Improving security in SCADA systems through firewall policy analysis

Ondrej Rysavy Jaroslav Rab Miroslav Sveda

Faculty of Information Technology
Brno University of Technology,
612 66 Brno, Czech Republic
e-mail:{rysavy, rabj, sveda}@fit.vutbr.cz

***Abstract*—Modern SCADA networks are connected to both the companys enterprise network and the Internet. Because these industrial systems often control critical processes the cyber-security requirements become a priority for their design.**

This paper deals with the network security in SCADA environment implemented by firewall devices. We proposed a method for verification of firewall configurations against a security policy to detect and reveal potential holes in implemented rule sets. We present a straightforward verification method based on representation of a firewall configuration as a set of logical formulas suitable for automated analysis using SAT/SMT tools. We demonstrate how such configuration can be analyzed for security policy violation that can be inferred from a security policy specification of an industrial automation system.

I. Introduction

SCADA (Supervisory Control and Data Acquisition) systems are commonly deployed to continuously monitor and control industrial processes to assure proper functioning, by automating telemetry and data acquisition. Historically, SCADA systems were believed to be secure because they were isolated networks: an operator console, or human-machine interface (HMI), connected to remote terminal units (RTUs) and programmable logic controllers (PLCs) through a proprietary purpose-specific protocol. Yielding to market pressure, that demands industries to operate with low costs and high efficiency, these systems are becoming increasingly more interconnected. Many of modern SCADA networks are connected to both the companys enterprise network and the Internet. Furthermore, it is common that the HMI is a commodity PC, which is connected to RTUs and PLCs using standard technologies, such as Ethernet and WLAN (see Fig. 1). Such configuration has exposed these networks to a wide range of security problems. The access to individual subnetworks are secured by firewalls that implement basic network security policy.

Securing networks properly by configuring firewall rules is difficult, time consuming and error-prone task. Wool has analyzed possible threats of incorrectly configured firewalls in [1] and called for methods that would help to improve the quality of firewall rules. The stated observation considers the complexity and the size of firewall rule sets as the main source of errors. He identified major source of difficulties in creating complex firewall configurations. Although Wool considered only a small set of relatively obvious errors, his survey demonstrated that a rule set having 1000 items includes more than 8 errors on average.

The approach described in this paper is close to the work done by Guttman [2], Bera, Ghosh and Dasgupta [3], and Al-Shaer et el [4]. Similarly we develop the method that is able to verify correctness and consistency of firewall configurations against network security policy given a set of simple policy rules. We show a simple translation of policies and firewall rules into logical formulas and describe the Satifiability Modulo Theory (SMT) verification method. The SMT tools employ algorithms for solving logical formulas with respect to combinations of background theories expressed in classical first-order logic with equality. In the present work we use Microsoft's Z3 tool that implements an efficient SMT decisions procedures.

Packet filters implement the basic level of security policies in the network. By restricting the accessibility of certain services, computers or subnetworks, we deploy rough but efficient security measures. Our network model deals only with IP addresses and services or ports. Therefore, the analysis does not reflect hardware or Operating Systems (OS) attacks. The contents of TCP/UDP packets are not examined, but it is possible to extend the description to support this. Our primary goal is to verify safety or resistance of the network with respect to the effect of dynamic routing. Therefore, this classification includes only basic categories of network security properties. Since it can utilize typical fields from IP, TCP, or UDP headers, namely source/destination IP address and service/port allows us to specify wide range of different communications to be analyzed in the network.

This paper is structured as follows: Section II discusses various packet filter representations. Section III presents representation of filtering rules in form of SMT formulas. In Section IV we define a verification method for a single firewall configuration. This is extended to the cascade of firewalls in Section V, thus providing a method for system-wide security policy verification. In Section VII we present a preliminary experimental results showing performance of the presented method. The paper concludes in Section VII by comparing presented method to related work and suggesting further development.

