A Query Driven Security Testing Framework for Enterprise Network

Padmalochan Bera[1], Soumya Kanti Ghosh[2]

[1]Infosys Labs, Electronic City
Bangalore 560100, India

[2]School of Information Technology, Indian Institute of Technology Kharagpur
Kharagpur 721302, India

Email: padmalochan_bera@infosys.com, skg@iitkgp.ac.in

Abstract— *Due to extensive use of various network services and web based applications and heterogeneous organizational security requirements; enterprise network configuration is becoming very complex that imposes high operational workload on both regular and experienced administrators. This complexity extensively reduces overall network assurability and usability which in turn make the network vulnerable to various cyber-attacks. Network Access Control Lists (ACLs) is a standard for implementing security configurations in enterprise networks. However, the size and distributed placement of the ACLs in the network impose significant complexity as well as introduce potential scope of security misconfigurations.*

In this paper, we present a query driven security testing framework to assess the correctness and consistency of the access control list (ACL) based security implementations in an enterprise network. It will allow the network administrators to systematically test the ACL configurations with various interactive service access queries. The framework is built on top of a satisfiability analysis (SAT) engine. The efficacy of the framework is evaluated with extensive experimentations on real and synthetic networks.

Index Terms—*Network Security, Access Control Lists Satisfiability Analysis, Security Testing*

1. INTRODUCTION

Today, large amount of business and service offerings by various software enabled product companies is realized through transactions over enterprise networks (enterprise LAN). In this process, the extensive use of network services and web based applications impose high operational workload and complexity that potentially reduces the network assurability and usability. It has been reported that more than 60% of cyber attacks occur due to network misconfigurations and security policy violations.

Typically, an enterprise LAN consists of set of network *zones* corresponding to different units, connected through various interface switches or routers. The organizational security policies are configured in the routers through a set of access control lists (ACLs) in a distributed manner. Each ACL is an ordered set of rules $\{r_1, r_2, .., r_n\}$ which permits/denies service access paths between source and destination zones under different security constraints. In this scenario, the network administrators need to systematically test the distributed ACL configurations to ensure the correct enforcement of security policy. These tests typically cover static network reachability under various constraints (like, access paths routed through or avoiding specific zones, temporal access paths, etc.) and dynamic reachability under link failures. Dynamic reachability is targeted to assess the robustness of the implementation through alternative routing paths.

The complexity of this testing process arises due to the presence of (i) temporal service access constraints; (ii) *intra* and *inter* ACL conflicts; and (iii) inconsistent *hidden access paths* in the ACL implementations. In our earlier works [18] [11], we present verification frameworks for checking satisfaction of network security implementations with a given organizational policy. The framework also checks the presence of ACL conflicts and hidden access paths in the implementations. In [17], we evaluate the implementations under network link failures. However, there is a need of an integrated test framework to systematically evaluate various access paths under complex security constraints. This will help in tracing various security holes in the implementations.

In this paper, we present a formal query driven security testing framework which allows the network administrators to evaluate the distributed ACL implementations with various service access queries. The queries are specified using a high level specification language (QSL). The enterprise security policy can be also tested as a combination of [conjunction ($\wedge$) or disjunction ($\vee$)] of multiple queries. The framework models the network topology and the ACL configuration as a graph based network access model that captures the ACL rules distributed across the network interfaces. For the purpose of query based testing, the framework reduces the network model and the QSL queries into set of Boolean clauses and checks the satisfiability of the query in the model. Apart from general reachability testing, the framework is also capable of checking complex queries like,

- Existence of service access paths during specified time.
- Existence of service access paths passing through or avoiding some network zones.
- Existence of access paths under network link failures.

1.1 State of the Art

Existing literatures on network security analysis primarily concentrate on inconsistency and redundancy checks but most

of the works are not formally verified. The tools, Firmato [1] and Lumeta [2] can specify an abstract network access control policy and firewall rules that satisfy the policy but lacks in incorporating temporal and hidden access constraints. Al-Shaer et al. [3] worked on Firewall Policy Advisor. But, these tools can handle simple set of policy constraints.

There are few works on network security analysis using formal methods. FIREMAN Toolkit [4] detect inconsistencies and redundancies in network of firewalls. The tool formulates all possible requests and incorporates model checking 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 using an efficient BDD representation. In another work, Matsumoto and Bouhoula [5] present a SAT based approach for verifying firewall configurations with respect to security policy requirements. Recently, Liu et al. proposed frameworks for verifying [7] and querying [8] network firewalls. They have used Firewall Decision Diagrams (FDDs) for representing firewall rules and deployed algorithms for verifying and querying the packets to be accepted and denied. However, these frameworks do not address verification of distributed firewall configurations with enterprise-wide security policy under complex access constraints. An enterprise policy may require restricting access paths during a specified time or through some vulnerable zones. The existing firewall implementations may avoid such paths due to lack of knowledge on the security need. The dependencies between firewall rules may implicitly introduce inconsistent *hidden access paths*. The implementations may violate the policy under link failures. However, none of the earlier works address these problems.

Matousek et al. [10] proposed a formal model for network wide security analysis. They model the network topology with changing link states and deploy bounded model checking of security properties using SAT-based decision procedure. However, this work is unable to ensure whether a security implementation is fault tolerant under arbitrary link failures. Also, they enumerate all possible link state combinations in a network which may lead to state explosion problem for large networks. *Quarnet* [9] is another tool for querying and verifying network reachability. However, it does not consider temporal service constraints and the hidden access paths in their analysis. It does not check the reachability of a node through/avoiding specific zone. Moreover, a framework for analyzing the correctness of the security implementations under topology changes has not been addressed earlier.

