Verify Consistency between Security Policy and Firewall Policy with Answer Set Programming

Yiwen Liang, Wenjun Deng
Computer School of Wuhan University
Wuhan, 430072, China
ywliang@whu.edu.cn, Whuwjdeng@gmail.com

***Abstract*—Firewalls are core elements in network security, the effectiveness of firewall security is dependent on configuring firewall policy correctly. Firewall policy is a lower-level policy which describes how firewall actually implements security policy. Security policy is a higher-level policy which defines the access that will be permitted or denied from the trusted network. Compare with software engineer, security policy is a design, firewall policy is a set of codes. It is useful to discover inconsistency between security policy and firewall Policy. In this paper, we present a method of verifying consistency between security policy and firewall policy, which applies the idea of model checking. First of all, two policies and the consistency are represented with logic programs. Then the verification is applied by testing whether the logic formula of the consistency is satisfied in the semantics of the logic programs. Furthermore, We prove that the method has an unique answer which can be computed in polynomial time.**

I. Introduction

To protect a trusted network from an untrusted network, security administrators design a security policy which defines those services that will be permitted or denied from the restricted network. A firewall is a collection of components or a system which implements a security policy with a set of rules. The rule set is a firewall policy which describes how the firewall will carry out restricting the access and filtering the services as defined in the security policy. The protection depends on that the firewall policy is consistent to the security policy [1].

Unfortunately, it has been observed that most firewall policies on the Internet are poorly designed and have many errors[2]. There are three reasons for that[3]:First, firewall policy languages tend to be arcane, very low level, and highly vendor special. Secondly, firewall policy is sensitive to rule order. Several rules may match a particular packet, and usually the first matching rule is applied. So changing the rule order, or inserting a correct rule in a wrong place, may lead to unexpected behavior and possible security breaches. Thirdly, the rule number of firewall policy is huge.

The three reasons show that it is complicated for administrators to check the consistency between security policy and firewall policy. It may lead to a situation that security policy is right, while firewall policy is wrong. To solve the problem, we present a method of verifying consistency between security policy and firewall policy using answer set programming(ASP). The term ASP was coined by Vladimir Lifschitz to name a new declarative programming paradigm that has its roots in stable model semantics of logic programs and implementations of this semantics developed in the late 90's. ASP is already used to solve access control problem[4]. In paper[4], authors propose a formal authorization language *AL* providing its semantics through Answer Set Programming. ASP has several properties fit for solving access control problem: Firstly, while ASP has its roots in logic programming, it can use high level language represent security policy and firewall policy. Secondly, ASP has a few solvers, such as smodels[5], which are able to compute semantics automatically and efficiently. Thridly, ASP carries out non-monotonic reasoning through negation as failure. Firewall configuration and firewall policy are both non-monotonic because former rules are prior to latter rules.

In this paper, we present a method of verifying consistency between security policy and firewall policy, which applies the idea of model checking(Figure 1). The security policy and firewall policy are both represented with logic model–logic programs, and consistency is represented as a formal property of the policies with a logic formula. Then the consistency problem is solved by testing whether the logic formula of the consistency is satisfied in the semantics of the logic programs. We transform security policy into a logic program *LA*, and transform firewall policy into a logic program *LB*. And then *LA*'s semantics *SA* and *LB*'s semantics *SB* are computed respectively. After that, the consistence is determined by the semantics of the logic program *LC* which consists of *SA*, *SB* and the logic formula of the consistency property. Furthermore, We also prove that the method has an unique answer which can be computed in polynomial time.

II. Related Work

To solve that problem, researchers present automatic methods to analyze and verify firewall configuration. In papers[6], [7], [8], authors present a set of techniques and algorithms that automatically discover policy anomalies in centralized and distributed firewalls to reveal rule conflicts and potential problems. In papers[9], [10], [3], authors design and implement firewall analysis tools, which allow administrators to easily discover and test the global firewall configuration. E. Pasi and Z. Jukka present a tool based on constraint logic programming which allows users to write higher level operations for detecting common configuration mistakes[11]. S. Pozo et. al present a CSP based approach to automatically