978-1-4673-4471-5/$25.00 © 2013, IEEE

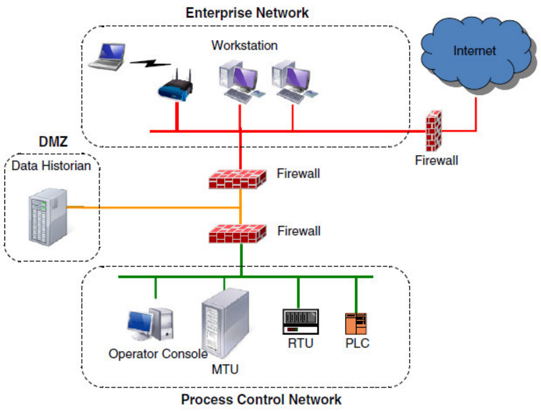

Fig. 1: An example of modern SCADA network.

II. Representation of Packet Filters

Firewall configurations are usually written in form of access control lists (ACL). An ACL format is easy to understand for network administrators and it is also suitable for machine processing. Nevertheless, for an efficient formal analysis this format may represent a problem because it admits conflicting rules. Conflicting rules are pairs of rules that match the same set of packets. These conflicts are solved at runtime by implementing first match semantics. However, certain classes of conflicts can signalize a configuration error, for instance, a rule that completely hides some other rules. Several methods to check conflicts in ACLs and constructing a non-conflicting *rule sets* were proposed, e.g. [5], [6], [7], [8].

Rules have multidimensional structure. Dimensions correspond to fields in a packet header, in particular, source and destination addresses, port numbers and a protocol type. Formally, we define a rule as a tuple $\langle src, dst, srv, act\rangle$, where *src* and *dst* are set of addresses, *srv* is a set of services, and *act* is an action.

A logical formula that is a translation of a simple rule $r = \langle s, d, v, a\rangle$ consists of a conjunction of all selectors. A selector is represented by a predicate that extracts required header field from packet p. Thus, for rule r the formula is written as follows:

$$src\_adr(p) \in s \wedge dst\_adr(p) \in d \wedge service(p) \in v.$$

A list of all possible selectors is shown in Table I A network-mask convention is adapted for representing a sequence of continuous addresses. For instance, address prefix 147.229.12.0/24 is a set of addresses ranging from 147.229.12.0 to 147.229.12.255. We can use the standard set operations, e.g., $src\_adr(p) \in 147.229.12.0/24$ or $dst\_adr(p) \in 147.12.28.0/24 \cup 147.12.30.0/24$. The latter can be expanded to $dst\_adr(p) \in 147.12.28.0/24 \vee dst\_adr(p) \in 147.12.30.0/24$, which allows us to use network-mask format for the canonical address representation.

Often, rule sets implicitly assume the existence of a default rule, which has the lowest priority and matches all packets not matched by any of the previous rules. For a single rule set of an ACL configuration we compute two logical filter representations. A *positive* filter represents all packets permitted by the ACL configuration. A *negative* filter represents all packets denied by the ACL configuration.

TABLE I: Network field selectors

Function	Description
$dst\_adr(p)$	Destination address of a packet p.
$src\_adr(p)$	Source address of a packet p.
$dst\_port(p)$	Destination port of udp or tcp datagram carried in packet p.
$src\_port(p)$	Source port of udp or tcp datagram carried in packet p.
$service(p)$	Service of a packet p.

III. Filter Representation

As proposed in [9] the output of reachability analysis and the input for consecutive security property analysis consist of a collection of reachability sets for forwarding paths in an analyzed network. There are various methods to calculate reachability sets. In this section, we discuss several issues related to these calculations. We overview the problem of efficient address encoding and rule set representation.

Guttman has described an approach to deal with abstract address scheme [2]. The abstract address is a symbolic name of a host or a subnetwork. This address scheme avoids dealing with huge IP address space, which consists of 2^{32}addresses. An abstract packet consists of an abstract source address, an abstract destination address, service identification, and a flow orientation. The flow direction represents the communication direction that is either client to server, or server to client. This approach leads to very reasonable complexity which is dependent on the size of the network and mainly on the number of interesting destinations and services. For an example, considering a network with N different distinguished addresses, S different distinguished services, then the abstract packet space of size will be $N^2 \cdot 2S$.