Rest of the section is organized as follows. In section 2, the architecture of the proposed security testing framework has been described. The modeling of network topology and ACL rules has been described in Section 2.1. The query specification language (QSL) and modeling of various service access queries have been described in Section 3. Section 4 describes the Security Testing of the network model using different access queries. The experimental results and evaluation of the framework have been presented in Section 5.

2. PROPOSED SECURITY TESTING FRAMEWORK

The proposed framework primarily concentrates on testing security implementations through various service access queries with complex constraints. The framework formally models the network topology and the ACL configurations as a graph based network access model. Then, it incorporates the ACL conflict analyzer and hidden access path analyzer to resolve the ACL rule conflicts and to generate a refined access model respectively. Then, it allows users to represent the access queries through a high level specification language, QSL. Queries can be of two types: (1) Static Topology based queries; (2) Fault based queries. Static queries are evaluated using Boolean satisfiability analysis (SAT), whereas the fault based queries are evaluated using graph mining procedures. In both the cases, the framework reports the access path traces. The framework (refer Figure 1) consists of three modules:

- Network Access Configuration Module
- Query Specification Module
- Query Testing Module

Network access module parses the topology and the distributed ACL implementation to extract a graph based network access model. It is defined as a 3-tuple $NG<N, I, F>$, where, N represents the set of network devices; either a network zone or a network routers. I represent the set of interfaces between the network devices and F represents the set of access control lists (ACLs) associated to the interfaces. An ACL consists of a set of rules where, each rule can be interpreted as $(P, action)$. Here, P is a predicate representing the packets (service, source and destination IP) those are matched by the rule and *action* represents the decision on the matched packets. Packets not matched by the current rule are forwarded to the next rule until a match is found or the end of the ACL is reached. At the end of an ACL, the default action is applied. Intuitively, the network access model completely defines the service access paths across the network as set of *"permit"* and *"deny"* ACL rules. Then, this module represents the individual and distributed ACLs into Boolean models. In this process, the access routes and access route ACLs are formally modeled along all possible routes from a source S to a destination D. Then, the *conflict analyzer* incorporates procedures for detecting and resolving ACL rule conflicts in the model. In result, the distributed conflict-free network model is derived as a single Boolean function, $M_{acl}(V:service, S:source, D:dest, T:time)$, that states that the service V , is allowed between the source S and destination D, at time, T. In M_{acl}, we consider only the "permit" ACL rules as the "deny' rules are default and captured by $\neg M_{acl}$. Finally the hidden access rules are applied on M_{acl} to derive a refined network access model, M_{acl}^{fin}. It formally models the hidden rules through a set of quantified Boolean formulas and incorporates these clauses in the network access model.

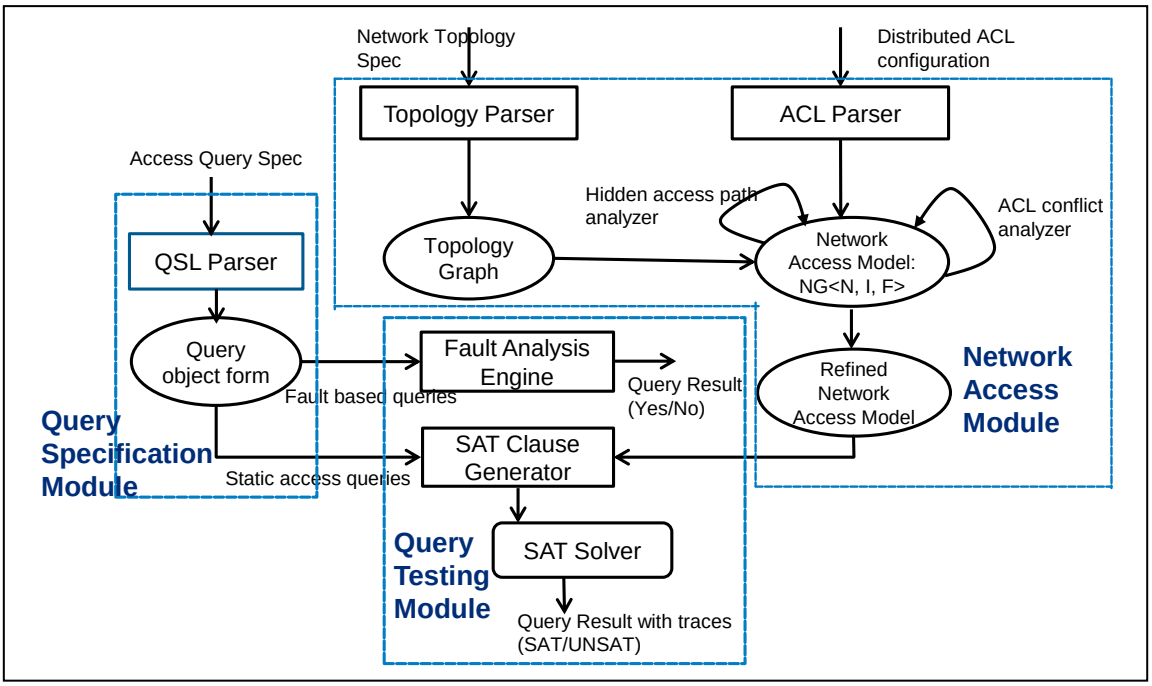

Figure 1: Network Security Testing Framework

Query specification module represents access queries through a high-level query language called, QSL. The queries can be classified into two categories: (i) Static Topology based queries; and (ii) Fault based queries. Static queries include finding direct service access paths, access paths under some time constraints, access paths including/excluding some zones, etc. Whereas, fault based queries include finding access paths under network link failures with various constraints.