978-0-7695-3336-0/08 $25.00 © 2008 IEEE
DOI 10.1109/CSSE.2008.1089

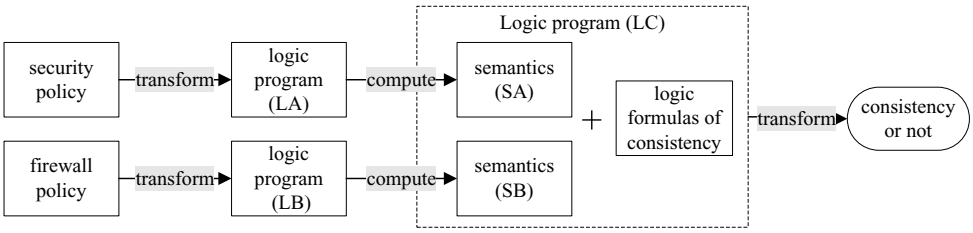

Fig. 1. Process of Verifying Consistency

diagnose firewall rule sets[12]. Different from others' work, S. Pozo et al present the specification of a high level policy to diagnose the firewall conformity with a specified firewall policy.

The main difference between these works and ours is that we verify the the inconsistency between security policy and firewall policy, instead of focusing on the analysis of firewall policy. Security policy is a design, firewall policy is a set of codes that implement the design. It is important to search errors in the codes, and it is more important to verify the consistency between the design and the codes. In [12], authors construct a Constraint Satisfaction Problem to detect and identify the possible inconsistencies between the specified policy and the firewall rule set. However, the authors only verify the soundness of the consistency as that "for each security policy , there exists a firewall policy which matches the security policy." They do not verify the completeness of the consistency that "for each firewall policy, there exists a security policy which matches the firewall policy." Our method verifies the soundness and the completeness. And We also prove that the method has an unique answer which can be computed in polynomial time.

III. Answer Set Programing

A **logic program** consists of rules of the form

$$L_0 \leftarrow L_1, \ldots, L_m, not\ \ L_{m+1}, \ldots, not\ \ L_n. \qquad (1)$$

where $0 \leq m \leq n$ and L_i are literals, $0 \leq i \leq n$. A literal is an expression of the form A or $\neg A$, where A is an atom. We call the symbol $\neg$ "classical negation" to distinguish it from the symbol not used for negation as failure. The L_0 is the $head$ of rule, and $L_i(1 \leq i \leq n)$ is the $body$ of rule.

Answer Set is a semantics for logic programs. Firstly, we give the definition of answer set for logic program without not.

A $Herbrand\ interpretation\ S$ of a logic program P is said to $satisfy$ the rule $L_0 \leftarrow L_1, \ldots, L_m$ if

1) $L_0 \neq \emptyset$: $\{L_1, \ldots, L_m\} \subseteq S$ implies $L_0 \in S$.
2) L_0=$\emptyset$: $\{L_1, \ldots, L_m\} \nsubseteq S$.

For a logic program P without not, a $Herbrand\ model\ M$ is a $Herbrand\ interpretation$ that satisfies all rules in P. For a logic program P without not, a $answer\ set\ S$ is a $Herbrand\ model$ of P, which is minimal among all $Herbrand\ models$ of P.

And then, for a logic program P, given a set of literals S, we give two steps to transform a logic program P with not into a logic program P^S without not. P^S is a program obtained from P as follow:

1) delete each rule that has a not L_i in its body with $L \in S$, and
2) delete all not L_i in the bodies of the remaining rules.

Obviously, answer set of P^S is defined, which is called $\alpha(P^S)$. For a logic program P , A $answer\ set\ S$ is a set of literals, which is minimal set satisfies S=$\alpha(P^S)$.

I. Niemelä introduce a number of interesting applications of ASP, including planning, decision support for the fight controllers of space shuttles, web-based product configuration, configuration of a Linux system distribution, computer aided verification, VLSI routing, network management, security protocol analysis, network inhibition analysis, linguistics, data and information integration, and diagnosis[13].

IV. Logic Model for verifying consistency

In this section, we present the logic model based on ASP for representing security policy and firewall policy. The logic model consists of four parts: packet, security policy and firewall policy, order, compare.