Different approach was proposed by Bera, Ghosh and Dasgupta in [3]. In their work, the IP address space is explicitly represented by bit variables. The bit variables $s_1, \ldots, s_{32}$ represents a source address, bit variables $d_1, \ldots d_{32}$ represents a destination address, and a vector of bit variables $v_1, \ldots, v_n$ of the appropriate length n, represents a service. A flow direction may be modeled separately by a single bit variable or encoded in the service vector. In this way, there is an explicit representation not only for each packet but also for each network represented in network-mask format.

Independently on whether we use abstract address representation or explicit representation, we construct logical formula for each rule in a filter. These are used in composition of formulas for positive and negative filters. Such formula can be encoded as a SAT instance using the Boolean reduction approach, which is defined in detail for explicit address scheme in [3]. If the abstract address scheme is used each abstract address has to be represented by a single Boolean variable.

These two approaches differ from the number of Boolean variables in generated SAT instances. While explicit representation requires the fixed number of variables, the number of variables used by abstract approach depends on the number of abstract addresses. On the other hand, the former may generate a large number of clauses while the latter tends to keep number of clauses smaller. It remains for future work to analyze and compare both approaches from the practical perspective on real data.

IV. SMT-based Verification Method

In this section, we describe an SMT-based verification method for validation of a network security policy. Given requirements on a packet flow and a filter specification in form of a rule set, we compute a subset of rules that violates these requirements. If the subset is empty than all requirements are satisfied.

First, we present the method to verify a single filter against a security policy. Later this method will be extended for verifying a cascade of filters. Checking if the specified packet flow p is permitted by a filter f it is enough to show that formula $\bar{f} \wedge p$ cannot be satisfied. For instance, assuming that s_0, s_1, s_2 are atomic propositions capturing abstract packet properties. Then filter $\bar{f}$ and a policy p are expressed as follows:

$$\bar{f} = \bigvee \begin{array}{l} s_0 \wedge s_1 \\ s_0 \wedge s_2 \\ s_1 \wedge s_2 \end{array}, \quad p = s_0 \wedge \bar{s_1}$$

In thi case, it is possible to find an assignment $s_0 = 1, s_1 = 0, s_2 = 1$ that satisfies $\bar{f} \wedge p$. While this gives us the required answer we would like to obtain more information to track the problem. To do so, we enrich the filter representation with information that refers to corresponding filtering rules.

$$\bar{f} = \bigvee \begin{array}{l} r = 0 \wedge s_0 \wedge s_1 \\ r = 1 \wedge s_0 \wedge s_2 \\ r = 2 \wedge s_1 \wedge s_2 \end{array}, \quad p = s_0 \wedge \bar{s_1}$$

where r is a bit vector that encodes a rule number. Using this annotation the answer contains information on deny rule that denied the analyzed packet flow, which is, $r = 1$.

To capture network security policy we employ Security Policy Specification Language (SPSL) as defined in [10]. This simple language allows us to express services available between different network zones. For network presented in Fig.1, such policy specification can be as follows:

```
zone ENTP [10.10.100.0/24];
zone DMZ [10.10.10.0/24];
zone PCN [10.10.200.0/24];
zone Inernet [*];

service HTTP = TCP [port = 80];
service SSH = TCP [port=22];
service TELNET = TCP [port=23];

policy p1 = deny [telnet,http]([ENTP],[PCN]);
policy p2 = deny [*]([Internet],[PCN]);
policy p3 = permit [http]([Internet],[DMZ]);
```

For instance, a specification of policy $p1$ can be converted to the following SMT representation:

```
01  (define-fun p_1 () Bool
02  ; deny [telnet,http]([ENTP],[PCN])
03     (and
04         (= (bvand dst_ip PCN_MASK) PCN)
05         (= (bvand src_ip ENTP_MASK) ENTP)
06         (or (and (= pt TCP)(= dst_pn HTTP))
07             (and (= pt TCP)(= dst_pn TELNET))
08         )
09      )
10  )
```

This policy denies telnet and http traffic to the Process Control Network. This is encoded by specifying source (line 5) and destination (line 4) address ranges of the packets that should be denied. Lines 6 and 7 describe protocol type and destination port numbers that correspond to telnet and http traffic, respectively. Addresses are encoded as bit vectors of size 32. Encoding constraints on addresses follows the general pattern:

```
(= (bvand x net_mask) net_addr)
```

Here, `bvand` is a standard bit wise AND operation on bit vectors. Port numbers are encoded as bit vectors of size 16. Using this direct encoding it is possible to directly express policy rules using a standard bit vector theory available in SMT tools.