The query testing module evaluates the query with respect to the refined network access model, M_{acl}^{fin} and reports the result. The static queries are reduced into Boolean clauses whose satisfaction is checked with the model M_{acl}^{fin} using a QBF SAT solver *quaffle* [12]. In case of SAT result, the framework reports the presence of access path along with the trace in the network. On the other hand, fault based queries are processed through fault analysis engine that uses graph mining procedure. Fault Analysis engine first takes the refined network access model M_{acl}^{fin} as input and derives a *service flow graph* (G_V) under each network service, V considering the ACL rules associated to the network interfaces (edges of the graph). Then, it finds the *min-cut*, τ_V (minimum edge cut), or each service flow graph, G_V. Finally, it computes the smallest *min-cut* of all service flow graphs representing the *fault tolerance* value of the network access model. Then, the engine evaluates the fault based query by applying graph mining procedure on the service flow graph G_V and checking whether the query satisfies the *fault tolerance* value if required.

2.1 Modelling Distributed ACLs and Conflict Analysis

The presented framework parses the network topology and the distributed ACL implementation to extract a graph based formal network access model, *NG<N, I, F>*. Once the network access model has been derived, the implementation module represents the individual and distributed ACLs into Boolean clauses. Although ACLs are configured independently, it is required to model the combined effect of the distributed ACLs in the network to verify the end-to-end security behavior. Our framework models individual ACL F_I associated to the interface I into two Boolean functions:

$$FP_I = \vee_{r_i \in F_I} \{r_i \mid r_i.action = "permit"\}$$

$$FD_I = \vee_{r_i \in F_I} \{r_i \mid r_i.action = "deny"\}.$$

Then, distributed ACLs along all possible access routes between a source S and destination D has been modeled. In this process, the notions of access routes and access route ACLs have been introduced.

Definition 1 [Access Route]: An *Access Route* $AR_i^{(S,D)}$ is defined as a sequence of routers (R_1, R_2, …., R_n) from source S to destination D in the network where, $< R_i, R_{i+1} > \in I$ and D is reachable from S through S, R_1, R_2, …, D.

Definition 2 [Access Route ACL]: An *Access Route ACL*, $ARCL^{(S,D)}$ between a source S and destination D is a combined model of distributed ACLs along all possible access routes, $AR_1^{(S,D)}$, $AR_2^{(S,D)}$, …, $AR_n^{(S,D)}$. It is represented using following two Boolean functions

$$ARCL^{(S,D)}(FP) = FP_{AR_1^{(S,D)}} \vee FP_{AR_2^{(S,D)}} \vee .. \vee FP_{AR_n^{(S,D)}}$$

$$ARCL^{(S,D)}(FD) = FD_{AR_1^{(S,D)}} \vee FD_{AR_2^{(S,D)}} \vee .. \vee FD_{AR_n^{(S,D)}}$$

here, $FP_{AR_1^{(S,D)}} = \Lambda_{I \in AR_i} FP_I$ and $FD_{AR_1^{(S,D)}} = V_{I \in AR_i} FD_I$.

Then, this module incorporates *conflict analyzer* for detecting *inconsistencies* and *redundancies* in the model. A pair of ACL rules is inconsistent, if one rule allows some access paths which are denied by the other. On the other hand, redundancy indicates repetition of rules. Conflicts may occur in two levels: (1) *intra* ACL - covers conflicts between rules in individual ACL and (2) *inter* ACL - covers conflicts between rules in multiple ACLs. These are modeled as follows:

Intra ACL Redundancy:

Case 1: $FP_I \wedge \{\vee_i r_i \mid r_i.action = "permit"\}$

Case2: $FD_I \wedge \{\vee_i r_i \mid r_i.action = "deny"\}$

Intra ACL Inconsistency: $FP_I \wedge FD_I$ for each ACL F_I

Inter ACL Inconsistency: $FP_{AR_i^{(S,D)}} \wedge FD_{AR_i^{(S,D)}}$ for each access route, $AR_i^{(S,D)}$.

Here, *intra* ACL redundancies are modeled as conjunction between the Boolean function FP_I/FD_I and the *"permit"/"deny"* ACL rules under individual ACL, F_I. On the other hand, *intra* ACL inconsistencies are modeled as conjunction between the functions FP_I and FD_I. The *inter* ACL inconsistencies are modeled as conjunction between the Boolean functions $FP_{AR_1^{(S,D)}}$ and $FD_{AR_1^{(S,D)}}$ for each access route, $AR_i^{(S,D)}$ considering the distributed ACLs along that route. The procedure detects conflicts using Boolean satisfiability analysis. The conflicts are reported as "error(s)" along with satisfiable instances. The network administrator reconfigures the ACLs accordingly. The conflict-free ACLs are formally reduced to a Boolean function: $M_{acl} = V_{S \in N, D \in N} ARCL^{(S,D)}(FP)$. In M_{acl}, only the "permit" ACL rules are considered as the "deny' rules are treated as default and captured by the function, $\neg M_{acl}$.

