Discover Inconsistencies between Firewall Policies

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

Abstract

Firewalls are core elements in network security, the effectiveness of firewall security is dependent on configuring firewall policy correctly. Firewall policy describes the access that will be permitted or denied from the trusted network. In corporate network, several firewalls are set up and administrated by different teams. The consistency between those firewall policies is crucial to corporate network security. In this paper, we present a method of discover the inconsistencies between two given firewall policies. Firstly, Firewall policies are represented with logic formulas, the consistency regarded as a property of firewall policies is represented with logic formulas. The inconsistencies are discovered from the semantics of the union of the logic formulas. Furthermore, we prove that the method has the unique answer which can be computed in polynomial time. Firewall policies often need to be updated, as networks evolve. Many firewall policy errors are caused by the side effects of policy updates. Our method can be used to verify the updates of firewall policy by comparing the policy before changes and the policy after changes.

1. Introduction

Firewalls are crucial elements in network security, and they are widely used to protect private networks from untrusted networks. 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. In corporate network, several firewalls are deployed and administrated by different teams. Verifying consistency between those firewall policies is crucial to corporate network[1] .

Unfortunately, verifying firewall policy is a complicated work, most firewall policies on the Internet are poorly designed and have many errors[2]. There are three reasons for that[3]: Firstly, 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 a firewall policy is huge. Research[3] reveals that the average rule number of 37 Check Point Firewall policies is 968.

The three reasons explain that it is necessary to present an automatic method for discovering inconsistencies between firewall policies. We present such a method based on *ASP*(Answer Set Programming). 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. A formal authorization language *AL* is propose to solve access control problems which providing its semantics through *ASP*[4]. *ASP* has several properties fit for solving access control problems: Firstly, while *ASP* has its roots in logic programming, it can use high level language to 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. Thirdly, *ASP* applies nonmonotonic reasoning through negation as failure. Firewall configuration and firewall policy are both nonmonotonic because former rules are prior to latter rules.

In this paper, we present a method of automatically discovering inconsistencies between two given firewall policies (Figure 1). Firstly, the two given policies are both represented with logic programs *LA* and *LB*. The semantics *SA* of *LA* and the semantics *SB* of *LB* are computed respectively. Meanwhile the consistency regarded as a formal property of firewall policy is represented with a logic program *LC*. By merging *SA*, *SB* and *LC*, we get a logic program *LD*. The semantics *SD* represents the discovery of the inconsistencies. And, we also prove that the method has the unique answer which can be computed in polynomial time.

2. Related works

Researchers present automatic methods to analyze and verify firewall policy. Al-Shaer et al. present a set of techniques and algorithms that automatically discover policy anomalies in centralized and distributed

978-0-7695-3488-6/08 $25.00 © 2008 IEEE
DOI 10.1109/KAM.2008.160

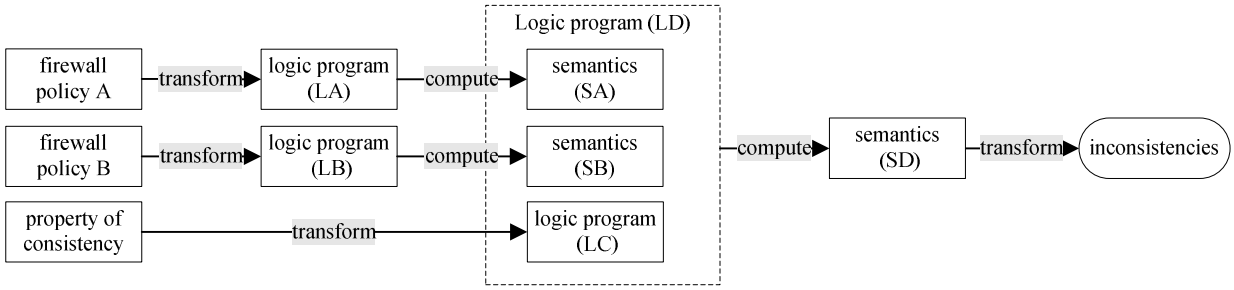

Figure 1 Process of verifying consistency

firewalls to reveal rule conflicts and potential problems. The core data structure used in those papers is Firewall Policy Tree[6-8]. In papers[3][9][10], authors design and implement firewall analysis tools, which allow administrators to easily discover and test the global firewall configuration. Their tools complement existing vulnerability analyzers and port scanners, as they can be used before a policy is actually deployed, and they operate on a more understandable level of abstraction. 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 diagnose firewall rule sets[12].

