2016 IEEE 40th Annual Computer Software and Applications Conference

Efficient Design of Firewall Temporal Policies

Mahesh Nath Maddumala
Computer Science & Electrical Engineering
University of Missouri - Kansas City
Kansas City, MO, USA
mnmg2d@mail.umkc.edu

Vijay Kumar
Computer Science & Electrical Engineering
University of Missouri - Kansas City
Kansas City, MO, USA
kumarv@umkc.edu

***Abstract*—Firewalls are the first line of defense in cybersecurity. They prevent malicious and unwanted network traffic entering the perimeters of organizations. The strength of a firewall lies in its policy configuration which is also a crucial task for any security administrator. The scope of Firewall policies have been expanding to address ever changing security requirements of an organization. In this process, new security parameters have been researched and one such parameter is temporal policy. Firewall temporal policy is a firewall policy that allows or denies a network packet based on specified day and time range of the policy in addition to the packet filtering rules. Firewall vendors such as CISCO and Palo Alto have already featured firewall temporal policies in their security products. Inclusion of temporal policies in firewall policies results in additional overhead for storing and scanning Firewall policies. As temporal policies are represented in week days and time, they consume considerable amount of space.**

In this paper, we present an innovative and efficient method for representing temporal policies which includes compact representation of temporal policies and detection of anomalies using set operations. Our approach significantly reduces the storage requirement and improves the scanning functionality of firewall. We also present a new method of creating policy sets based on week days.

Keywords-firewall; temporal policies; time-based policies; firewall policies

I. INTRODUCTION

Firewalls filter malicious and unwanted network traffic entering the perimeters of organizations. To do so, the filtering process uses various fields of the IP packet such as protocol type, source IP address, source port, destination IP address, destination port, etc. Firewall method was introduced in late 1980s, since then it has been the most commonly deployed packet filtering technique as the first line of defense. Although with time and with security needs of organizations, significant improvement were introduced, still many important issues remains unresolved. One of such issues is the composition of new firewall policies and their efficient implementation. In this paper, we deal with this issue and present an innovative approach to handle this task.

The objective of our ongoing research is to advance firewall policy specification to reduce the complexity and to give administrator more control over the firewall management. This will significantly reduce the manual effort to keep firewall up to date (insert new and remove redundant policies). In this paper, we report our work on the design issues of firewall temporal policies and present an innovative and efficient method for representing temporal policies. In order to increase the strength of a firewall (accurately stopping unwanted and malicious packets) we have used finer "policy granularity" of temporal parameters such as day and time.

II. OUR CONTRIBUTION

Although firewall is organization-specific, a general framework must be defined so that organizations can generate instances of this framework for their needs. In our effort of building such framework, we have incorporated innovative ideas and parameters such as time and day based filtering (temporal parameters), policy anomaly identification, subset and superset of policies, etc. In this paper we mainly focus on the issue of temporal policies and refer to other issues for maintaining the continuity in our presentation.

In time based filtering, packets can be denied or allowed based on day and time parameters of the policy. We proposed a mathematical approach to optimize the representation of temporal policies. We establish that our way significantly saves space and processing time. We also proposed an optimized way of parsing rules by segregating them into policy sets based on the current day. The final outcome is an efficient model of representing temporal policies and cost-effective firewall management system.

The structure of the paper is as follows. In Section III, we discuss the related work and in section IV, we present our approach of firewall temporal policies and various anomalies caused due to their misconfiguration. In Section V, we present the design of temporal firewall policies starting with their numeric representation followed by their grouping based on week day. In Section VI, we present our implementation details and Section VII presents conclusion of our work.

III. RELATED WORK

Time-based filters are widely in use to control network traffic. Vendors such as CISCO [10] and Palo Alto [11] equipped their firewalls with time based policies.

Temporal policy specification is also available in IP tables [12].