2.2 HIDDEN ACCESS PATH ANALYSIS

Hidden access paths may exist in a network due to transitive access relationships between various network services. These access paths can be formally represented as follows:

$\forall X \forall Z((\exists Y, serv_1(X,Y) \wedge serv_2(Y,Z) \Rightarrow serv_2(X,Z)$..[eq1]

$\forall X \forall Z((\exists Y \exists T, serv_1(X,Y)[T] \wedge serv_2(Y,Z)[T]$

$\Rightarrow serv_2(X,Z)[T].$[eq2]

Here, $X, Y, Z \in N$ represent network *zones* and T represents *time-constraint*. The $serv_1$ and $serv_2$ represent the network services which are transitively dependent on each other. Equation 1 represents *static hidden access paths*, whereas, equation 2 represents the *temporal hidden access paths* at time *T*. The combination of network services that can create hidden access paths needs to be known for testing. This requires the domain knowledge on network service dependencies. It is also possible to represent the multi-hop access paths using hidden access model. In that case, the related services under this dependency are same, i.e., *serv*1 = *serv*2. Our framework reduces the *hidden access rules* into a set of quantified Boolean formulas (QBF) and incorporated in the network access model. However, our framework also allows users to make hidden access path analyzer as optional component. This is controlled by a flag *H*. If *H* = 1, then hidden access path analysis is performed on the ACL implementation model. Next section describes the modeling of service access queries.

3. QUERY SPECIFICATION MODULE

A query specification language (QSL) has been presented for modeling the queries. Static topology based access queries evaluate the network reachability with specified constraints under the static network topology, whereas, the fault based access queries reason about the presence service access paths under dynamic topology (i.e., link failures).

3.1 QSL SYNTAX

The QSL queries follow the following syntax:

FIND PATH
FROM Z_S TO Z_d
WHERE (Service $\in V$)
[IN TIME $\in T$] [EXCLUDE Z_P] [INCLUDE Z_Q]
[HIDDEN H] [MAX FAULT f]

A QSL query typically finds access paths between a source Z_S to destination Z_d depending on different constraints. Here, the constraints specified within [ ] are optional. In the WHERE clause the (*Service* $\in V$) indicates the network service which is again combination of protocol (protocol type) and port(service port number) in the destination zone. In the FROM and TO clauses *Zs* and *Zd* indicates the source and destination zone respectively. The optional time constraint is represented as TIME $\in T$ where *T* is any valid time period. In our model, a valid time period can be combination of *days in a week* and *hours and minutes in a day*. The [INCLUDE Z_Q] and [EXCLUDE Z_P] clauses indicate the set of network zones to be included or excluded for evaluating the access queries. [HIDDEN *H*] clause indicates whether "hidden access rules" to be considered during the evaluation of access queries.

3.2 QSL SEMANTICS

The proposed framework takes the QSL queries as input and evaluates it with respect to the ACL implementation. It results "yes" or "no" depending on the query evaluation. The tool reports "yes", if the access query is satisfiable with respect to the ACL implementation along with satisfiable instance(s). The semantics of QSL queries has been described with couple of example queries.

Example QUERY1:
FIND PATH
FROM ZONE_1 TO ZONE_2
WHERE (PROTOCOL $\in$ TCP) $\wedge$ (PORT $\in$ 22)
IN TIME $\in$ SUNDAY <07:00-23:59> EXCLUDE ADMIN
HIDDEN H = 1

QUERY1 evaluates to "yes", if the "ssh" access path exists from any source in *ZONE*_1 to any source in *ZONE*_2 during 07:00 − 23:59 in *Sunday* not going through *ADMIN*. The significance of the query is that the administrator wish to allow "ssh" access paths from *ZONE*_1 to *ZONE*_2 during 07:00 − 23:59 in *Sunday* only through *ADMIN* for keeping track of the packets. The inclusions of hidden access rules signify to check the availability of such indirect access paths.

Example QUERY2:
FIND PATH
FROM ZONE_2 TO PROXY
WHERE (PROTOCOL ∈ TCP) ∧ (PORT ∈ 80)
HIDDEN H = 1
MAX FAULT = 2

QUERY2 evaluates to "yes" along with the access path, if the "http" access path exists from any source in *ZONE*_2 when two (2) network links (arbitrary) are down. This query is evaluated by the fault analysis module.

Apart from the above type of queries, the framework can also handle other queries following the QSL syntax. In addition, it can evaluate sequence of queries as a group; say a complete security policy specification. In that case, the framework models only *"permit"* policy rules and the *"deny"* rules are treated as default. In the next section, we describe how the queries are tested with respect to the network access model.

4. SECURITY QUERY TESTING MODULE

The framework evaluates the static topology based access queries using Boolean satisfiability (SAT) analysis; whereas, fault based queries are evaluated using graph mining based testing procedures. The SAT based approach reduces the testing problem into a Boolean formula *F* and checks its satisfiability. SAT based approach acquired significant attention due to tremendous time tradeoffs of modern SAT [6] and QBF-SAT solvers [13] [14]. Our framework reduces the refined network access model M_{acl}^{fin} and the static QSL queries into Boolean functions under the same formalism. Then, the satisfiability of the access query is checked with the model using *quaffle* QBF solver. The next section describes the Boolean reduction of the network access model M_{acl}^{fin}.

4.1 BOOLEAN REDUCTION OF NETWORK ACCESS MODEL

Boolean reduction of the network model requires functional mapping of the rule components into Boolean clauses. The rule components include *service (protocol, port number), source IP, destination IP, time-constraint* and *action*. A network zone can be specified as single IP address or range of IP addresses. So, we model the *source* and *destination* zones with 32 Boolean variables each, namely, (s_0, s_1,..., s_{31}) and (d_0, d_1,..., d_{31}) respectively. A range of IP addresses can be translated using disjunction (∨) of IP blocks. Address ranges with masks are reduced by bit-wise and-ing the masks with the base addresses. Similarly, protocol type and service port numbers are mapped into 5 and 16 Boolean variables, namely, (p_0, p_1, .., p_4) and (i_0, i_1, .., i_4) considering 32 different protocols and 65356 different service ports respectively.

