rule conflict free and network topology independent. It firstly requires removal of various inter-rule conflicts from the service flow rule sets associated to each router interface separately, and then merging the conflict free rule sets in a single rule base. The inter-rule conflicts may occur due to rule component dependencies as described follows.

Rule subsuming conflict: Consider a pair of ACL rules P_1 and P_2 under the same ACL group where P_1 precedes P_2;

P_1: Permit TCP X1, Y1 eq *ssh;*

P_2: Deny TCP X, Y eq *ssh*; such that $X1 \subset X$ and $Y1 \subset Y$.

Here, the source and destination of P_2 subsume those of P_1 and the rules are top-down order dependent. The pair of rules semantically means that *"ssh"* service access from any source in X to any destination in Y is denied except those where source and destination are X1 and Y1 respectively. To make these rules conflict-free (i.e., order independent), requires addition of two new rules P_2' and P_2'' in place of P_2 where,

P_2': Deny TCP (X-X1), Y eq *ssh*;

P_2'': Deny TCP X, (Y-Y1) eq *ssh*.

Similar type of conflict may occur between a pair of static and temporal ACLs under the same ACL group with same service component.

Rule Over-riding conflict: Consider a pair of ACL rules P_3 and P_4 under the same ACL group where P_3 precedes P_4 in the rule set,

P_3: Permit TCP X, Y eq *http*;

P_4: Deny TCP X, Y eq *http*.

Here P_3 overrides P_4 that means P_4 can't hold. As the rule order is top down, resolving this conflict requires deletion of P_4 from the rule base. On the other hand, consider a pair of rules P_5 and P_6,

P_5: Permit X, Y eq *ssh*;

P_6: Permit Z, W eq *ssh*; such that, $((X \subset Z) \wedge (Y \subset W)) \vee ((Z \subset X) \wedge (W \subset X))$.

In such cases where the rules hold the above dependency relation and their service and action components are identical, order-major rule (P_5) overrides the order-minor rule (P_6) which means P_6 can't hold. Resolving this conflict requires deletion of P_6 from the rule base.

The inter-rule conflict removal procedure resolves these conflicts from the rule base through selective insertion and deletion of rules to or from the rule base. Once the conflicts from each rule set are resolved, the main procedure removes the binding information and merges the rule sets into a single rule base, namely, *Conflict_Free_ACL_Base* which is network topology independent. Then, the *hidden access path analysis* procedure is applied on the rule base, *Conflict_Free_ACL_Base*, which is described in the following section.

4.2.3. Hidden Access Path Analysis

The hidden service access paths may exist in a network due to the transitive access relationships between various known network services. In section 1, it has been shown how hidden *http* access paths may appear in a network through conflicting *ssh* service. The *hidden access paths* can be modelled through a set of formulas in predicate logic. For example, the hidden *http* access paths can be formally represented as follows.

$\forall X \forall Z((\exists Y, ssh(X,Y) \wedge http(Y,Z) \Rightarrow http(X,Z))$--------(1)

$\forall X \forall Z((\exists Y, telnet(X,Y) \wedge http(Y,Z) \Rightarrow http(X,Z))$--------(2)

$\forall X \forall Z((\exists Y, \exists T, ssh(X,Y)[T] \wedge http(Y,Z)[T] \Rightarrow http(X,Z)[T])$--------(3)

$\forall X \forall Z((\exists Y, \exists T, telnet(X,Y)[T] \wedge http(Y,Z)[T] \Rightarrow http(X,Z)[T])$--------(4)

As mentioned above, the formulas (1) and (2) represent *static hidden http access paths* whereas, (3) and (4) represent the *temporal hidden http access paths* with some time constraint T. Here, X, Y, Z $\in$ All_Zone represent network zones and All_Zone represents total (internal) network which is *disjoint union* of distinct network zones.