Among available works on firewalls, most have focused on conflict resolution of firewall policies, only a few on temporal policies. Eronen and Zitting [9] designed a constrained logic programming-based system to model firewall policies which deals with conflict resolution only. Bandara *et al* [3] also used logic programming to represent and resolve policy conflicts. Similarly works reported in [4-8], also resolve conflicts based on the fields of TCP/IP protocol. To the best of our knowledge, only the works reported in [1, 2] have used time-based policies in resolving conflicts. They have used BIt-vector based Spatial CALculus (BISCAL) and characterization vectors to detect the conflicts. Although they used time-based approach, the representation of week day's list is not optimized, rather they only focused on detection of conflicts associated with space and time parameters of the policy. In our work, we have introduced (a) an optimized way of representing temporal policies by using numeric approach and (b) used set operations over weekdays for anomaly detection which reduces the processing time to compare list of weekdays.

IV. Our Approach

We propose a numeric-based approach to represent firewall temporal policies. We consider periodic time (week day and time) parameters to specify temporal policies. We used predicate-based logic programming to implement our work.

A. Temporal Policies

Firewall policy is an ordered list of packet filtering rules defining which network packets are allowed or denied based on TCP/IP layers header fields. Firewall temporal policy is a firewall policy that allows or denies a network packet based on specified day and time range of the policy in addition to the packet filtering rules. Time range of a policy is expressed as start and end time in 24 hour military time format and the day field is expressed as a list of days which is a subset of {Mon, Tue, Wed, Thu, Fri, Sat, Sun, Weekdays, Weekends, Anyday}. Weekdays is defined as {Mon, Tue, Wed, Thu, Fri} and Weekends is defined as {Sat, Sun}. Anyday is a list of all the week days. For example, a temporal policy can be specified as, "Block the Facebook on weekdays from 0800 to 1700."

B. Anomalies in Temporal Policies

A temporal policy is said to "match" if the packet arrival day and time falls within the specified day and time range respectively. We explain a match using the example policy "Block the Facebook on weekdays from 0800 to 1700." If a Facebook page request arrives to the firewall of the organization on any of the weekdays during the hours 0800 to 1700, an ordered list of policies is searched to find a match. If there is a match, an associated action is performed on the request, which is deny in this example. As it is practiced, a default policy of "deny everything" is listed at the end of policies.

A typical organization may have several hundred firewall policies. When a packet arrives then the entire ordered list of policies is searched for a match and appropriate decision is taken. In cases when a network packet matches more than one policy then only the first match is considered and the rest of the matches are ignored. This approach often leads to erroneous configuration of policies and violates their consistency.

Anomalies are caused due to the misconfiguration of policies. An anomaly exists if two conflicting outcomes are listed in the ordered list of policies. For example if a Facebook packets arrive between 0800 to 1700 and a search of the policy list finds two rules one says block the packet and the other says allow the packet then a policy anomaly is said to occur. We identify two types of anomalies in temporal policies: *conflict* and *redundant.*

Conflict anomaly between two temporal policies *Px* and *Py* occurs when a packet's arrival day *'d'* and time *'t'* match the day and time range of policies *Px* and *Py*, but their decisions are *different*.

Definition 1 (Conflict anomaly): Two temporal policies *Px* and *Py* are said to *conflict* if $d \in \{Px.days \cap Py.days\}$ *and* $(Px.start\_time \leq t \leq Px.end\_time)$ *and* $(Py.start\_time \leq t \leq Py.end\_time)$ *and* $(Px.action \neq Py.action)$.

This definition says that decision is contradictory for the same policies.

Redundancy between two temporal policies Px and Py occurs when a packet arrival day *'d'* and time *'t'* matches the day and time range of policies Px and Py whose actions are *same*.

Definition 2 (Redundant anomaly): Two temporal policies Px and Py are said to be *redundant* if $d \in \{\text{Px.days} \cap \text{Py.days}\}$ and $(\text{Px.start\_time} \leq t \leq \text{Px.end\_time})$ and $(\text{Py.start\_time} \leq t \leq \text{Py.end\_time})$ and $(\text{Px.action} = \text{Py.action})$.