Table 1. Boolean Reduction of the ACL rules

Protocol(P): FP(p_0,p_1,..,p_4) Service_Port(I):FI(i_0,i_1,…,i_{15}) Src_zone_IP(SIP):FS(s_0,s_1,..,s_{31}) Dst_zone_IP(DIP):FD(d_0,d_1,..,d_{31}) Time(T):FT(dt_0,dt_1,dt_2,th_0,..,th_4,tm_0,..,tm_5) Action(g):A(g)
Procedure Reduce_Rule() **Input**: An ACL rule r_i **Output**: Boolean reduction of the rule r_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. $T_i \Leftrightarrow FT_i(dt_0, dt_1, dt_2, th_0,.., th_4, tm_0,.., tm_5) \wedge$ 8. $r_i \Leftrightarrow (Serv_i \wedge SIP_i \wedge DIP_i \wedge T_i) \wedge$ 9. return r_i 10. END
Example: Boolean Reduction of a rule **Rule(r_1):** deny http(ZONE_1, PROXY) $Serv_1$ = [http:(TCP(1),80)] $\Leftrightarrow (\neg p_0 \wedge p_1) \wedge (\neg i_0 \wedge i_1 \wedge \neg i_2 \wedge i_3 \wedge \neg i_4 \wedge .. \wedge \neg i_7)$ SIP_1=[ZONE_1:10.0.0.0/10] $\Leftrightarrow (\neg s_0 \wedge .. \wedge \neg s_3 \wedge s_4 \wedge \neg s_5 \wedge s_6 \wedge \neg s_7 \wedge \neg s_8 \wedge \neg s_9)$ DIP_1=[PROXY:172.16.0.22] $\Leftrightarrow (d_0 \wedge \neg d_1 \wedge d_2 \wedge \neg d_3 \wedge d_4 \wedge d_5 \wedge \neg d_6 \wedge \neg d_7 \wedge \neg d_8 \wedge .. \wedge d_{11} \wedge \neg d_{12} \wedge .. \wedge \neg d_{26} \wedge d_{27} \wedge \neg d_{28} \wedge d_{29} \wedge d_{30} \wedge \neg d_{31})$ $r_1 \Leftrightarrow Serv_1 \wedge SIP_1 \wedge DIP_1$

A *network service* is modeled as conjunction (∧) of a protocol and a service port number. The time constraints are modeled as disjunction of its valid periods. Each time period can contain *day of week*, *hours* and *minutes*. These components are mapped into a set of Boolean variables, namely, (dt_0, dt_1, dt_2), (th_0, th_1, .., th_4) and (tm_0, tm_1,.., tm_5) respectively. Here, we have considered the granularity of time in minute. The functional mapping of rule components into Boolean variables is depicted in Table 1. After the reduction of rule components, the framework models the intra and inter ACL conflicts into Boolean clauses. These conflicts are detected by checking the satisfaction of the conflict model using *quaffle* solver. Once these conflicts are resolved, the tool refines the network model incorporating the Boolean clauses for hidden access rules. The refined network model is represented by the function M_{acl}^{fin}.

4.2 STATIC QSL QUERIES PROCESSING

The static query components are functionally reduced into a set of Boolean variables under the same formalism as described in the last section. Each query is represented as conjunction (∧) of the components. The "EXCLUDE" query incorporates an extra clause with negation of the Boolean

variable associated to the specified zone. The "INCLUDE" query incorporates a clause with the Boolean variable associated to the zone. The framework also allows users to analyze the queries with hidden access path analyzer as optional. If H = 1, the query is analyzed with the refined ACL implementation model, M_{acl}^{fin} including hidden access rules. If H = 0, the hidden access rules are not considered. The SAT formulation of *QUERY1* is presented as follows:

$$[(M_{acl}^{fin} \Rightarrow ((Protocol = TCP) \wedge (Port = 22) \wedge$$
$$(SIP \in ZONE\_1) \wedge (DIP \in ZONE\_2) \wedge$$
$$(T \in Sunday < 07:00 - 23:59 >))] \wedge$$
$$[(M_{acl}^{fin} \Rightarrow \neg((Protocol = TCP) \wedge (Port = 22) \wedge$$
$$(SIP \in ZONE\_1) \wedge (DIP \in ADMIN) \wedge$$
$$(T \in Sunday < 07:00 - 23:59 >))]$$

The queries which do not involve hidden access rules are checked with the conflict-free implementation model, M_{acl}. A query results "TRUE" if the corresponding Boolean formula is satisfiable and the SAT instance indicates the access trace.

4.3 Fault Based QSL Query Processing

In our earlier work [18], the fault analysis of network security implementations has been described. It essentially finds the minimum edge cut (*min-cut*) of different service flow graphs under the network access model (*NG*) and then calculates the fault tolerance value of the model that represents maximum number of link failures it can tolerate. A service flow graph G_V includes only the edges of *NG* through which corresponding service accesses are allowed. The fault analysis module finds the size of the minimum edge cut (*min-cut*) of each service flow graph G_V which represents the fault tolerance value (τ_V) under the service *V*. Then, the fault tolerance value of the network access model is calculated as $\tau_{NG} = min\{\tau_V\} - 1$. Here, τ_{NG} represents the maximum number of link failure the network model can tolerate to maintain its consistency. Our framework uses this procedure for analyzing the fault based access queries. For example, the evaluation of *QUERY2* (refer Section 3) has been described as follows:

Evaluation of QUERY2:
1. Derive the *http⟨TCP;* 80*⟩* service flow graph, G_{http}
2. Find the maximum fault tolerance value, τ_{NG} of network access model, *NG*
3. IF (*http*(*ZONE* 2*; PROXY*) ∈ *Ghttp*)∧*_NG* ≥ 2 THEN
4. Return *TRUE*
5. ELSE Return *FALSE*
6. END IF

The input to the fault analysis module is the query path, i.e., *http⟨TCP;* 80*⟩*(*S;D*) and the maximum fault tolerance value 2 specified in the query. The analysis module first derives the corresponding service flow graph, *Ghttp* and then checks whether the access path belongs to *Ghttp* and $\tau_{NG} \geq 2$. The query is evaluated to *True* if the conditions are satisfied, otherwise evaluated to *False*. For example, the *QUERY2* is evaluated in a typical enterprise network shown in Figure 2 and its ACL configuration shown in Table 2.