The main difference between these works and ours is that we present a method that can compare two given firewalls and discover all inconsistencies between them in human readable format. While other works focus on one firewall policy. The most related work to ours is [1]. In that paper, Authors present a method that can compare two given firewalls and output all functional inconsistencies between them. The core data structure they use is Firewall Policy Tree. Our work can not only discover and output the inconsistencies, but also locate an inconsistency to a rule of the firewall policy which work of [1] does not do. It means that our work provides more information about the inconsistencies.

3. Answer set programming

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

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

where $0 \leqslant m \leqslant n$ and Li are literals, $0 \leqslant i \leqslant n$. A literal is an expression of the form A or ¬ A, where A is an atom. We call '¬ ' 'classical negation' to distinguish it from '*not*' used for 'negation as failure'. The L_0 is the head of rule, and L_i $(1 \leqslant i \leqslant n)$ 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 $L_0 \leftarrow L_1,....,L_m,not\ L_{m+1},....,not\ L_n$ if

1) $L_0 \neq \phi : \{L_1,....,L_m\} \subseteq S$ implies $L_0 \in S$
2) $L_0 = \phi : \{L_1,....,L_m\} \not\subset S$

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 PS without not. PS is a program obtained from P as follow:

1) delete each rule that has a *not* L_i in its body with $L_0 \in S$
2) delete all *not* L_i in the bodies of the remaining rules.

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

I. Niemela introduces 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].

4. Logic model for discovering inconsistency

In this section, we present the logic model based on ASP for representing network topology, firewall policy and consistency. The logic model consists of four parts: packet, firewall policy, order and inconsistency discovery.

4.1. 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 network 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 administrators.

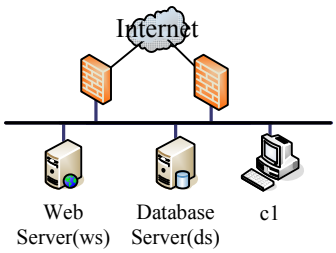

Figure 2 A firewall

Firewall policy consists of different fields that represent information relevant to the filtering. The fields are: **protocol**, **source IP**, **destination IP**, **source port**, **destination port** and **action**.

The predicates used to represent those fields are:

- **protocol**: predicate protocol(X) describes that X 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.
- **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.

Predicate pac(Pro,Sip,Dip,Spt,Dpt) represents a 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.
```

4.2. Firewall policy

In this section, we present logic formulas for representing a firewall policy. Each rule in a firewall policy is of the form <predicate> → <decision>. The <predicate> of a rule is a boolean expression over some packet fields. The <decision>of a rule can be accepted, denied. An example of firewall policies (Cisco style) is:

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

The predicate rule(Num, Act, Pro, Sip, Dip, Spt, Dpt) represents that the Num-th rule say that pac(Pro, Sip, Dip, Spt, Dpt) is dealt with action Act. While comparing two firewall policies a and b, we use predicate a_rule(Num, Act, Pro, Sip, Dip, Spt, Dpt) and b_rule(Num, Act, Pro, Sip, Dip, Spt, Dpt) to represent rules in a and b respectively. The logic program represents the firewall policy mentioned in the above paragraph is:

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

To protect the firewall itself from unauthorized access, it is common to have a stealth rule of the form: From anywhere, to the firewall, with any service, drop[2].

Lemma 1: If there is a stealth rule in a firewall policy, then for every packet pac(Pro, Sip, Dip, Spt, Dpt), there exists at least one rule(Num, Act, Pro, Sip, Dip, Spt, Dpt) which matches the packet.

4.3. Order

Firewall policy is sensitive to policy order. It means that Several rules may match a particular packet. Conflict rules and redundancy rules in a firewall policy should be omitted. Predicate ab(Num, Act, Pro, Sip, Dip, Spt, Dpt) represents that there are a rule(Num, Act, Pro, Sip, Dip, Spt, Dpt) and a rule(Num2, Act2, Pro2, Sip2, Dip2, Spt2, Dpt2), and Num>Num2 ∧ Pro=Pro2 ∧ Sip=Sip2 ∧ Dip=Dip2∧Spt=Spt2∧ Dpt=Dpt2. So, ab(Num, Act, Pro, Sip, Dip, Spt, Dpt) means rule(Num, Act, Pro, Sip, Dip, Spt, Dpt) should be omitted. Predicate access(Num, Act, Pro, Sip, Dip, Spt, Dpt) represents the rule after conflict rules and redundancy rules are omitted. While comparing two firewall policies a and b, predicate a_access(Num, Act, Pro, Sip, Dip, Spt, Dpt) and b_access (Num, Act, Pro, Sip, Dip, Spt, Dpt) represent rules in a and b respectively. The logic program for the omission is:

```
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.
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).
```

Lemma 2: If there is a stealth rule in a firewall policy, then for every packet pac(Pro, Sip, Dip, Spt, Dpt), there exists a unique access(Num, Act, Pro, Sip, Dip, Spt, Dpt) which matches the packet.

4.4. Inconsistencies discovery

Definition 1: For a packet pac(Pro, Sip, Dip, Spt, Dpt), if there are a a_access(Num1,Act1,Pro,Sip, Dip, Spt, Dpt) in a firwall policy a and a b_access(Num2, Act2, Pro, Sip, Dip, Spt, Dpt) in a firwall policy b, and Act1!=Act2, then there is a inconsistency between the Num1-th rule and the Num2-th rule about the packet pac(Pro,Sip,Dip, Spt, Dpt).

We use predicate inconsistency(X, Y, A1, A2, Pro, Sip, Dip, Spt, Dpt) to represent that there is an inconsistency between a access(Num1, Act1, Pro, Sip, Dip, Spt, Dpt) and b access(Num2, Act2, Pro, Sip, Dip, Spt, Dpt). According to the definition 1 and 2, the logic program of inconsistency discovery is:

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

Table 1 Firewall policy designed by team A

num	Protocol	Source IP	Destination IP	Source port	Destination port	Decision
1	TCP	*	ws	*	http	accept
2	*	c1	*	*	*	deny
3	TCP	*	*	*	*	deny

Table 2 Firewall policy designed by team B

num	Protocol	Source IP	Destination IP	Source port	Destination port	Decision
1	*	c1	*	*	*	deny
2	TCP	*	ws	*	http	accept
3	*	*	ws	*	*	deny
4	TCP	*	*	*	*	deny

Table 3 Inconsistencies between the two firewall policies designed by Teams A and B

Protocol	Source IP	Destination IP	Source port	Destination port	Team A(rule number)	Team B(rule number)
TCP	c1	ws	!http	http	accept(1)	deny(1)
TCP	c1	ws	http	http	accept(1)	deny(1)