For the proper assessment of the implementation with respect to the policy model, all such hidden access paths should be incorporated in the implementation model. Basically, the proposed hidden access path analysis procedure modifies the inter-rule conflict free rule base, *Conflict_Free_ACL_Base*, by inserting new rules or updating the access permissions of existing rules. The procedure consists of two major phases. First phase modifies the rule base by analyzing transitivity between every pair of rules under similar service types and the second phase modifies the rule base by analyzing such relationships between every pair of rules under different service types. This procedure is directly implemented in SAT reduction phase of implementation model depicted in next section. It basically reduces the set of predicate logic formulas (representing the hidden access rules) into set of quantified boolean clauses and incorporates those clauses to the boolean model corresponding to *Conflict_Free_ACL_Base*. The complete boolean model generated through this procedure represents the implementation model M_I .

4.3. QSAT Based Verification Module

QSAT based approach reduces the verification problem into a quantified boolean formula *f* and checks its satisfiability. Although satisfiability analysis is NP complete problem, still this technique is becoming popular today due the tremendous time tradeoffs of modern SAT solvers [10] and QBF SAT solvers [18] [19]. In our approach, both the security policy and implementation models are reduced into set of boolean clauses, M_P and M_I respectively. Then satisfiability of the formula $f = M_P \oplus M_I$ is checked using a QBF SAT solver. It requires translation of the formula into conjunctive normal form (CNF). The formula, *f*, contains clauses over quantified variables due to the inclusion of hidden access path analysis in M_I. So, it is translated into QDIMACS CNF [17] format and satisfiability is checked using *quaffle* QBF SAT solver [18].

4.3.1. Boolean Reduction of the Models

In this phase, policy and implementation rule bases are reduced into a set of boolean clauses. The rules in the models are broadly classified into two major types: (a) Generic access control rules which are common to both models (b) Hidden access rules which are specific to the implementation model. Although the boolean model reduction algorithms are nit the scope of this paper but these procedures are briefly described here.

Policy Model Reduction: The policy model consists of a set of generic access control rules. The reduction of the policy model firstly requires the mapping of the generic rule components into a set of boolean variables. The rule components include service (protocol, port number), source zone, destination zone and time constraint. A network zone can be specified as single IP address or range of IP addresses. So, source and destination zones are mapped to 32 boolean variables each. A range of IP addresses can be translated using disjunction ($\vee$) operator. Address ranges with masks can be reduced by bit-wise anding (&&) the masks with the base addresses. Similarly, protocol type and port numbers are mapped into appropriate boolean variables. In both the models, time constraints are modelled as disjunction of its valid periods.

Each valid period can contain day of week, day of a month, and hours in a day as component. Each component of a valid period is mapped into a set of boolean variables.

Algorithm 1: **Boolean Reduction of Policy Model**

Functional mapping of rule components into boolean variables:

Protocol(P):FP(p_0 ,p_1)

PortNo(I):FI(i_0,i_1,...,i_7)

Src_IP(SIP):FS(s_0,s_1,...,s_{31})

Dst_IP(DIP):FD(d_0,d_1,...,d_{31})

Time(T):FT(dt_0,dt_1,dt_2,t_0,..,t_4)

Action(g):A(g)

Algorithm:: ***Reduce_Pol_Model()***

Input: Policy Rule Base {PR_1, PR_2,.., PR_N}

Output: Reduced Policy Model M_P

1. BEGIN
2. X_1=1/True
3. FOR each policy rule PR_i (i=1 to N)
4. R_i =Reduce_Gen_Rule(PR_i)
5. $X_{i+1} \Leftrightarrow ((X_i \vee R_i) \wedge g_i) \vee ((X_i \wedge \neg R_i) \wedge \neg g_i))$
6. END FOR
7. $M_P \Leftrightarrow X_{N+1}$
8. END

Where, [g_i =1, if action(R_i) = "permit"
= 0 , if action(R_i)= "deny"].

Procedure: ***Reduce_Gen_Rule()***

Input: A generic rule PR_i

Output: Boolean Reduction of the rule PR_i

1. BEGIN
2. $P_i \Leftrightarrow FP_i(p_0,p_1) \wedge$
3. $I_i \Leftrightarrow FI_i(i_0,i_1,..,i_7) \wedge$
4. $Serv_i \Leftrightarrow (P_i \wedge I_i) \wedge$
5. $SIP_i \Leftrightarrow FS_i(s_0,s_1,..,s_{31}) \wedge$
6. $DIP_i \Leftrightarrow FD_i(d_0,d_1,..,d_{31}) \wedge$
7. $T_i \Leftrightarrow FT_i(dt_0,dt_1,dt_2,t_0,..,t_4) \wedge$
8. $Ri \Leftrightarrow (Serv_i \wedge SIP_i \wedge DIP_i \wedge T_i)$
9. Return R_i
10. END