We demonstrate the translation of ACL configuration to positive and negative filters using the following ACL snippet:

```
R  ip access-list extended paper-example
1   permit icmp any any echo-reply
2   permit icmp any any echo
3   deny   ip any 10.10.10.0 0.0.0.255
4   deny   ip any 10.10.11.0 0.0.0.255
5   permit  ip any any
```

These five rules permit any icmp echo and echo-reply traffic and forbid other traffic to target network. The translation to SMT yields four definitions of functions. Note that default permit rule is not translated.

```
(define-fun f1_r1 () Bool
; permit icmp any any echo-reply
    (and
       (= pt ICMP)
       (= dst_pn ECHO_REPLY)
    )
)
(define-fun f1_r2 () Bool
; permit icmp any any echo
    (and
       (= pt ICMP)
       (= dst_pn ECHO)
    )
)
(define-fun f1_r3 () Bool
; deny   ip any 10.10.10.0 0.0.0.255
    (and
       (= (bvor dst_ip #x000000ff) #x0a0a0aff)
    )
)
(define-fun f1_r4 () Bool
; deny   ip any 10.10.11.0 0.0.0.255
    (and
       (= (bvor dst_ip #x000000ff) #x0a0a0bff)
    )
)
```

Rules constraint only properties explicitly defined. Argument `any` is not represented as it expresses that the variable is constrained by the valid range of the corresponding type, which is implicitly enforced by the type system of SMT. The translation of addresses and wild cards are according to the following pattern:

```
(= (bvor x wildcard) (bvor address wildcard))
```

To verify that ACL obeys a network security policy we need to obtain a representation in form of two partial filters. The negative filter, denoted as `f1_deny`, is a boolean formula that is satisfied for all denied abstract packets. Likewise, the positive filter, denoted as `f1_permit`, is a boolean formula that is satisfied for all permitted packets. We use this splitting to simplify the process of verification and finding counter-examples. The general method for computation of permit and deny filters is presented as Algorithm 1. We will explain the idea of this algorithm on an example of a deny filter. A list of ACL rules is processed in a reverse order. The deny filter formula is constructed in several steps. The immediate result of each step is denoted as f_d^i. Initially, f_d^0 is empty. The formula f_d^{i+1} is constructed as follows:

- If rule r is deny than its logical representation ϕ_r is added to formula $f_d^{i+1} = f_d^i \vee \phi_r$.
- If rule r is permit than its logical representation ϕ_r is combined with filter as $f_d^{i+1} = f_d^i \wedge (\neg \phi_r)$.

Note that in the algorithm the construction of a formula is slightly modified to improve compactness of the resulting formula. All consecutive rules sharing the same action is threated in a single step. Thus, in case of deny rule, we have $f_d^{i+1} = \bigvee f_d^i, \phi_{r_1}, \ldots, \phi_{r_n}$. The deny filter for ACL from the previous example is generated as follows:

```
01  (define-fun f1_deny () Bool
02       (and
03           (not f1_r1)
04           (not f1_r2)
05           (or
06               (and f1_r4 (= deny 4))
07               (and f1_r3 (= deny 3)))))
```

It can be seen that with deny rules there are annotations referring to ACL rules. The annotations allow us to infer information for counter-examples. The permit rule is computed in similar way. Line 8 contains a representation of permit all rule. Permit/Deny all rules match all abstract packets, thus logical representation is constant `true`.

```
01  (define-fun f1_permit () Bool
02       (or
03           (and f1_r1 (= permit 1))
04           (and f1_r2 (= permit 2))
05           (and
06               (not f1_r3)
07               (not f1_r4)
08               (and true (= permit 5)))
```