```
% ip, port and protocol.
ip(ws). ip(c1). ip(other). port(http). port(other). protocol(tcp). protocol(other).
%generate domain of packages
pac(X,Y1,Y2,Z1,Z2) :- protocol(X), ip(Y1), ip(Y2), port(Z1), port(Z2), Y1!=Y2.
%rules of rule set a
a_rule(1,accept,Pro,Sip,Dip,Spt,Dpt) :- pac(Pro,Sip,Dip,Spt,Dpt), Pro==tcp, Dip==ws, Dpt==http.
a_rule(2,deny,Pro,Sip,Dip,Spt,Dpt) :- pac(Pro,Sip,Dip,Spt,Dpt), Sip==c1.
a_rule(3,deny,Pro,Sip,Dip,Spt,Dpt) :- pac(Pro,Sip,Dip,Spt,Dpt).
%rules of rule set b
b_rule(1,deny,Pro,Sip,Dip,Spt,Dpt) :- pac(Pro,Sip,Dip,Spt,Dpt), Sip==c1.
b_rule(2,accept,Pro,Sip,Dip,Spt,Dpt) :- pac(Pro,Sip,Dip,Spt,Dpt), Pro==tcp, Dip==ws, Dpt==http.
b_rule(3,deny,Pro,Sip,Dip,Spt,Dpt) :- pac(Pro,Sip,Dip,Spt,Dpt), Dip==ws.
b_rule(4,deny,Pro,Sip,Dip,Spt,Dpt) :- pac(Pro,Sip,Dip,Spt,Dpt).
% delete priority of rule set a
a_ab(X,A1,Pro,Sip,Dip,Spt,Dpt) :- a_rule(X,A1,Pro,Sip,Dip,Spt,Dpt), a_rule(Y,A2,Pro,Sip,Dip,Spt,Dpt),  Y < X.
a_access(X,A1,Pro,Sip,Dip,Spt,Dpt) :- a_rule(X,A1,Pro,Sip,Dip,Spt,Dpt),  not a_ab(X,A1,Pro,Sip,Dip,Spt,Dpt).
% delete priority of rule set b
b_ab(X,A1,Pro,Sip,Dip,Spt,Dpt) :- b_rule(X,A1,Pro,Sip,Dip,Spt,Dpt), b_rule(Y,A2,Pro,Sip,Dip,Spt,Dpt), Y < X.
b_access(X,A1,Pro,Sip,Dip,Spt,Dpt) :- b_rule(X,A1,Pro,Sip,Dip,Spt,Dpt), not b_ab(X,A1,Pro,Sip,Dip,Spt,Dpt).
%compare
inconsistency(X,Y,A1,A2,Pro,Sip,Dip,Spt,Dpt) :- a_access(X,A1,Pro,Sip,Dip,Spt,Dpt), b_access(Y,A2,Pro,Sip,Dip,Spt,Dpt), A1!=A2.
%display
hide.
show inconsistency(X,Y,A1,A2,Pro,Sip,Dip,Spt,Dpt).
```

Figure 3 The logic program of inconsistencies discover

5. 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 in 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 not access any service in the network.
3) every node in the network can't access each other.

Suppose that we give this specification to two teams Team A and Team B which design the firewall policies, as shown in Tables 1 and 2, respectively. And the inconsistencies between the policies are in Table 3.

The logic program of inconsistencies discovery is Figure 3. The Answer set of the logic program of inconsistencies discovery is：

```
smodels version 2.32. Reading...done
Answer: 1
Stable Model inconsistency(1,1,accept,deny,tcp,c1,ws,oth
er,http) inconsistency(1,1,accept,deny,tcp,c1,ws,http,ht
tp)
True
Duration: 0.63
Number of choice points: 0
Number of wrong choices: 0
Number of atoms: 362
Number of rules: 361
Number of picked atoms: 0
Number of forced atoms: 0
Number of truth assignments: 361
Size of searchspace (removed): 0 (0)
```

6. 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 2: 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 $L_0 \leftarrow L_1,....,L_m,not\ L_{m+1},....,not\ L_n$ in *P* is *local stratified* with respect to *stratum*, if *stratum*(L_i) $\leqslant$ *stratum*(L_0), where1$\leqslant$i$\leqslant$m, and *stratum*(*not* L_j) <*stratum* (L_i), where $m+1 \leqslant i \leqslant n$
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 3[4]: Let P_1, P_2 be two *locally stratified* logic programs. Program $P_1 \cup P_2$ is locally stratified if *head*(P_1)$\cap$*head* (P_2) =Φ

Theorem 1: Logic programs LA, LB, LC, LD in Figure 1 are *locally stratified*.

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 the unique answer which can be computed in polynomial time. So this method will work also in practice. Because of logic language it uses, it is more comprehensible for security administrator.

Acknowledgement

The work was supported by National Natural Science Foundation of China under Grant No. 60573038.

References

[1] A. X. Liu and M. G. Gouda, "Diverse firewall design," IEEE RANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, vol. 19(8), pp. 1–15, 2008.
[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] E. S. Al-Shaer and H. H. Hamed, "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.

Proof: It is obviously that LA, LB, LC are locally stratified according to the definition 2. Because LD is the union of SA, SB and LC, LD is locally stratified according to the lemma 3.

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 theorem 1, we can prove theorem 2.

5. Conclusions

This paper presents a method for automatically discovering inconsistencies between firewall policies Such a method is extremely useful in many ways. For example, it can be used to verify the update of firewall policy by comparing the policy before changes and the policy after changes.