After rule components mapping, the policy model reduction algorithm functionally reduces each rule into a boolean clause [as conjunction of rule components], translates each clause based on the action component ("*permit/1"* or "*deny/0"*) and finally combines all translated clauses into a single boolean clause. The boolean model generated from this phase defines the policy model, M_P. The functional mapping of the policy rule components and main policy model reduction procedure is presented in ***Algorithm 1***.

Implementation Model Reduction:

This phase firstly reduces the generic ACL rules corresponding to *Conflict_Free_ACL_Base,* which is similar to policy rule reduction. Then the hidden rules are reduced sequentially and associated to the reduced generic rule base (represented by, boolean model, Y_{N+1}, refer ***Algorithm2***). It finally produces the reduced implementation model M_I .

Algorithm 2: **Boolean Reduction of Implementation Model**

Algorithm:: Reduce_Imp_Model()

Input: *Conflict_free_ACL_Base* $\{IR_1, IR_2,.., IR_N\}$

Output: Reduced Implementation model, M_I

1. BEGIN
2. Y_1=1/True
3. FOR each generic ACL rule IRi (i=1 to N)
4. $FR_i = Reduce\_Gen\_Rule(IR_i)$
5. $Y_{i+1} \Leftrightarrow ((Y_i \vee FR_i) \wedge g_i) \vee ((Y_i \wedge \neg FR_i) \wedge \neg g_i))$
6. END FOR
7. M_I = *Reduce_hidden_rule()*
8. END

Procedure: ***Reduce_hidden_rule()***

Input: Generic rule model, YN_{+1}, and Hidden rule set $\{HR_1,\ldots, HRN\}$

Output: Reduced Implementation model, M_I

1. BEGIN
2. FOR each reduced generic rule FR_i (i=1 to N)
3. $TDIP_i \Leftrightarrow DIP_i \wedge$
4. $DSIP_i \Leftrightarrow FDS_i(TDIP_i, s_0, s_1, .., s_{31})$
5. END FOR
6. *Encode_comp()*
7. $M^0 = Y_{N+1}$
8. FOR each hidden rule HR_i (i=1 to N)
9. M^1 =Update_model_bool_hidden($M^{(i-1)}$, HR_i)
10. END FOR
11. $M_I = M^N$
12. END

Procedure: *Update_model_bool_hidden*(M^0, HR_1)

Input: Boolean model M^0 and hidden rule HR_1

$[HR_1:: \forall X \forall Z, \exists Y, ssh(X,Y) \wedge ssh(Y,Z) \Rightarrow ssh(X,Z)]$

Output: Updated model M^1 with reduction of hidden rule HR_1

1. BEGIN
2. $\forall X \forall Z, \exists Y$, IF $ssh(X,Y) \in M^0$ and $ssh(Y,Z) \in M^0$ THEN
3. $M^1 \Leftrightarrow M^0 \vee ssh(X,Z)$
4. Return M^1
5. END IF
6. END

The hidden rule reduction procedure starts with some pre-processing tasks. At first, for each generic ACL rule, the destination and the source matches to the destination are mapped to set of boolean variables other than their primary mapped variables. Then the total set of distinct source, destination and time-constraint are functionally encoded to set of boolean variables. These tasks are to enhance the hidden rule reduction procedures. After that, the main procedure sequentially reduces each hidden rule into a quantified boolean formula and updates the generic ACL rule model, Y_{N+1}. Initially, Y_{N+1} is assigned to a model M^0, then it is sequentially updated by the hidden rules to produce intermediate models, M^1, M^2 and so on. The final model generated from this phase represents the reduced implementation model M_I. The procedure for the reduction of implementation model is presented in ***Algorithm 2***.

4.3.2. QBF SAT Solver and Q-SAT Query Formation

We have used *quaffle* QBF solver [18] as the verification tool. It takes Q-SAT query in standard conjunctive normal form (CNF) and checks its satisfiability. The commonly used format for storing quantified CNF formula (of QSAT problems) in ASCII files is QDIMACS format [17].