A. packet

In this section, we present the predicate for representing a packet, and the logic formula for generating the universe of packet. In our model, all the objects of firewall policy and firewall rules, such as host and port, have names. For example, in Figure 2, every service and every client has a name which identifies unique node in the networking instead of the IP address. Because meaningful names are more expressive than raw IP addresses and port numbers, a high level of abstraction is more comprehensible for security administrator. Both security policy and firewall policy consist of different fields that represent information relevant to the filtering. The fields are: **protocol**, **source IP**, **destination IP**, **source port**, **destination port**, **action**.

The predicates used to represent those fields are:

- **protocol**: predicate protocol(X) describes that X is a protocol. For example, protocol(tcp) means tcp is a protocol.
- **source and destination IP**: predicate ip(X) describes that X is a IP address. In our model, X is a name of node in the network. For example, in the Fig 2, we use ip(c1) to represent the IP address of client c1.

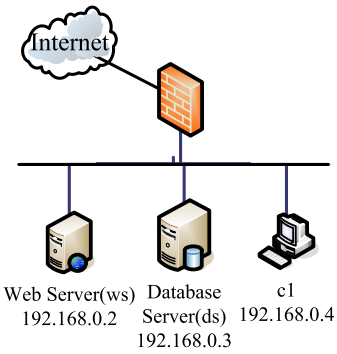

Fig. 2. A Firewall

- **source port and destination port**: predicate port(X) describes that X is a port number. In our model, X is a name of a port. For example, in the Fig 2, we use port(http) to represent the port 80 for web service.

We use predicate pac(Pro,Sip,Dip,Spt,Dpt) to represent packet. The rule that generates packet is:

```
pac(X,Y1,Y2,Z1,Z2) :- protocol(X), ip(Y1), ip(Y2),
                      port(Z1), port(Z2), Y1!=Y2.
```

B. Security Policy and Firewall Policy

In this section, we present logic formulas for representing a security policy and a firewall policy. The difference between them is that security policy use a high-level, nature language, and firewall policy use a low-level, rule based language. For example, in FIG 2, a security policy says that the web server *ws* provides web service using 80 port for any node in the network except client c1. The firewall polices with the same meaning of the security policy(Cisco style) are:

```
1 deny    tcp  192.168.0.4  192.168.0.4  eq 80
2 permit  tcp  any          192.168.0.4  eq 80
```

We use logic programs to represent security policy and firewall policy. The predicate p_rule(Num, Act, Pro, Sip, Dip, Spt, Dpt) represents a security policy, and the predicate f_rule(Num, Act, Pro, Sip, Dip, Spt, Dpt) represents a firewall policy. The former predicate means that the Num-th policy says that pac(Pro, Sip, Dip, Spt, Dpt) is dealt with action Act. The latter predicate means that the Num-th rule says that pac(Pro, Sip, Dip, Spt, Dpt) is dealt with action Act. The logic program rules represent the security policy and firewall policies mentioned in the above paragraph are:

C. Order

Both security policy and firewall policy are sensitive to policy order. It means that Several rules may match a particular packet. First of all, conflict policies and redundancy policies in security policy and firewall policy should be omitted. We use predicate p_ab(Num1, Act1, Pro, Sip, Dip, Spt, Dpt) to

security policy

```
p_rule(1,accept,Pro,Sip,Dip,Spt,Dpt) :-
                    pac(Pro,Sip,Dip,Spt,Dpt), Pro==tcp,
                    Sip!=c1, Dip==ds, Dpt==http.
```

firewall policy

```
f_rule(1,deny,Pro,Sip,Dip,Spt,Dpt) :-
                    pac(Pro,Sip,Dip,Spt,Dpt),  Pro==tcp,
                    Sip==c1, Dip==ds, Dpt==http.
f_rule(2,accept,Pro,Sip,Dip,Spt,Dpt) :-
                    pac(Pro,Sip,Dip,Spt,Dpt),  Pro==tcp,
                     Dip==ds, Dpt==http.
```