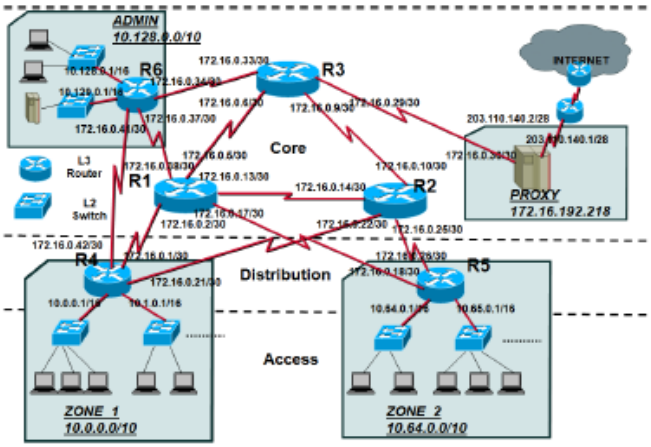

Figure 2: Test-Net: A Typical Enterprise Network

Table 2. Test-ACL: An ACL Implementation under Test-Net

Router R4, Interfaces 172.16.0.1, 172.16.0.21, 172.16.0.42:
```
accesslist 1 deny TCP 10.0.0.0 255.192.0.0 host 172.16.192.218 eq http;
accesslist 1 deny TCP 10.0.0.0 255.192.0.0 10.64.0.0 255.192.0.0 eq ssh
accesslist 1 deny TCP 10.0.0.0 255.192.0.0 10.128.0.0 255.192.0.0 eq ssh
```
Router R5, Interface 172.16.0.18,172.16.0.26:
```
accesslist 2 permit TCP 10.64.0.0 255.192.0.0 host 172.16.192.218 eq http
accesslist 2 permit TCP Any 10.64.0.0 255.192.0.0 eq ssh
accesslist 2 deny TCP 10.64.0.0 255.192.0.0 10.0.0.0 255.255.0.0 eq ssh
accesslist 2 deny TCP 10.64.0.0 255.192.0.0 10.128.0.0 255.192.0.0 eq ssh
```
Router R6, Interface 172.16.0.34,172.16.0.37,172.16.0.41:
```
accesslist 3 permit TCP 10.128.0.0 255.192.0.0 host 172.16.192.218 eq http
acceslist 3 permit TCP 10.128.0.0 255.192.0.0 10.64.0.0 255.192.0.0 eq ssh
```
Router R1, R3 and R4, All Interfaces:
```
accesslist 5 permit TCP any any eq ssh
accesslist 5 permit TCP any host 172.16.192.218 eq http
```

Figure 3 shows the corresponding *http* service flow graph.

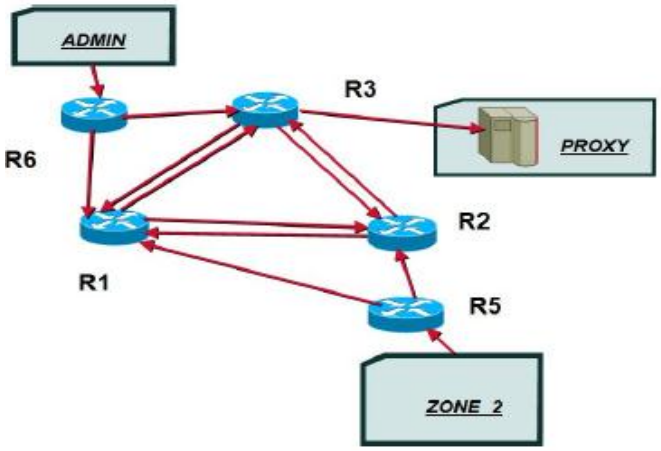

Figure 3: http service flow graph for Test-ACL

The maximum number of link failure failures allowed (τ_{NG}) for the ACL implementation (refer Table 2) is 1. Thus, the *QUERY2* evaluates to *False*. If the links *I⟨*R5,R1*⟩* and *I⟨*R5,R2*⟩* are down, then the "http" access path does not exists in the network model. However, if the query is modified with the clause MAX FAULT 1, then it will be evaluated to *True*. In this way, our framework evaluates the fault based queries.

5. RESULT AND EVALUATION

The various modules of our security testing framework have been implemented in C programming language under Linux environment. Lex-Yacc is used to develope parsers for topology, ACL configurations, and QSL queries. The ACL conflict analyzer and SAT based query processing modules use efficient *quaffle* QBF solver [15]. The service flow graph construction, *min-cut* calculation and graph traversals algorithms are also implemented in C. In this section, we evaluate the performance of our framework in terms of time requirements. The framework has been tested under 200 different ACL configurations in more than 50 different test networks and 10 millions of ACL rules. We have analyzed the impact of different parameters on query processing.