Q-SAT query for our problem: "Is the ACL implementation model (M_I) exactly equals to policy model (M_P)". So, it is sufficient to check the un-satisfiability of the expression: $f = M_P \oplus M_I$. In the previous subsection, boolean reduction of M_I and M_P has been described. In the framework, the QSAT query is translated into CNF using standard algorithm for 3-CNF satisfiability [14]. The algorithm forms truth table for every sub-expression containing disjunctions of conjunctions and converts it into CNF applying De-Morgan's rules where each clause contains at most 3 literals. For example, equivalent CNF for the SAT query f is represented as $(M_P \vee M_I) \wedge(\neg M_P \vee M_I)$. The CNF formula is represented into QDIMACS format and is provided as input to *quaffle* QBF solver. It checks the SAT or UNSAT of the formula.

4.4. Verification Results

The verification framework has been tested with various test cases of implementations under defined policy specifications in an enterprise LAN. Few of those experimentations are shown in **Table 2**. It shows number of policy and ACL rules along with the number of variables, quantified variables and clauses in the reduced QSAT query under each test case.

Table 2. Verification Results

Test Cases	P	I	V	Q	C	Quaffle Output	T_{CR}	T_{SAT}	T_E
TC1	10	10	80	21	159	SAT	2.84	7.16	1.19
TC2	23	25	88	24	198	UNSAT	4.57	8.17	0.88
TC3	23	28	88	24	201	SAT	4.63	8.34	0.75
TC4	38	38	94	28	231	UNSAT	5.21	9.33	1.17
TC5	45	32	94	22	215	SAT	5.11	8.47	0.93
TC6	45	45	90	25	245	UNSAT	5.76	9.17	1.12

P: Number of policy rules

I: Number of ACL rules in *Conflict_free_ACL_Base*

V: Number of boolean variables in the Q-SAT query

Q: Number of quantified variables in the Q-SAT query

C: Number of CNF clauses in the Q-SAT query

T_{CR}: ACL Conflict and topology dependency Removal time (in seconds)

T_{SAT}: SAT reduction time (in seconds)

T_E: *quaffle* runtime (in milliseconds)

TC: Test cases

The result shows SAT/UNSAT of the Q-SAT query (i.e.) along with the SAT reduction time and *quaffle* execution time to generate the outputs. Here, the SAT result implies that the implementation does not satisfy the policy (i.e., $M_P \neq M_I$) whereas the UNSAT implies implementation satisfies the policy (i.e., $M_P = M_I$). The SAT reduction time is much higher than the SAT execution time. The SAT reduction time is linearly dependent on the number of policy, ACL rules and hidden rule reduction time. Normally, hidden rule reduction time remains constant as the number of hidden rules is fixed in the model. On the other hand the ACL Conflict removal time is dependent on the number of ACL rules and their inter-dependency.

5. Fault Analysis of Security Policy Implementation

The pre-requisite for the fault analysis module is a correct ACL implementation with respect to the global policy specification. The correct ACL implementation is extracted by the verification module of the framework described earlier. The flow diagram for the fault analysis procedure is shown in **Fig.3** To start with, the network with the ACL rule distribution is modeled as a graph described as follows.

5.1. Network Access Model

Definition 1: A network access model is defined as a 3-tuple N = <D, I, F>**,** where,

- D is a finite set of network devices. Again network devices (D) can be of two types: DR indicates a finite set of network routers and DE indicates the end point devices (network zones, workstations or web servers etc)
- $I \subseteq \{D \times D\}$ is a finite set of network interfaces between network devices, such that for every physical interface between D_1 and D_2 there is a pair of lines or channels: $I_{12} = <D_1, D_2>$ and $I_{21} = <D_2, D_1>$. Again, network interfaces (I) can be of two types: $IE \subseteq \{DR \times DE\} \mid \{DE \times DR\}$ indicates set of network interfaces which connect end point devices to routers; and $IR \subseteq \{DR \times DR\}$ indicates set of interfaces between a pair of routers.
- F is a finite set of ACL rules assigned to the edges of the graph.