Table 1 illustrates some sample policies. Suppose when a video streaming (VS) packet arrives on Wednesday at 1300 hours, it matches polices 2 and 3. However, as per first-match rule, action of the policy 2 (deny) is performed on the packet which is different from the policy 3 (allow). In this case policy 2 is said to be in *conflict* with policy 3. Similarly when a Facebook (F) access is requested on Wednesday at 1100 hours, policies 1 and 2 are matched and as actions of the both policies are same, they are *redundant* to each other. Note that source and destination domains are assumed to be same for all the policies and omitted in the sample policies of table 1 (Policy number = P# and Permission = P) as the focus is more on temporal policies. Such occurrences are common in manual management of firewalls.

TABLE I. Sample Policies

P#	Action	Service	Days	Time
1	Deny	VS	(M, W)	0800-1200
2	Deny	VS	Anyday	Any time

3	Allow	VS	(W, F)	1200-1500
4	Deny	F	Weekdays	0800-1200
5	Deny	F	Weekdays	1300-1700

Anomalies in temporal policies can be detected by analyzing the relationship between any two policies with respect to day and time fields. Anomalies between any two policies may occur if any one of the following conditions exists with respect to day and time: *Subset (⊂), Superset (⊃), Equal (=)* and *Overlap (Δ)* and when their packet filtering rules are not disjoint. Suppose policy *Px* precedes policy *Py* and considering the relationships with respect to the day alone, the conditions are explained below using sample policies of Table1.

Subset: P1.days ⊂ P2.days

Superset: P2.days ⊃ P3.days

Equal: P4.days = P5.days

Overlap: P1.days *Δ* P3.days

As the temporal policies are represented in both week day and time, we need to consider all the possible combinations of relationships between every two polices with respect to day and time in order to discover anomalies. Here two policies Px and Py are compared only if Px precedes Py in the policy list. Table 2 presents a complete list of all possible combinations of relationships between two policies and anomalies.

TABLE II. ALL POSSIBLE COMBINATIONS OF RELATIONSHIPS AND ANOMALIES

Day	Time	Action	Anomaly
Subset	Subset	Same	Redundant
Subset	Equal	Same	Redundant
Subset	Superset	Same	No anomaly
Subset	Overlap	Same	No anomaly
Subset	Subset	Different	Conflict
Subset	Equal	Different	Conflict
Subset	Superset	Different	Conflict
Subset	Overlap	Different	Conflict
Equal	Subset	Same	Redundant
Equal	Equal	Same	Redundant
Equal	Superset	Same	Redundant
Equal	Overlap	Same	No anomaly
Equal	Subset	Different	Conflict
Equal	Equal	Different	Conflict
Equal	Superset	Different	Conflict
Equal	Overlap	Different	Conflict
Superset	Subset	Same	No anomaly
Superset	Equal	Same	Redundant
Superset	Superset	Same	Redundant
Superset	Overlap	Same	No anomaly
Superset	Subset	Different	Conflict
Superset	Equal	Different	Conflict
Superset	Superset	Different	Conflict
Superset	Overlap	Different	Conflict
Overlap	Subset	Same	No anomaly
Overlap	Equal	Same	No anomaly
Overlap	Superset	Same	No anomaly
Overlap	Overlap	Same	No anomaly
Overlap	Subset	Different	Conflict
Overlap	Equal	Different	Conflict
Overlap	Superset	Different	Conflict
Overlap	Overlap	Different	Conflict

Table 3 presents a summary of all anomalies. Although redundant is mentioned as an anomaly, it does not violate policy definitions. However, it is still mentioned as anomaly as it increases the number of policies unnecessarily and thus reducing the performance of processing the firewall rules. Here the main concern is the conflict anomaly as it creates the conflict between two policy definitions and hence violates the consistency of the policies.