Impact of Network and ACL Size on Model Generation: The network size is the overall nodes including: routers and network zones. We used the network that ranges from 25 to 1000 nodes, with total configuration size (ACL rules) up to 10 million per network. Figure 4(a) shows the impact of network and ACL size on generating the access model. The model generation time is almost linearly dependent on both network and ACL size. It is observed that this time lies within 8 seconds for 1000 nodes and 1000 ACL rules/node (i.e., total 10 million ACL rules). Thus, for large networks, our framework generates the model in reasonably good time.

Impact of Network Size on ACL Conflict Analysis: SAT reduction time of conflict analysis procedure is linearly dependent on the network and ACL size. On, the other hand, the *quaffle* run-time (for conflict detection) is dependent on the total number of Boolean variables (almost remains constant with varying ACL size) that lies within a second for 1 million of ACL rules. Thus, the overall conflict analysis time is linearly dependent on the network and ACL size. Figure 4(b) shows that the analysis time lies within 5 seconds for 1000 network nodes and a total of 10 million ACL rules.

Impact of ACL Size on Static Query Testing: Static query processing time has two components, (1) Reduction time of the model and query; and (2) *quaffle* runtime. Figure 4(c) shows that the reduction time is linearly dependent on the ACL size. However, the *quaffle* run-time depends on the total number of Boolean variables (almost remains constant with varying ACL size and additional variables incurred for hidden rules). The static query processing time lies within 7 seconds for a total of 1 million ACL rules. It is observed that, for a particular network model, this time almost remains same for different variations of queries (e.g., direct reachability, reachability excluding/including a network zone, etc).

Impact of Network Size and ACL rules on Fault based Query Testing: Fault based query processing time consists of three components (i) service flow graph construction time, (ii) *min-cut* finding time and (iii) graph traversal time. The impact of network and ACL size on these components has been shown in Figure 4(d). The service flow graph construction time is almost linearly dependent on the number of active network nodes under each service. On the other hand, we use efficient algorithm presented in [15] for finding *min-cut* of the service flow graphs. It works in $O(|N^V||I^V| \log(|N^V|^2/|I^V|)$ time, where $|N^V|$ and $|I^V|$ represent the number of nodes and edges in the graphs respectively. It is to be noted that, under a given network access model, service flow graph construction and *min-cut* finding time affect only once on different query processing time. The service flow graph traversal algorithm works in $O(|N^V|+|I^V|)$ time. Figure 4(d) shows, the network size has a considerable impact on min-cut finding time. Figure 4(e) shows the overall fault based query processing time with varying network and ACL size. Here, we vary the network size between 100 and 1000 (with 100-1000 rules per node). For 1000 nodes and 10 million ACL rules, the average fault based query processing time lies within 35 seconds.

Evaluation of static queries with Query complexity: We have analyzed SAT based query processing time with varying query complexity. We have tested 4 different type of queries, namely, (1) *basic reachability,* (2) *temporal reachability* (queries with time constraints); (3) *hidden access path finding,* and (4) *EXCLUDE/INCLUDE query* (finding path routed through/avoiding a specific network zone). Figure 4(f) shows the time required for evaluating these queries under a network of 400 nodes and with varying ACL size between 40000 and 32 million (100 − 800 ACL rules/node). It shows, that *hidden access path query* time requires more time than others. This is incurred from the QBF reduction time of hidden access rules.

From the above analysis, it has been observed that, the fault based query processing requires significantly more time than static queries. Thus, our framework uses SAT based procedure in most of the analysis. However, graph mining based decision can be used for in-depth query processing on various network constraints like, faults, network congestion and policy based routing. In the same line of work, *FIREMAN* [4] and *ConfigChecker* [16] [9] tools use BDD based model for analyzing firewall configurations and various types of network reachability. However, none of these tools consider temporal access rules and analyzes the effect of link failures in the configuration. Moreover, the analysis of hidden access paths has not been addressed earlier which plays a significant role in checking the correctness of the ACL configurations.

6. CONCLUSION

In today's complex enterprise networks, there is an increasing need of validating the security configurations to enforce the organizational policy. It is becoming more difficult due to complex security needs of the organizations with various access constraints as well as dynamic changes in network topology. In this paper a formal security testing framework has been presented which takes the network topology and the distributed ACL implementations as input; detects the *intra* and *inter* ACL conflicts and allows the users to evaluate various service access queries under different constraints. The framework has been tested with various access queries and distributed ACL implementations under different test networks and experimental results are reported. The framework is useful to the network administrators for debugging the ACL implementations with various constrained access queries.

