

# Firewall Constraint Analysis for Continuous Compliance

*Susan Hinrichs, shinrich@illinois.edu*  
*Alan Carroll, amc@network-geographics.com*

## 1 Introduction

Firewall and network security appliances enforce much of the network portion of an organization's security policy. The configuration of these devices is challenging, and the mapping between the security policy and the configuration is far from clear.

An organization's security policy is shaped in part by industry standards (e.g., PCI DSS, HIPAA, GLB) and best practices (e.g., NIST 800-41 and IOS 27002). In order to continue operating in good standing within the industry or a geo-political region, the organization must periodically prove operational compliance with some set of these standards and best practices.

For firewalls (and network security appliances), compliance is verified by human review of design and implementation and traffic scanning. Ultimately some form of human review must be performed to verify procedures and verify that documentation matches reality. One commonly used means of automating compliance is traffic scanning using tools such as nmap to send traffic packets from various points in the network to determine where traffic passes. This is a form of black box testing. The scanning technique does not use details of the firewall configuration, so scanning can be performed with any vendor's devices in the network. However, scanning cannot report on the details of how packets are processed within firewalls (e.g., are the packets translated or statefully inspected in addition to being permitted by the firewall). Scanning also requires access to a production network and the ability to insert traffic from multiple points within that network or the resources to duplicate the production network. While scanning is a useful tool that is called out in some compliance requirements, e.g. PCI-DSS, additional tools should be added to the auditor's arsenal to increase compliance automation.

By defining sets of constraints on security policy operations against traffic flows, an organization can use efficient constraint analysis against a description of the firewall's operation (i.e. the configuration) to automate at least part of standards compliance. This paper outlines a model for performing such constraint analysis and describes our implementation in the InfoSector tool set[1].

In Section 2 the paper reviews related work in canonical models and analysis for firewall and router operations, and it describes the time efficient model used by the InfoSector tool set. In Section 3 the paper presents an expression language for expressing traffic flow constraints. In Section 4 the paper presents an example of PCI-DSS constraints expressed in the InfoSector constraint language. This Section also presents results of running these constraints against an example firewall scenario.

## 2 Configuration Modeling

The behavior of the vast majority of modern firewalls is specified by ordered lists of traffic flow specifications. This could be access control lists in classic Cisco configurations, policy lists in Checkpoint or Netscreen devices, or ordered chains in IP Tables scripts. While the syntax differs between devices, each entry in one of these lists identifies a subset of traffic in terms of a tuple of

intervals (source address range, destination address range, protocol range, potentially source port range, potentially destination port range, potentially ICMP message type range). The entry or list then specifies the actions that the security device should perform the specified set of potential packets.

Thus, it is possible to parse configurations and place the key configuration information into a common network security model. Guttman[2] first described a technique for parsing router access control lists (ACLs) and building a packet flow model. Others[3,4] have followed up on this work to parse in network configurations and build higher level packet handling modules.

## **2.1 Related Work**

A number of commercial and research groups have developed network security management tools that present a common security management model and generate the appropriate device-specific configuration to implement the desired model. Cisco Secure Policy Manager [5] and Solsoft [6] both present a global policy model. The user defines his network architecture and identifies where the configurable security enforcing devices are in that architecture and then he defines a global policy of desired traffic flows, tunnels, etc. The management tool then generates the appropriate configurations for the enforcing points. Most security devices have one-to-one embedded web-based GUIs. Some management tools use a variety of multi-assignment techniques to share policy rules to multiple enforcing devices, e.g., Checkpoint, NetScreen, Cisco Firewall MC. While most commercial tools must support generation of configurations for multiple families and versions of security devices for that vendor, Solsoft is unique in generating configurations for multiple vendors. In the research space Bartel, et.al.,[7] created Firmato, a toolkit for creating management tools for different vendors' devices. Guttman and Herzog[8] present NPE as a tool that both generates filtering router configurations and also analyzes existing configurations and suggests how the configurations should be fixed.

Configuration analysis has been pursued by a number of groups as an alternative to direct configuration generation. Some analysis concentrate on identifying "bad" firewall configurations. Al-Shaer and Hamed [9] and Wool [10] have cataloged sets of possible conflicts that arise in today's common linearly ordered access lists. In [11], Yuan, et.al. present in FIREMAN techniques for identifying common errors in firewall configurations. FIREMAN, tools from Al-Shaer and Hamed and a number of commercial products (e.g., Netscreen and Cisco Firewall MC) perform rule conflict analysis which examines access lists or rule lists and presents conflicts within the lists. This identifies rules within the same list that could be matched by the same packet. Some conflicts will occur in most ordered access lists. Generally these are narrow permit rules followed by broader deny rules, but the conflicts are a good place to focus a configuration review. Typos and other development errors can result in unexpected conflicts.

Mayer, Wool, and Ziskind[12] have developed a firewall analyzer that augments the Firmato query engine to generate an exhaustive report of how all possible packets will be handled by a device configured with a specific configuration. This exhaustive report removes any ambiguity from trying to understand how the packets will be processed. This report can be performed proactively without actually configuring the device or passing any traffic. The user can then review the report to understand how any particular packet will be processed. However, due to fragmenting later broader rules around earlier narrow rules, the exhaustive report will be much larger than the original configuration.