Because, ACL rules can be applied in both directions of the link, we consider that the set I contain for every link two items, $<D_i, D_j>$ and $<D_j, D_i>$. Typically, there are two links between a pair of routers based on the direction of flow which are represented as separate port in our generic module. According to this definition, the network access model for the example network shown in section 1 is a graph shown in figure **Fig.3**. This network model is considered as running example for rest of the analysis.

5.1. Service Flow Graph Formation

In this phase, based on the correct ACL implementation {the ACL rules associated to various network interfaces} service flow graphs are formulated for each network service. A service flow graph is defined as follows:

Definition 2: A service flow graph $G_S<D_S, I_S>$ under the network service S is defined as a connected directed sub-graph of the network model N<D, I, F>, where $D_S \subseteq D$ and $I_S \subseteq I$ such that $\forall I_S$, $F(I_S) \bullet$ action = "permit".

The definition states that the service flow graph includes only the edges on which corresponding service access is allowed. The service flow graphs are formed by simply removing the edges from **N** which are associated with "deny" rules under a particular service. Then to keep the graph connected, the disconnecting nodes (a subset of D) are removed. For example, consider the security policy for our running network access model:

Policy1: "ssh access is not allowed from ZONE_1 to ZONE_2 whereas the reverse flow is allowed"

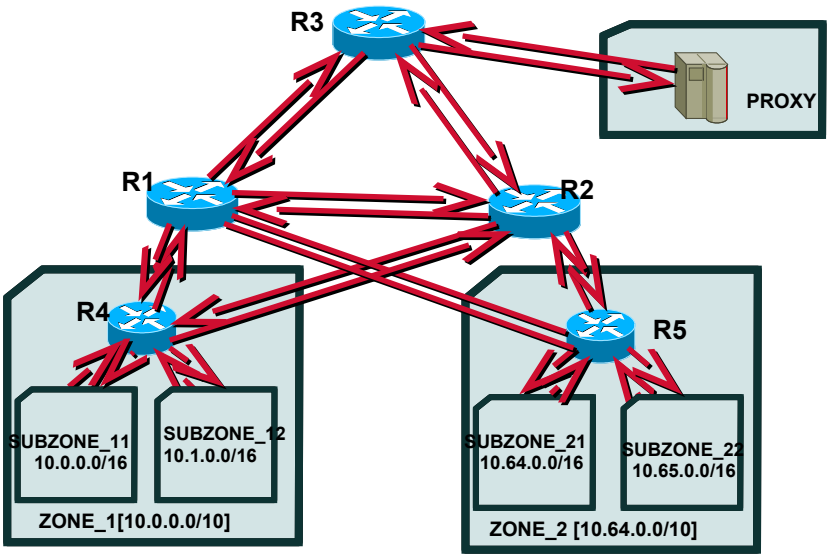

Fig.3 Network Access Model Corresponding to Fig.1

Now, consider the ACL implementation scenario corresponding to the policy:

Implementation1:

Permit ssh(ZONE_2, ZONE_1); Deny ssh(ZONE_1, ZONE_2);
Permit ssh(R1, R4); Permit ssh(R2,R4); Permit ssh(R5, R4);
Deny ssh(R1, R3); Deny ssh(R1, R2); Deny ssh(R2, R3);
Deny ssh(R4, R1); Deny ssh(R2, R5).

The *"ssh" service flow graph* corresponding to *Implementation1* is shown in **Fig.4(a).**

Similarly, consider the following security policy described in section **1**.

Policy2: "Internet (http) access is NOT allowed from the ZONE_1".