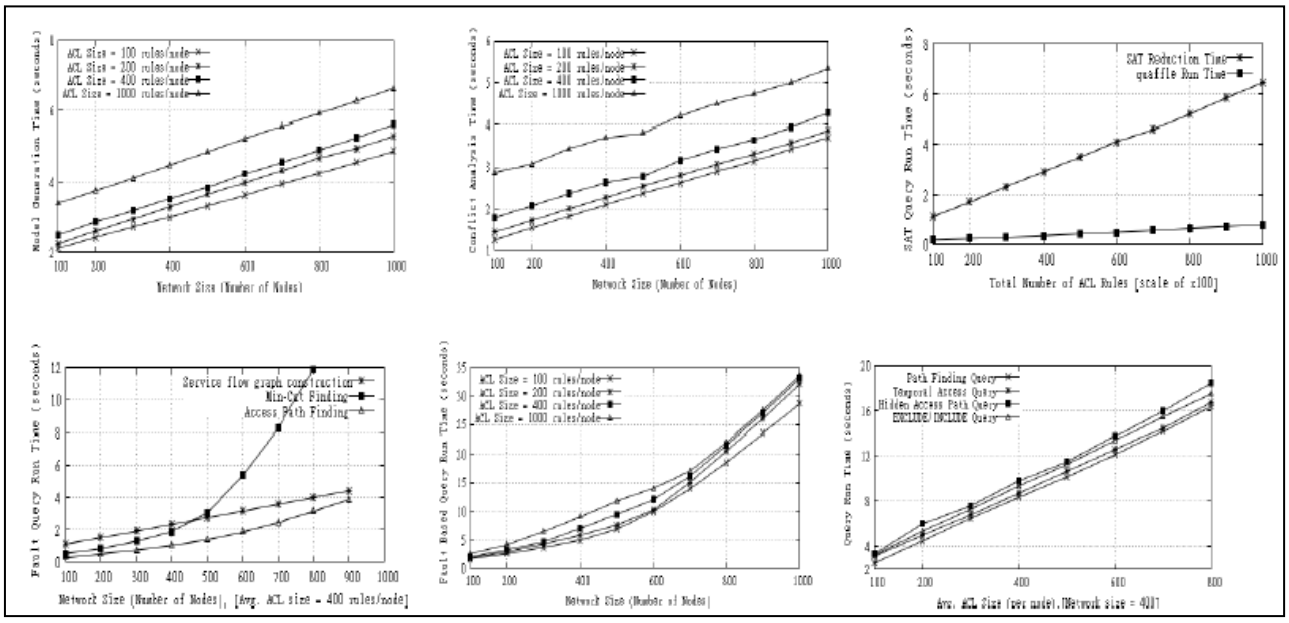

Figure 4: (a) Time to Generate Network Access Model wrt. Network and ACL size; (b) ACL Conflict Analysis Time wrt. Network and ACL size; (c) SAT based Query processing time wrt. ACL size; (d) Fault based Query Processing Time Components wrt. Network and ACL size; (e) Fault based Query Processing time wrt. Network and ACL size; (f) Variations in SAT based Query Processing time wrt. Query Complexityhttp service flow graph for Test-ACL

REFERENCES

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

[2] E. S. Al-Shaer and H. H. Hamed. *Discovery of Policy Anomaliesin Distributed Firewalls.* In IEEE INFOCOM'04, pp. 2605-2626, Hong Kong, China, March 2004.

[3] E. S. Al-Shaer and H. H. Hamed. *Firewall Policy Advisor for Anomaly Discovery and Rule Editing.* In IFIP/IEEE Symposium on Integrated Network Management, pp. 17-30, Colorado Springs, 2003.

[4] L. Yuan, J. Mai, Z. Su, H. Chen, C. Chuah, and P. Mohapatra. *FIREMAN: A Toolkit for Firewall Modeling and Analysis.* In 27th IEEE Symposium on Security and Privacy, CA, USA, May 2006.

[5] S. Matsumoto and A. Bouhoula. *Automatic Verification of Firewall Configuration with Respect to Security Policy Requirements.* In CISIS'08, pp. 123-130, Barcelona, Spain, October 2008.

[6] Y. S. Mahajan, Z. Fu, and S. Malik. *Zchaff 2004: An efficient SAT solver.* In Proceedings of 8*th* International Conference on Theory and Application of Satisfiability Testing, LNCS 3542, pp. 360-375, Scotland, June 2005.

[7] M G Gouda, A X Liu and M Jafry. *Verification of Distributed Firewalls.* In IEEE GLOBECOM, pp. 1-5, New Orleans, USA, November, 2008.

[8] A X Liu and M G Gouda. *Firewall Policy Queries.* In IEEE Transactions on Parallel and Distributed Systems, vol 20(6), pp. 766-777, 2009.

[9] A. R. Khakpour and A. X. Liu *Quarnet: A Tool for Quantifying and Verifying Network Reachability.* In Conference on Computer & Communication Security, pp. 1-13, Chicago, USA, 2009.

[10] P. Matousek, J. Rab, O. Rysavy, M. Sveda. *A Formal model for Network-wide Security Analysis.* In 15th IEEE International Conference and Workshop on ECBS, Belfast, Ireland, 2008.

[11] P. Bera, S. K. Ghosh and Pallab Dasgupta *Policy based Security Analysis in Enterprise Networks -A formal approach* IEEE Transactions on Network and Service Management, 7(4), pp. 231-243, 2010.

[12] Y. Yu and S. Malik. *Yquaffle QBF solver.* Available from http://www.princeton.edu/chaff/quaffle.html. Accessed on March 2009.

[13] 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.

[14] L. Zhang and S. Malik. *Towards Symmetric treatment of Conflicts and satisfaction in quantified Boolean satisfiability.* In 8*th* International Conference on Principles and Practice of Constraint Programming (CP 2002), LNCS 2470, pp. 200-215, NY, USA, 2002.

[15] J. Hao and J. B. Orlin. *A Faster Algorithm for finding the minimum cut in a graph* In 3*rd* ACM-SIAM Symposium on Discrete Algorithms, pp. 165-174, Orlando, Florida, 1992.

[16] E. Al-Shaer, W. Marrero, A. El-Atawy and K. ElBadani. *Network Configuration in a box: Towards end-to-end verification of network reachability and security* In 17th IEEE International Conference on Network Protocols (ICNP2009), pp. 123-132, NJ, USA, October 2009.

[17] P. Bera, S. K. Ghosh and Pallab Dasgupta. *FIntegrated Security Analysis Framework for an Enterprise Network – A Formal approach.* In IET Information Security Journal, vol. 4(4), pp. 283-300, December 2010.

[18] P. Bera, S. K. Ghosh and Pallab Dasgupta. *Formal Verification of Security Policy Implementations in Enterprise Networks* In Proceedings of 5th International Conference of Information System Security (ICISS), vol. 5905 (2009), pp. 117-131, India, December 2009.