represent that there are a p_rule(Num1, Act1, Pro, Sip, Dip, Spt, Dpt) and a p_rule(Num2, Act2, Pro, Sip, Dip, Spt, Dpt), and Num1>Num2. In other words, p_ab(Num1, Act1, Pro, Sip, Dip, Spt, Dpt) means p_rule(Num1, Act1, Pro, Sip, Dip, Spt, Dpt) should be omitted. We use predicate p_access(Num, Act, Pro, Sip, Dip, Spt, Dpt) to represent the security policies after conflict policies and redundancy policies are omitted. The logic program rules for the omission are:

```
p_ab(X,A1,Pro,Sip,Dip,Spt,Dpt) :-
                    p_rule(X,A1,Pro,Sip,Dip,Spt,Dpt),
                    p_rule(Y,A2,Pro,Sip,Dip,Spt,Dpt), Y < X.
p_access(X,Action,Pro,Sip,Dip,Spt,Dpt) :-
                    p_rule(X,Action,Pro,Sip,Dip,Spt,Dpt),
                    not p_ab(X,Action,Pro,Sip,Dip,Spt,Dpt).
```

On the other hand, predicate f_ab(Num, Act, Pro, Sip, Dip, Spt, Dpt) and f_access(Num, Act, Pro, Sip, Dip, Spt, Dpt) are used to omit conflict policies and redundancy policies of firewall policy.

D. Compare

Definition 1: A security policy p_access(Num1, Act1, Pro1, Sip1, Dip1, Spt1, Dpt1) and a firewall policy f_access(Num2, Act2, Pro2, Sip2, Dip2, Spt2, Dpt2) match each other if and only if (Act1=Act2) $\wedge$ (Pro1=Pro2) $\wedge$ (Sip1=Sip2) $\wedge$ (Dip1=Dip2) $\wedge$ (Spt1=Spt2) $\wedge$ (Dpt1=Dpt2).

Definition 2: security policy is consistent to firewall policy if and only if for each policy in security policy, there exists a policy in firewall policy matching the security policy, and for each policy in firewall policy, there exists a policy in security policy matching the firewall policy.

We use predicate conflict(X, Y, A1, A2, Pro, Sip, Dip, Spt, Dpt) to represent that there is a inconsistency between p_access(Num1, Act1, Pro, Sip, Dip, Spt, Dpt) and f_access(Num1, Act2, Pro, Sip, Dip, Spt, Dpt). According to the definition 1 and 2, the logic program rule verifying consistency is:

```
inconsistent(X,Y,A1,A2,Pro,Sip,Dip,Spt,Dpt) :-
                    p_access(X,A1,Pro,Sip,Dip,Spt,Dpt),
                    f_access(Y,A2,Pro,Sip,Dip,Spt,Dpt), A1!=A2.
```

```
ip(ds). ip(ws). ip(c1). ip(other).    port(http). port(oracle). port(other).
protocol(tcp). protocol(other).

pac(X,Y1,Y2,Z1,Z2) :- protocol(X), ip(Y1), ip(Y2),
                       port(Z1), port(Z2), Y1!=Y2.

p_rule(1,accept,Pro,Sip,Dip,Spt,Dpt) :- pac(Pro,Sip,Dip,Spt,Dpt),
                                        Pro==tcp, Dip==ws, Dpt==http.
p_rule(2,accept,Pro,Sip,Dip,Spt,Dpt) :- pac(Pro,Sip,Dip,Spt,Dpt), Pro==tcp,
                                        Sip==c1, Dip==ds, Dpt==oracle.
p_rule(3,deny,Pro,Sip,Dip,Spt,Dpt) :- pac(Pro,Sip,Dip,Spt,Dpt).

p_ab(X,A1,Pro,Sip,Dip,Spt,Dpt) :- p_rule(X,A1,Pro,Sip,Dip,Spt,Dpt),
                                  p_rule(Y,A2,Pro,Sip,Dip,Spt,Dpt), Y < X.
p_access(X,A1,Pro,Sip,Dip,Spt,Dpt) :- p_rule(X,A1,Pro,Sip,Dip,Spt,Dpt),
                                  not p_ab(X,A1,Pro,Sip,Dip,Spt,Dpt).

hide.
show p_access(X,A1,Pro,Sip,Dip,Spt,Dpt).
```