TABLE III. SUMMARY OF ANOMALIES IN TEMPORAL POLICIES

Day	Time	Action	Anomaly
Any*	Any*	Different	Conflict
Subset	Subset/Equal	Same	Redundant
Equal	Subset/Equal/ Superset	Same	Redundant
Superset	Equal/Superset	Same	Redundant

* “Any” means “any relation except disjoint”.

V. OPTIMIZATION

We optimized the design of temporal firewalls in two phases. In the first phase, we introduce the numeric representation of week days which reduces the storage space used by specification of list of week days and also reduces the time taken in scanning the firewall polices and comparing the list of weekdays to detect anomalies. In second phase, we propose the idea of grouping same day policies into policy sets which results in reduced set of policies. This approach reduces the time taken to match the policies.

A. Numeric representation of temporal policies

To represent days in temporal policies, additional storage space is required. With the list of days’ representation, it consumes significant amount of space and also incurs additional processing time to find a relationship between pair of polices to detect anomalies. To achieve cost-effective representation of days, we introduce the idea of a numeric representation. Here, instead of specifying set of days, a unique numeric value is assigned to every unique subset of week days. The week days are positioned in an order from Monday to Sunday as {Mon, Tue, Wed, Thu, Fri, Sat, Sun}. A binary “1” is assigned to each week day present in the days field of the policy and a binary “0” is assigned to the each week day absent in the days field of the policy. The assignment of binary values 1 and 0 to the week days follows the order from Monday to Sunday to form a 7 bit sequence which is used to calculate the decimal value. This decimal value is used to represent the days field of the policy.

Consider a day field of a policy is {Wednesday, Friday}. A binary 1 is assigned to Wednesday and Friday and a binary zero is assigned to rest of the week days. This generates the representation {0, 0, 1, 0, 1, 0, 0} which forms a 7 bit binary sequence $(0010100)_2$ equivalent of decimal value 20. Table 4 presents some sample conversions from week days to binary form to decimal value.

TABLE IV. NUMERIC REPRESENTATION OF WEEKDAYS

P#	Days	M	T	W	Th	F	S	S	DV*
1	{M, T, W}	1	1	1	0	0	0	0	112
2	{Sat, Sun}	0	0	0	0	0	1	1	3
3	{M, W, F}	1	0	1	0	1	0	0	84
4	{Anyday}	1	1	1	1	1	1	1	127

*DV = Decimal values

Table 5 is a decimal representation of Table 1.

TABLE V. DECIMAL REPRESENTATION OF DAYS FROM TABLE1

P#	Action	Service	Days	Time
1	Deny	Video streaming	80	0800-1200
2	Deny	Video streaming	127	Any time
3	Allow	Video streaming	20	1200-1500
4	Deny	Facebook	124	0800-1200
5	Deny	Facebook	124	1300-1700

One of the important task is to check if a packet arrival day is a member of policy's day set. This can be done by performing bitwise AND operation on specific day and policy's day set. If a packet arrival day is '*d*' and policy's day set is *P.days* then membership function is defined as "*If (d$\wedge$ P.days) is non-zero then d is a member of P.days else d is not a member of P.days",* where '$\wedge$' is a bitwise operation. For example, to check if Wed is a member of policy 1's day's list in the Table 4, we have to perform binary $\wedge$ operation between value of Wed and value of day's list of policy 1. Here value of the Wed is 16 and value of day's list of policy 1 is 112. Bitwise operation $\wedge$ of 16 and 112 yields non-zero value. So Wed is a member of day's list of policy 1.

The possible relationships between each pair of policies with respect to weekdays are: *subset, superset, overlap, equal and disjoint.*

B. Algorithm to find relationship between a pair of policies with respect to days field

As set of week days are represented using decimal value, finding relationships between pair of policies is not straightforward. We have taken the advantage of set operations to find the relationship between two sets of week days.