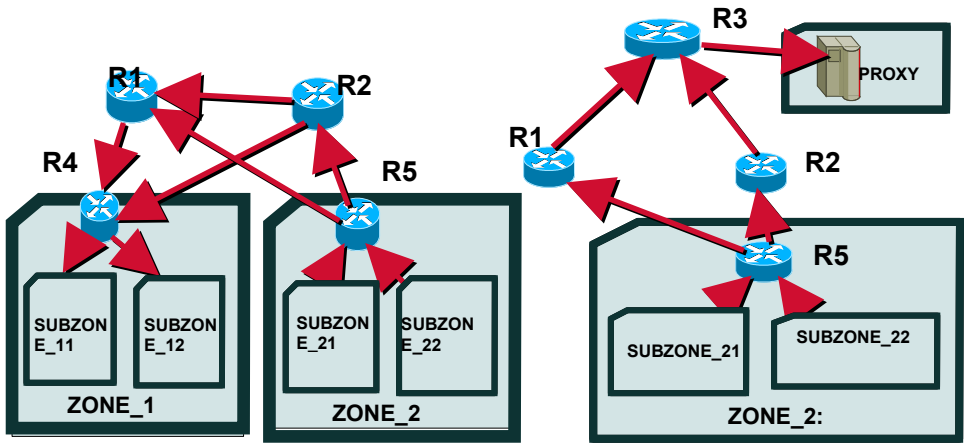

Fig.4 (a) "ssh" service flow graph; (b) "http" service flow graph

The *Implementation2* shows a correct ACL implementation corresponding to the policy.

Implementation2:

Permit http(ZONE_2, PROXY); Deny http(ZONE_1, PROXY);

Permit http(R5, R1); Permit http(R5, R2); Permit http(R1, R3);

Permit http(R2, R3); Permit http(R3, PROXY); Deny http(any, any).

Here, the last rule indicates that all the access paths are treated as default "deny" except those which are explicitly defined prior to the rule. The *"http" service flow graph* corresponding to that implementation is shown in **Fig.4(b)**. In this way, service flow graphs are formed under each network service.

5.2. Fault Analysis under Network Link Failures

The proposed fault analysis problem can be stated as follows: "***To find the maximum number of link failures, the network access model N can tolerate and still satisfy the security policy of the network"***

To solve this problem, we introduce the concept of critical links.

Definition 3: A *critical link* E_{CR} in a service flow graph, G_S is an edge which connects an end point device, DE_i to a network router, DR_j; i.e., $\forall E_{CR}: \{DE_i \times DR_j\} \subseteq IE$.

These edges are critical because of the fact that the failure of any of these edges makes the ACL implementation incorrect unless they are associated with "deny" service access rules. In our analysis, it is considered that there are no critical link failures. However, from the definition of the service flow graph, it is implied that the critical links with "deny" access control rules do not exist in a service flow graph. Hence, the critical links are removed from the service flow graphs prior to the fault analysis procedure.

The fault analysis problem is solved by finding the size of the minimum cut (min-cut) in each service flow graph G_S excluding the *critical edges* which basically represents the fault tolerance value (χ_s) for that service S. Then the minimum of the fault tolerance values associated to different service flow graphs, i.e., $\chi_N = \min\{\chi_s\}$ is calculated which represents the fault tolerance value for the complete ACL implementation under the network access model N. The *fault analysis algorithm* is presented in ***Algorithm 3*.**

***Algorithm 3:* Fault Analysis on Network Access Model**

1. BEGIN
2. FOR each network service S
3. Generate the service flow graph, G_S, from the network access model, N
4. END FOR
5. FOR each service flow graph, G_S
6. Remove the critical links E_{CR} from the graph
7. Find the min-cut of G_S
8. χ_S = Size of the min-cut of G_S
9. END FOR
10. $\chi_N = \min(\chi_S)$
11. Return χ_N
12. END

There are standard algorithms [21] [22] [23] to find minimum cut in a directed graph which uses its close relationship to the maximum flow problem. These algorithms find a minimum cut separating a designated source s from a designated sink node t, and then by varying the sink node, find a minimum cut in a directed graph G as a sequence of at most 2n-2 maximum flow problems. However, in the present approach, the efficient algorithm presented by J. Hao and J. B. Orlin [20] is implemented. It solves the minimum cut problem in a graph in a time $O(|V||E| \log(|V|^2/|E|)$. The algorithm starts with a source node $s \subseteq S$ and solves (n-1) minimum S-t cut problems, where each node in {N-{s}} is considered as a sink node for one of these n-1 problems. After having found the minimum cut for sink node t, the algorithm transfer t to S and select the node with minimum distance label. The algorithm, carefully identifies the nodes disconnected from the sink in a very efficient manner and labelled those nodes as "dormant" nodes. More significantly, the algorithm partitions the node set N into two parts W and D=N-W, where D is the set of "dormant" nodes, and W is a set of nodes that are "awake". The algorithm maintains the property that their never is an arc of the "residual network" directed from a dormant node to a node that is "awake". This algorithm is used to find the *min-cut* for each service flow graph G_S. The complexity and detail analysis of the *min-cut* algorithm [20] is beyond the scope of the paper.