Fig. 3. A Logic Program of Security Policy

```
ip(ds). ip(ws). ip(c1). ip(other).    port(http). port(oracle). port(other).
protocol(tcp). protocol(other).

pac(X,Y1,Y2,Z1,Z2) :- protocol(X), ip(Y1), ip(Y2),
                       port(Z1), port(Z2), Y1!=Y2.

f_rule(1,accept,Pro,Sip,Dip,Spt,Dpt) :-pac(Pro,Sip,Dip,Spt,Dpt),
                                        Pro==tcp, Dip==ws, Dpt==http.
f_rule(2,accept,Pro,Sip,Dip,Spt,Dpt) :- pac(Pro,Sip,Dip,Spt,Dpt),
                                        Pro==tcp, Sip==c1, Dip==ds.
f_rule(3,deny,Pro,Sip,Dip,Spt,Dpt) :- pac(Pro,Sip,Dip,Spt,Dpt).


f_ab(X,A1,Pro,Sip,Dip,Spt,Dpt) :- f_rule(X,A1,Pro,Sip,Dip,Spt,Dpt),
                                  f_rule(Y,A2,Pro,Sip,Dip,Spt,Dpt), Y < X.
f_access(X,A1,Pro,Sip,Dip,Spt,Dpt) :- f_rule(X,A1,Pro,Sip,Dip,Spt,Dpt),
                                  not f_ab(X,A1,Pro,Sip,Dip,Spt,Dpt).
hide.
show f_access(X,A1,Pro,Sip,Dip,Spt,Dpt) .
```

Fig. 4. A Logic Program of Firewall Policy

V. An Example

In this section, we present an example to illustrate how the model works. The complete process has been implemented using the ASP solver smodels[5]. The example network (Figure 2) consists of three zones and several systems. An example firewall policy for this network, written in natural language, is:

1) every node in the network can access host Web service ws through port http(80).
2) client c1 can access database service ds through port oracle(1521).
3) except the condition in former two rules, every nodes in the network can not access each other.

Figure 3 describes a logic program of the security policy.

Suppose that firewall policy is not configured correctly according to the security policy. The firewall administrator thinks that client c1 can access database service ds. Figure 4 describes a logic program of the firewall policy.

```
smodels version 2.32. Reading...done
Answer: 1
Stable Model inconsistent(3,2,deny,accept,tcp,c1,ds,othe
r,other) inconsistent(3,2,deny,accept,tcp,c1,ds,oracle,o
ther) inconsistent(3,2,deny,accept,tcp,c1,ds,http,other)
 inconsistent(3,2,deny,accept,tcp,c1,ds,other,http) inco
nsistent(3,2,deny,accept,tcp,c1,ds,oracle,http) inconsis
tent(3,2,deny,accept,tcp,c1,ds,http,http)
True
Duration: 0.93
Number of choice points: 0
Number of wrong choices: 0
Number of atoms: 1156
Number of rules: 1155
Number of picked atoms: 0
Number of forced atoms: 0
Number of truth assignments: 1155
Size of searchspace (removed): 0 (0)
```

Fig. 5. The Answer Set of the Example

Using smodels, we get semantics SA of the logic program LA and semantics SB of the logic program LB. union of those two semantics plus the rules of verifying consistency(in section 3.4) is the logic program LC, and its semantics is SC. Figure 5 displays the answer set of the LC. It points out that the policy is not consistence with the rule set, because the third policy is conflict with the second rule.

VI. Computational Properties

In this section, we study basic computational properties of our method.

We present the concept of local stratification for extended logic programs[14].

Definition 3: Let *P* be an extended logic program and *Lit* be the set of all ground literals of *P*:

1) A *local stratification* for *P* is a function *stratum* from *Lit* to the countable ordinals.
2) Given a *local stratification stratum*, we extend it to ground literals with negation as failure by setting *stratum(not L)* = *stratum(L)* + 1, where *L* is a ground literal.
3) A rule $L_0 \leftarrow L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n$ in *P* is locally stratified with respect to *stratum* if *stratum*(L_0) $\geq$ *stratum*(L_i), where $1 \leq i \leq m$, and *stratum*(L_0) $>$ *stratum*($notL_j$), where $m+1 \leq i \leq n$
4) *P* is called *locally stratified* with respect to *stratum* if all of its rules are locally stratified. *P* is called *locally stratified*if it is locally stratified with respect to some local stratification.