In the algorithm, *Px.days* and *Py.days* are the decimal values of days set of policies *Px* and *Py* respectively. "AND" is a bitwise AND operation which is an equivalent to set intersection operator.

```
Algorithm Relationship (Px.days, Py.days)
Begin
    If (Px.days == Py.days) then
        Relation = "EQUAL"
Else if (Px.days AND Py.days) == 0 then
        Relation = "DISJOINT"
    Else if (Px.days AND Py.days) == Px.days then
            Relation = "SUBSET"
    Else if (Px.days AND Py.days) == Py.days then
            Relation = "SUPERSET"
    Else
            Relation = "OVERLAP"
    End if
End
```

C. Correctness of the Relationship algorithm

To prove the correctness of the relationship algorithm, we have to prove that the corresponding relationship conditions are correct.

Equal: The condition for equal relation does not need proof as the condition (Px.days == Py.days) is a direct comparison of two decimal values for equivalence.

Disjoint: we need to prove that Px.days and Py.days are disjoint if (Px.days AND Py.days) == 0. Assume that Px.days and Py.days are not disjoint. Then, there should be an element x in both sets Px.days and Py.days. Since x is in both sets Px.days and Py.days, (Px.days AND Py.days) will be not be 0 which is a contradiction to our condition. So, Px.days and Py.days are disjoint.

Subset: we need to prove that Px.days is a subset of Py.days if (Px.days AND Py.days) == Px.days. Assume that Px.days is not a subset of Py.days. Then, there is an element x in Px.days that is not in Py.days. Since x is not in Py.days, (Px.days AND Py.days) will not include x. Thus, (Px.days AND Py.days) $\neq$ Px.days. But our condition is (Px.days AND Py.days) == Px.days, which is a contradiction. So Px.days is a subset of Py.days.

Superset: The proof of superset condition is similar to the subset condition.

Overlap: If all the above conditions are false, then only the left over possible relation is overlap.

Once the relationship is determined, Table 3 is used to detect the possible anomaly and same to be reported.

D. Policy sets based on week day

The design of temporal policies can be optimized by filtering out the policies corresponding to each weekday and group them as a policy set. In this way, we can have a separate policy sets for each weekday. As we have seven weekdays (Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday), we get seven policy sets. Every day, corresponding policy set is chosen and applied to the firewall. This reduces the processing time to process all the firewall policies. With this optimization, only time has to be verified. The firewall system can apply the corresponding policy set every day at time 0000 hrs. and the policy set is valid till the time 2359 hrs. (Time 2359 hours represent the time between 23 hours 59 minutes 00 seconds and 23 hours 59 minutes 59 seconds).

Consider the sample policies from Table 1, policy set for Friday consists of policies P2, P3, P4 and P5, whereas policy set for Saturday consists of only one policy P2. In later case, when a packet arrives on Saturday, only one policy will be scanned, though there are five policies in the policy list.

VI. Implementation

We used prolog based logic programming language ECLiPSe [13] to represent our optimized design of temporal policies. Declarative nature of logic programming makes it easy to specify the temporal policy rules. As logic programs are used to describe relations, it is a better choice to represent and analyze relations of temporal policies.

There are three basic constructs in Prolog: facts, rules, and queries. Facts and rules are used to create knowledge bases. We represented temporal policies as facts. Relations and anomalies are represented as rules and finally we used queries to detect the anomalies.

A temporal policy is represented as a fact as follows, *pTime(policyNo,policyName,days,start_time, end_time, action),* where *policyNo* is policy number in an increasing order, *policyName* is the name of a policy, *days* is the decimal value of week days, *start_time* and *end_time* indicate the time range to apply the policy and finally *action* is a binary decision of allow or deny. For instance, policy 1 in table 5 is represented as *pTime(1, deny_video_1, 80, 800, 1200, deny).* Here *policyName* is not mentioned in the table.