5.3. Analysis & Discussion

The proposed fault analysis approach has been tested on various security implementations in a defined network with specific set of security policies. It is to be noted that the verification module (described in section **4**) capable of providing the correct ACL implementations corresponding to a given policy specification in a network. Table 5 shows the fault analysis on three such test cases of implementations (described in previous section) under the running example network.

The result shows that the service wise fault tolerance values corresponding to the *Implementation1* and *Implementation2* equal to 1 which means the implementations can tolerate at most 1-link failure (except the critical links). So, if their combined implementation scenario is considered (taking both *"ssh"* and *"http" service flow graphs* in account), the fault tolerance value for the network access model N (shown in **Fig.3**) is the minimum of these values which equals to 1. So, the complete network access model can tolerate at most 1-link failure as a whole. In this process, the fault tolerant parameter for any specific ACL implementation can be determined which indicates the maximum number of link failures the implementation can tolerate and still satisfy the security policy of the network. It will help the network administrator to debug the ACL implementations in dynamic changes of network topology due to network link failures.

5. CONCLUSION

In today's complex enterprise network (LAN), there is an increasing requirement of validating the distributed security implementations with the organizational global security policy with dynamic changes in the network topologies. Moreover, it is more challenging to ensure that the security implementation satisfies the global policy although there are certain link failures in the network. In this paper, we have proposed a complete security analysis framework which can verify the security implementations with respect to a defined policy specification in an enterprise LAN and also finds the fault tolerance parameter of a correct ACL implementation (under network link failures). The efficacy of the framework has been aptly demonstrated through examples and case study. The proposed framework will facilitate in debugging of network security implementation efficiently in presence and absence of network link failures and designing conflict free global security policies in a enterprise network.

REFERENCES

[1] Y. Bartal, A. Mayer, K. Nissim and A. Wool. *Firmato: A novel Firewall Management Toolkit.* ACM Transaction on Computer Systems, 22(4): 381-420, November 2004.

[2] E. Al-Shaer and H. Hamed. *Discovery of Policy Anomalies in Distributed Firewalls.* In Proc. of IEEE INFOCOM'04, pp. 2605-2626, 2004.

[3] P. Eronen and J. Zitting. *An Expert System for Analyzing Firewall Rules. Proc. of* 6th Nordic Workshop on Secure IT Systems, pp. 100-107, Denmark, 2001.

[4] E. Al-shaer and H. Hamed. *Firewall Policy Advisor for Anomaly Discovery and Rule Editing.* In 8th IEEE International Symposium on Integrated Network Management, pp. 17-30, Colorado Springs, 2003.

[5] L. Yuan, J. Mai, Z. Su, H. Chen, C. Chuah, and P. Mohapatra. *FIREMAN: A Toolkit for firewall modeling and analysis.* In IEEE Symposium on Security and Privacy, Berkeley, CA, 2006.

[6] *High Level Firewall Language.* Available from http://www.hlfl.org/

[7] B. Zhang, E. Al-Shaer, R. Jagadeesan, J. Riely and C. Pitcher. *Specifications of High level Conflict-Free Firewall Policy Language for Multi-domain Networks.* In 12th ACM Symposium on Access control models and Technologies (SACMAT), pp. 185-194, France, 2007.

[8] *CISCO: Configuring IP access lists.* CISCO white papers 23602 edition, July 2007.

[9] Joshua D. Guttman and Amy L. Herzog. *Rigorous automated network security management.* International Journal of Information Security, vol. 4, pp. 29-48, 2005.

[10] Y. S. Mahajan, Z. Fu, and S. Malik. *Zchaff 2004: An efficient SAT solver.* Lecture notes in Computer Science: Theory and Application of Satisfiability Testing, 3542, pp 360-375, 2005.

[11] C. Zhang, M. Winslet, and C. A. Gunter. *On the safety and efficiency of firewall policy deployment.* In IEEE Symposium on Security and Privacy, pp. 33-50, CA, USA, 2007.