Policy verification is performed by checking formulas representing policy and filter by the SMT tool. For restricting policies, p_1 and p_2 it means to find satisfying valuation for $p_1 \wedge f_d$. In SMT syntax this is represented by the following code block:

```
(assert (and f1_permit p_1))
(check-sat)
```

Algorithm 1 Computation of a permit filter

Require: An input access-control list L, represented as an ordered list of rules, $r_1, \ldots, r_n \in L$.

$$r_i \in \left[\begin{array}{l} action : \{permit, deny\}, pt : protocol, \\ src.ip : ip\_range, dst.ip : ip\_range, \\ src.pn : port\_range, dst.pn : port\_range \end{array} \right].$$

Ensure: A boolean formula representing the deny filter f_d.

```
$f_d :=$ true
R = L.Reverse
while R not empty do
  r := R.Pop
  if r.action = permit then
    p := true
    while r.action = permit & R not empty do
      $p := p \wedge \neg \phi_r$
      r = R.Pop
    end while
    $f_d := f_d \wedge p$
  else
    d := false
    while r.action = deny & R not empty do
      $d := d \vee \phi_r$
      r = R.Pop
    end while
    $f_d := f_d \vee d$
  end if
end while
```

The answer of SMT is `unsat`, which means that the conjunction cannot be satisfied and hence the filter f_1 is correct with respect to policy p_1. In case of policy p_2 the result given by SMT is `sat` and a random model is provided, e.g., an assignment satisfying `(assert (and f1_permit p_1))` is as follows:

```
permit = 2, pt = ICMP, src_ip = #x0a0a6400,
dst_pn = #x0800, dst_ip = #x0a0a0a00
```

Such result contains diagnostic information telling us that policy is violated by ACL because permit rule 2 matches ICMP echo-reply packets originated from 10.10.100.0 and destined to 10.10.10.0. However, these packets should be denied according to the policy.

A cascade of filters is verified by applying essentially the same approach as described in previous sections. permit and deny predicates are computed for each filter. Then these filters are combined to a single formula representing the cascade of filters.

- $f_p^c = f_p^1 \wedge \ldots \wedge f_p^n$,
- $f_d^c = f_d^1 \vee \ldots \vee f_d^n$,

where $f_p^1, \ldots, f_p^n$ are permit filter predicates and $f_d^1, \ldots, f_d^n$ are deny filter predicates. Permit filter is combined using $\wedge$ operator as a packet is permitted if it passes all ACL on the path. Contrary, a packet can be filtered by any ACL on the path and thus $\vee$ operator is used.

V. System-wide Analysis

In this section, we discuss an extension of a described method for verification of a security policy to system-wide scope. The main goal is to find a network states that violate the given security policy. Recall that security policy is a list of permitted and denied traffic between specified locations. Performing system-wide analysis amounts to check for every pair of network locations specified in a policy rule the permit or deny requirements on the traffic. As there can be multiple paths between these locations these have to be considered. Once we found that a path violates the policy rule it is reported to the user. Considering SCADA network as shown in Fig. 1. Then the topology of this network is capture by the following specification:

```
(declare-const path (Array Int Bool))

;path 1 = ENTP -> F1.1 -> F2.1 -> PCN
(define-fun fp1_permit () Bool
    (and f1_1_permit f2_1_permit))
;path 2 = ENTP -> F1.1 -> DMZ
(define-fun fp2_permit () Bool
    (and f1_1_permit))
;path 3 = PCN -> F2.2 -> F1.2 -> ENTP
(define-fun fp3_permit () Bool
    (and f2_2_permit f1_2_permit))
;path 4 = PCN -> F2.2 -> DMZ
(define-fun fp4_permit () Bool
    (and f_2_2_permit))

; checking violations for policy 1
(assert (or
    (and fp1_permit p1 (select path 1))
    (and fp2_permit p1 (select path 2))
    (and fp3_permit p1 (select path 3))
    (and fp4_permit p1 (select path 4))))
```

We use array to remark which paths violate the policy. The evaluation of SMT specification leads to finding a counter example in case of policy rule violation. The presented encoding brings any counter example depending on the run of SMT algorithm. However, it would be desirable if the produced counter example represent the largest subset of a rule set that violates a security policy. Using this approach the user is not confronted with an arbitrary counter example in case of policy violation, but with a counter-example that, if applied to path based policy checking, violates the greatest number of paths.