The day relation predicate *dayRel* is used to determine the relation between two policies with respect to week days where Px and Py are the policy names of policy x and policy y, Nx and Ny are the policy numbers, DaysX and DaysY are the days fields. The prolog statements for *dayRel* predicate are as follows,

```
dayRel(equal,Px,Py):- pTime(Nx,Px,DaysX,_,_,_), pTime(Ny,Py,DaysY,_,_,_),Nx < Ny, Px \== Py, ( DaysX =:= DaysY -> true;false).

dayRel(subset,Px,Py):- pTime(Nx,Px,DaysX,_,_,_), pTime(Ny,Py,DaysY,_,_,_), Nx < Ny, Px \== Py, DaysX =\= DaysY, ((DaysX /\ DaysY) =:= DaysX -> true;false).

dayRel(superset,Px,Py):- pTime(Nx,Px,DaysX,_,_,_), pTime(Ny,Py,DaysY,_,_,_), Nx < Ny, Px \== Py, DaysX =\= DaysY, ((DaysX /\ DaysY) =:= DaysY -> true;false).

dayRel(disjoint,Px,Py):- pTime(Nx,Px,DaysX,_,_,_), pTime(Ny,Py,DaysY,_,_,_), Nx < Ny, Px \== Py, ((DaysX /\ DaysY) =:= 0 -> true;false).
```

Time relation predicate *timeRel* is used to determine the relation between two policies with respect to time where StartX and EndX are the starting time and ending time of the policy x and StartY and EndY are the starting time and ending time of policy y. The prolog statements for *timeRel* predicate are as follows,

```
timeRel(equal,Px,Py):- pTime(Nx,Px,_,StartX,EndX,_ ), pTime(Ny,Py,_,StartY,EndY,_), Nx < Ny, Px \== Py, ( (StartX =:= StartY, EndX =:= EndY) -> true;false).

timeRel(subset,Px,Py):- pTime(Nx,Px,_,StartX,EndX, _), pTime(Ny,Py,_,StartY,EndY,_), Nx < Ny, Px \== Py, ( ((StartX >= StartY, EndX < EndY); (StartX > StartY, EndX =< EndY)) -> true;false).

timeRel(superset,Px,Py):- pTime(Nx,Px,_,StartX, EndX,_), pTime(Ny,Py,_,StartY,EndY,_), Nx < Ny, Px \== Py, ( ((StartX =< StartY, EndX > EndY); (StartX < StartY, EndX >= EndY)) -> true;false).

timeRel(disjoint,Px,Py):- pTime(Nx,Px,_,StartX, EndX,_),pTime(Ny,Py,_,StartY,EndY,_), Nx < Ny, Px \== Py, ( (StartX >= EndY ; EndX =< StartY) -> true;false).
```

Anomaly predicate *anomaly* is used to find the redundant and conflict anomalies between every two policies. The prolog statements for *anomaly* predicate are as follows,

```
anomaly(redundant,Px,Py):- dayRel(subset,Px,Py), timeRel(TimeRel,Px,Py), (TimeRel = subset; TimeRel = equal), pTime(_,Px,_,_,_,ActionX), pTime(_,Py,_,_,_, ActionY ), ActionX = ActionY.

anomaly(redundant,Px,Py):- dayRel(equal,Px,Py), timeRel(TimeRel,Px,Py), (TimeRel = subset ; TimeRel =equal;TimeRel=superset),pTime(_,Px,_,_,_,ActionX ),pTime(_,Py,_,_,_, ActionY ), ActionX = ActionY.

anomaly(redundant,Px,Py) :- dayRel(superset,Px,Py), timeRel(TimeRel,Px,Py), (TimeRel = equal ; TimeRel =superset),pTime(_,Px,_,_,_,ActionX),pTime(_,Py,_,_,_, ActionY ), ActionX = ActionY.

anomaly(conflict,Px,Py):- dayRel(DayRel,Px,Py), timeRel(TimeRel,Px,Py), DayRel \= disjoint, TimeRel \=disjoint,pTime(_,Px,_,_,_,ActionX),pTime(_,Py,_,_,_, ActionY ), ActionX \= ActionY.
```

To find out the anomalies, we have to query the above statements with, *findall((X,Px,Py),anomaly(X,Px,Py), Anomalies).*

We have not included the type of service in our implementation as it is assumed to be same or related for all the policies. We have tested our implementation by taking sample temporal policies. The sample policies are chosen in such a way that all the day and time relations are covered as specified in Table 2. Our implementation is able to find all the anomalies associated with the given sample policies.