[12] R. W. Ritchey and P. Amann. *Using model checking to analyze network vulnerabilities.* In IEEE Symposium on Security and Privacy, Berkeley, CA, USA, 2000.

[13] P. Matousek, J.Rab, O.Rysavy, M. Sveda. *A Formal model for network-wide security analysis.* 15th IEEE International Conference and workshop on ECBS, pp 171-181, Belfast, Ireland, 2008.

[14] T. Hofmeister, U. Schoning, R. Schuler, and O. Watanabe. *A probabilistic 3-SAT algorithm Further Improved.* 19th Annual Symposium on Theoretical Aspects of Computer Science, Lecture notes in Computer Sc, Vol.2285, Springer-verlag, pp 192-202, 2002.

[15] L. Zhang and S. Malik. *Towards Symmetric treatment of Conflicts and satisfaction in quantified Boolean satisfiability.* In Principles and Practice of Constraint Programming, pp 185-199, 2002.

[16] S. Matsumoto and A. Bouhoula. *Automatic Verification of Firewall Configuration with Respect to Security Policy Requirements.* Proceedings of the International Workshop on Computational Intelligence in Security for Information Systems (CISIS'08), Vol. 53/2009, Springer-Verlag, pp. 185-199, 2009.

[17] *QDIMACS Standard Version 1.1.* Available from http://www.qbflib.org/qdimacs.html/

[18] Y. Yu and S. Malik. *Yquaffle QBF solver.* http://www.princeton.edu/chaff/quaffe.html/

[19] E. Giunchinglia, M. Narrizzano, A. Tacchella. QUBE: A System for deciding quantified boolean formulas *satisfiability.* In International Joint Conference on Automated Reasoning (IJCAR), pp. 364-369, 2001.

[20] J. Hao and J. B. Orlin. *A faster algorithm for finding the minimum cut in a graph.* Proceedings of the 3rd ACM-SIAM Symposium on Discrete Algorithms, pp. 165-174, Orlando, Florida, 1992.

[21] D. Karger and C. Stein. *An $O(n^2)$ algorithm for minimum cuts.* In proceedings of the 25th ACM Symposium on the Theory of Computing, pp 757-765, San Diego, CA, NY, USA, May 1993.

[22] A. V. Goldberg and R. E. Tarjan. *A New Approach to the Maximum Flow Problem.* Journal of the ACM, vol. 35, pp. 921-940, 1988.

[23] R. K. Ahuja, T. L. Magmanti and J. B. Orlin. *Network Flows: Theory, Algorithm and Applications.* Prentice Hall, Englewood Cliffs, N.J, 1992.

[24] P. Bera, Pallab Dasgupta and S. K. Ghosh. *A Verification framework for Analyzing Security Implementations in an Enterprise LAN.* Proceedings of IEEE International Advance Computing Conference (IACC 09), pp. 1008-1015, March 2009.

Authors:

P.Bera is a PhD scholar in School of Information Technology at Indian Institute of Technology, Kharagpur, India. He completed his BE in Computer Science & Engineering from Jadavpur University, Kolkata, India and ME in Computer Science & Engineering from West Bengal University of Technology, Kolkata. His research interest includes network and information system security, distributed systems and formal analysis.

Pallab Dasgupta is a Professor in the department of Computer Science & Engineering at Indian Institute of Technology, Kharagpur, India. He completed his PhD degree from the same institute. He currently leads the Formal Verification research group at IIT Kharagpur, which has ongoing collaborations with several companies, including Intel, Synopsys, General Motors and National Semiconductors. His research interests include Formal Verification, Artificial Intelligence and VLSI. He has over 70 research papers and 2 books in these areas. Dr Dasgupta has been a recipient of the Young Scientist awards from the Indian National Science Academy, Indian National Academy of Engineering, and the Indian Academy of Science. He is a senior member of IEEE.

S.K.Ghosh is an Associate Professor in School of Information Technology at Indian Institute of Technology, Kharagpur, India. He completed his PhD degree from the same institute. His research areas include Network Security, Information System Security, Data Mining and GIS. He is a member of IEEE.