The idea of finding the greatest number of paths, which violates the policy rule is based on binary search procedure that guarantees to find the result in $\log_2 N$ steps. The search environment is initialized by introducing a counter array, which keeps the number of paths violating the policy rule. An index in the array is computed as follows:

$$\textit{sums}\ [i] := \ \textit{sums}\ [i-1] + \ \text{IF}\ path[i]\ \text{THEN}\ 1\ \text{ELSE}\ 0.$$

This initialization is encoded as follows:

```
(define-sort SumT () (Array Int Int))
(declare-const sums SumT)

(assert (= (select sums 0) 0))
```

```
(assert
  (forall ((i Int))
    (ite (select path i)
         (= (store sums i
           (+ (select sums (- i 1)) i)) sums)
         (= (store sums i
           (select sums (- i 1))) sums)
    )
  )
)
```

Note that it is better to unwind the forall statement to avoid dealing with quantifiers. The iteration consists of several steps for i by asserting the following:

```
(assert (= (select sums n) i))
```

Here, n is the total number of paths. Reading *sums* $[n]$ means to get a number of satisfied paths. The iterative steps are guided by the immediate results of SMT executions for the current instance.

VI. Results and Discussion

We experimentally implemented the proposed SMT-based method using Microsoft's Z3 tool. The results of execution of this method on problems of various size are shown in Table II.

The testing set of filtering rules consists of filters generated using the tool called ClassBench [11]. This generator is equipped with templates of filtering rules derived from a collection of real firewall configurations. The tool generates ACLs of different sizes and parameters. For our purpose, we generated filters for different templates, denoted as acl1-3 and fw1 and fw2. These templates differ by the number of conflicting rules. For every template a range of filters of various size was generated. We use rule sets generated for these templates as an imput to our tool that translated them to SMT specification, which was consumed by Z3 tool. We meassured time and memory requirements of the SMT method that checks rule set consistency.

Experiments were performed on a 2.53 Ghz Intel Core 2 Duo machine with 8 GB of RAM running Z3 version 4.3.1 in 64 bit mode. Table II contains results for different sizes of the problem. It can be seen that in most cases the time and memory consumption of the methods increases linearly with the number of rules in firewall configuration. The irregulairites are caused by the different number of conflicting rules in those samples.

VII. Conclusions

In this paper, we presented an approach for verifying ACL configurations by translating them to rule sets, which can be formally analyzed using SMT tools. The proposed method enables network administrators to observe the quality and correctness of firewall configurations, which improves the overall security in administered networks. This technique can be combined with other approaches supposed for securing industrial networks. The overview of security threats in industrial networks were presented by Alcaraz et el in [12] and later by Cardenas et el in [13]. These analyses emphasize the

TABLE II: Time and memory requirements of SMT procedure

Time[s]	10	100	1000	10000	100000
acl1	0.01	0.02	0.11	1.43	13.91
acl2	0.01	0.02	0.10	1.13	14.36
acl3	0.01	0.02	0.11	1.22	39.95
fw1	0.01	0.02	0.13	1.08	30.59
fw2	0.01	0.03	0.11	1.42	13.81

Memory[MB]	10	100	1000	10000	100000
acl1	2.35	2.95	7.81	55.41	459.11
acl2	2.36	2.94	7.73	55.45	460.55
acl3	2.31	2.97	7.84	55.48	456.98
fw1	2.34	2.98	7.84	55.45	455.28
fw2	2.34	3.00	7.89	55.45	458.47

importance of a combination of reactive and proactive methods in order to secure the system against deception and DoS attack.

Description of network security properties is related to the classification of threats and intrusion. There are plenty of different network security problems, such as HTTP attacks, spam, TCP flooding, DoS attacks, Web server misuse, spoofing and sniffing etc. Protection of critical components and network infrastructure is identified as a key requirements for improving security in SCADA system by Hentea in [14].