VII. Conclusion and Future Work

In our work, we proposed an optimal way of representing temporal policies using decimal representation of week days which reduces the space consumed by additional overhead of list of days and time and also reduces the processing time taken to detect the anomalies. We also proposed segregating rules into policy sets which reduces the number of rules to be parsed on a daily basis. This approach significantly improves the scanning of firewall policies. In our present work, we only discussed polices based on time parameter. We are working on its expansion to include location parameter in addition to time parameter in our future work. We are also planning to apply temporal policies in distributed firewall setup.

Acknowledgements

This work is funded by NSF grant (ID CNS-1347958).

References

[1] T. Subana, Y. Tateiwa, Y. Katayama, N. Takahashi, “Simultaneous analysis of time and space for conflict detection in time-based firewall policies,” Proc. of 10th IEEE CIT2010, in press.

[2] T. Subana, Y. Tateiwa, Y. Katayama, N. Takahashi, “An Improved Conflict Detection System with Periodic Cycle Treatment for Time-based Firewall Policies,” Proc. Of 19th IEEE ICCCN 2010, pp.1-8, Zurich, Switzerland, Aug 2010.

[3] Bandara, Arosha K., Antonis Kakas, Emil C. Lupu, and Alessandra Russo. "Using argumentation logic for firewall policy specification and analysis." In Large Scale Management of Distributed Systems, pp. 185-196. Springer Berlin Heidelberg, 2006.

[4] H. Hamed, AI. Shaer, “Taxonomy of Conflicts in Network Security Policies,” IEEE Communication Magazine, vol.44, no.3, pp.134-141, 2006.

[5] B. Zhang, E.AI. Shaer, R. Jagadeesan, J.Riely, C. Pitcher, “Specifications of a high-level conflict-free firewall policy language for multi-domain networks,” SACMAT’07, pp.185-194, Sophia Antipolis, France, June 2007.

[6] L. Yuan, J. Mai, Z. Su, H. Chen, P. Mohapatra, “FIREMAN: a toolkit for firewall modeling and analysis,” Proc. IEEE Symp. Security and Privacy, pp.199-213, Oakland, May 2006.

[7] V. Capretta, B. Stepien, A. Felty and S. Matwin, “Formal correctness of conflict detection for firewalls,” Proc. of ACM workshop on Formal Methods in Security Engineering, Virginia, pp.22-30, USA, Nov 2007.

[8] Hongxin Hu, Gail-Joon Ahn, Ketan Kulkarni, "Detecting and Resolving Firewall Policy Anomalies", IEEE Transactions on Dependable and Secure Computing, vol.9, no. 3, pp. 318-331, May/June 2012.

[9] P. Eronen and J. Zitting, “An expert system for analyzing firewall rules,” Proc. of 6th Nordic Workshop on Secure IT Systems (NordSec 2001), pages 100–107, November 2001.

[10] CISCO, "Creating and Applying Group Policies," [Online]. Available: https://documentation.meraki.com/MX-Z/Group_Policies_and_Blacklisting/Creating_and_Applying_Group_Policies. [Accessed: April 4, 2016].

[11] nrice, "How to Schedule Policy Actions," August, 2015. [Online]. Available: https://live.paloaltonetworks.com/t5/Management-Articles/How-to-Schedule-Policy-Actions/ta-p/56338. [Accessed: April 4, 2016].

[12] H.Eychenne, "iptables(8) - Linux man page," [Online]. Available: http://linux.die.net/man/8/iptables. [Accessed: April 4, 2016].

[13] "The ECLiPSe Constraint Programming System," [Online]. Available: http://eclipseclp.org/. [Accessed: April 4, 2016].