Lemma 1 ([4]): Let P_1, P_2 be two locally stratified logic programs. Program $P_1 \bigcup P_2$ is locally stratified if $head(P_1) \bigcap head(P_2) = \emptyset$

Theorem 1: Logic program LA, LB and LC in Figure 1 are locally stratified.

Proof: It is obviously that LA and LB are locally stratified according to the definition 3. Because LC is the union of LA and LB and the logic program in 4.3 section, LC is locally stratified according to the lemma 1. ■

Theorem 2: All of logic program LA, LB and LC in Figure 1 respectively have a unique model that can be computed in polynomial time.

Proof: proof From [14], a locally stratified AnsProlog program has a unique answer set([14] proposition 33)and can be computed in polynomial time([14] theorem 6.2.7). Based on Theory 1, we can prove the theory 2. ■

VII. Conclusion

This paper presents a method for automatically verifying consistency between security policy and firewall policy. Such a method is extremely useful in many ways. For example, a firewall administrator can use this tool to unambiguously demonstrate to their manager that the firewall satisfies the organization's security policy.

This method is based on Answer Set Programming which is a new declarative programming paradigm. While ASP has its roots in logic programming, it can be based on other formal systems such as propositional or first-order logic. Since ASP is capable of representing knowledge, this method can be applied with all kinds of firewall vendors.

We prove that the method has a unique answer which can be computed in polynomial time. So this method will work also in practice. Because logic language it use, it is more comprehensible for security administrator.

Acknowledgment

The work was supported by National Natural Science Foundation of China under Grant No. 60573038. The authors would like to thank Yong Ai, Kefu Gao, who gave a useful comment on preliminary version of this paper.

References

[1] S. Cobb, "Establishing firewall policy," in *Southcon/96. Conference Record*, Orlando, FL, USA, Jun. 1996, pp. 198–205.

[2] A. Wool, "A quantitative study of firewall configuration errors," *Computer*, vol. 37, pp. 62–67, Jun. 2004.

[3] A. Mayer, A. Wool, and E. Ziskind, "Offline firewall analysis," *International Journal of Information Security*, vol. 5, pp. 125–144, Mar. 2005.

[4] S. Wang and Y. Zhang, "Handling distributed authorization with delegation through answer set programming," *International Journal of Information Security*, vol. 6, pp. 27–46, Mar. 2007.

[5] I. Niemela and P. Simons, "Smodels – an implementation of the stable model and well-founded semantics for normal logic programs," in *the International Conference on Logic Programming and Nonmonotonic Reasoning*, 1997, pp. 420–429.

[6] E. S. Al-Shaer and H. H. Hamed, "Firewall policy advisor for anomaly discovery and rule editing," in *Integrated Network Management, 2003. IFIP/IEEE Eighth International Symposium on*, Orlando, FL, USA, May 2003, pp. 17–30.

[7] ——, "Modeling and management of firewall policies," *IEEE Journal on Selected areas in Communication*, vol. 1, pp. 1–10, 2004.

[8] E. Al-Shaer, H. Hamed, R. Boutaba, and M. Hasan, "Classification and analysis of distributed firewall policies," *IEEE Journal on Selected areas in Communication*, vol. 23, pp. 2069–2084, 2005.

[9] A. Wool, "architecting the lumeta firewall analyzer," in *10th conference on USENIX Security Symposium*, 2001, pp. 381–420.

[10] Y. Bartal, A. Mayer, K. Nissim, and A. Wool, "Firmato: A novel firewall management toolkit," *ACM Transactions on Computer Systems*, vol. 20, pp. 381–420, Apr. 2005.

[11] P. Eronen and J. Zitting, "An expert system for analyzing firewall rules," in *Nordic Workshop on Secure IT-Systems*, Nov. 2001.

[12] S. Pozo, R. Ceballos, and R. M. Gasca, "Csp-based firewall rule set diagnosis using security policies," in *The Second International Conference on Availability, Reliability and Security*, 2007, pp. 723 – 729.

[13] I. Niemela, "Answer set programming: A declarative approach to solving search problems," in *Logics in Artificial Intelligence*. Springer Berlin, 2006, pp. 15–18.

[14] C. Baral, *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.