Analysis of firewall configuration has been intensively studied. Namely, Guttmannin [2] proposed algorithm for computing reachibilty sets based on the firewall configurations. Bera et al in [10] proposed SAT-based methods for verification of security policy. Al-Shaer et al. [15] uses similar approach for representation of ACLs as permit and deny predicates. Their verification methods employ the BDD representation in model-checking procedure.

The network model presented in this paper deals only with IP addresses and services or ports. Therefore, the analysis does not reflect hardware or OS attacks. It also does not examine the contents of TCP/UDP packets. Therefore, this classification only includes selected categories of network security properties. Since it can utilize typical fields from IP, TCP, or UDP headers, namely source/destination IP address and service/port, it allows to specify wide range of different communications to be analyzed in the network.

In this paper we demonstrated the problem of automatic security analysis of IP based industrial networks. The presented verification method aims at validating network design against the absence of security and configuration flaws. The verification technique is based on the encoding problem into SMT instance solved automatically by the solver tool.

References

[1] A. Wool, "Trends in Firewall Configuration Errors: Measuring the Holes in Swiss Cheese," *IEEE Internet Computing*, vol. 14, no. 4, pp. 58–65, Jul. 2010.

[2] J. Guttman, "Filtering postures: Local enforcement for global policies," in *IEEE Symposium on Security and Privacy*. IEEE Comput. Soc. Press, 1997, pp. 120–129.

[3] P. Bera, S. Ghosh, and P. Dasgupta, "Formal Verification of Security Policy Implementations in Enterprise Networks," *Information Systems Security*, pp. 117–131, 2009.

[4] E. Al-Shaer, W. Marrero, A. El-Atawy, and K. ElBadawi, "Towards global verification and analysis of network access control configuration," *DePaul University, Chicago, IL, USA, Tech. Rep*, 2008.

[5] L. Cholvy and F. Cuppens, "Analyzing consistency of security policies," in *Security and Privacy, 1997. Proceedings., 1997 IEEE Symposium on*. IEEE, 1997, pp. 103–112.

[6] a. Hari, S. Suri, and G. Parulkar, "Detecting and resolving packet filter conflicts," in *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3. IEEE, 2000, pp. 1203–1212.

[7] E. Al-Shaer and H. Hamed, "Discovery of policy anomalies in distributed firewalls," in *Ieee Infocom 2004*. Ieee, 2004, pp. 2605–2616.

[8] S. P. Hidalgo, R. Ceballos, and R. M. Gasca, "Fast Algorithms for Consistency-Based Diagnosis of Firewall Rule Sets," *2008 Third International Conference on Availability, Reliability and Security*, pp. 229–236, Mar. 2008.

[9] G. Xie, D. Maltz, A. Greenberg, G. Hjalmtysson, and J. Rexford, "On static reachability analysis of IP networks," *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, pp. 2170–2183, 2005.

[10] P. Bera, S. Maity, S. Ghosh, and P. Dasgupta, "A Query based Formal Security Analysis Framework for Enterprise LAN," *2010 10th IEEE International Conference on Computer and Information Technology*, no. Cit, pp. 407–414, Jun. 2010.

[11] D. E. Taylor, "ClassBench: A Packet Classification Benchmark," *IEEE/ACM Transactions on Networking*, vol. 15, no. 3, pp. 135–511, Jun. 2007.

[12] C. Alcaraz, G. Fernandez, R. Roman, A. Balastegui, and J. Lopez, "Secure Management of SCADA Networks," *New Trends in Network Management, Cepis UPGRADE*, vol. 9, no. 6, pp. 22–28, 2008.

[13] A. a. Cardenas, S. Amin, and S. Sastry, "Secure Control: Towards Survivable Cyber-Physical Systems," in *Proceedings of the 28th International Conference on Distributed Computing Systems Workshops*. Ieee, Jun. 2008, pp. 495–500.

[14] I. N. Fovino, A. Carcano, and M. Masera, "A Secure and Survivable Architecture for SCADA Systems," *2009 Second International Conference on Dependability*, pp. 34–39, Jun. 2009.

[15] E. Al-Shaer, H. Hamed, and R. Boutaba, "Conflict classification and analysis of distributed firewall policies," *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 10, 2005.