Beyond basic misconfigurations, a number of groups have been looking at how to determine whether a firewall implementation meets the requirements of a standard or a policy. Nicol, et. al., have been

examining this from the process control systems (PCS) perspective for access control [13]. FIREWAN[11] will verify that the connectivity to white and black lists of addresses is appropriately enforced. Guttman and Herzon[8] with NPE use information about where a packet can traverse to also specify desired policy.

## 2.2 InfoSecter Canonical Model

With InfoSecter we are also pursuing the problem ensuring that a firewall implementation accurately enforces the organization's security policy and standards. In addition to basic access control, we developed techniques to track other features that can be applied by modern enterprise firewalls. In the following sections, we describe our analysis engine and constraint language.

To create the canonical operational firewall model, we chose to use nested d-dimensional interval trees to model the configuration. This basic technique is described in [14]. Our implementation relies heavily on work in Melhorn[15]. The details of our implementation differ a bit to improve performance constants. The regions in the tree correspond to sets of packets. The regions are decorated with actions that will be performed on specified packets, e.g., permit, translate, inspect. The decorations may also include bookkeeping information like original configuration line or comments from the configuration. We call this tree of decorated packet regions a *flowspace*.

The key insight is that the traffic specification dimensions are ortho-normal. Therefore we can create a flowspace in time  $O(n (\log(n))^d)$ , instead of  $O(n^2)$  where  $n$  is the number of regions and  $d$  is the number of dimensions. After construction the regions can be identified in time  $O(\log(n)^d)$ .

A flowspace type is implemented as an ordered set of *layer* types and a *payload* type. Each layer type models a dimension and has an associated metric type which is the type of values in that dimension (e.g. IPv4 address). The payload type is opaque to the generic implementation and is used to associate client data with a specific regions (e.g. firewall actions). The flowspace type computes a region type which is a d-tuple of pairs of the metric types and the layer types. Each pair stores an interval for a layer and together they describe an ortho-normal region in the flowspace.

One layer is distinguished as the “bottom” layer, and it forms a one dimensional sub-flowspace type that maps from intervals in that dimension to instances of the client data payload type. The second and later layers map from intervals in that dimension to instances of the next lower dimensional flowspace.

Inside each layer, the intervals are stored in a tree of maps. The *outer* tree is keyed by the minimum of intervals in the layer. For each minimum there is an *inner* map keyed by the maximum of the interval with the sub-tree rooted at that node. The inner map stores the payload for that layer, either the region data in the bottom case or the next dimension's tree in the other cases. Figure 1 shows a small example of a two dimensional flowspace with basic integers as dimension values.

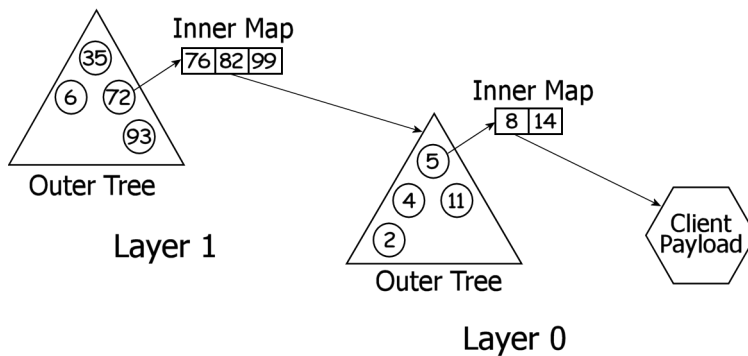


Figure 1: Two levels of a nested interval tree. Highlighting the path encoding (72-82,5-8,X)

Each node in the outer tree also stores the boundary hull for all nodes rooted at that node. That is, it stores the minimum minimum and the maximum maximum for all interval stored in that node and its children. This enables a search to skip the subtree if there is no intersection between the search interval and the sub-tree hull.

Accessing intersecting regions performed is via iterators. The region intersection operation returns a pair of iterators representing a possibly empty range of intervals. At each layer, the intersection operation finds the lowest node that could possibly include the search interval for that dimension, and it finds the highest node that could possibly include the search interval for that dimension. All nodes between these two extremal nodes are possible candidates for intersection. The iterators carry along sufficient state to skip over non-intersecting regions in the candidate set.

InfoSector implements the flowspace in four dimensions: source address, destination address, source service, and destination service. Mapping the source and destination addresses to separate dimensions is natural, but mapping the protocols and ports to dimensions is less obvious. We ended up defining services as a combination of protocol and auxiliary information, e.g. ports for TCP and UDP and message types for ICMP. This means that the protocol information is repeated between the source and destination services, so one could create “impossible” packet sets by specifying a TCP source service and a UDP destination service. This complicates the implementation somewhat, but this duplication of data simplified the overall analysis.

### 2.3 Modeling Layered Features

While the most obvious action of a firewall is to permit or deny packets, these are not the only security actions that can be performed by a modern firewall. Some firewalls can also perform address translation, tunneling, URL filtering, and stateful or deep inspection. In verifying that a security policy is being accurately enforced by a firewall, all the features applied by that firewall must be reviewed.

For some vendors, many features are applied in a single rule or policy entry, e.g. Checkpoint and Netscreen. In that case, verifying one feature is not much more difficult than verifying multiple features. For other vendors, e.g., Cisco and netfilter, different rule lists are used to apply each feature to each set of packets. In this case compliance tools must line up regions that have multiple features applied.

InfoSector builds the flowspace with the primary feature rules first, i.e., the basic packet permit or deny

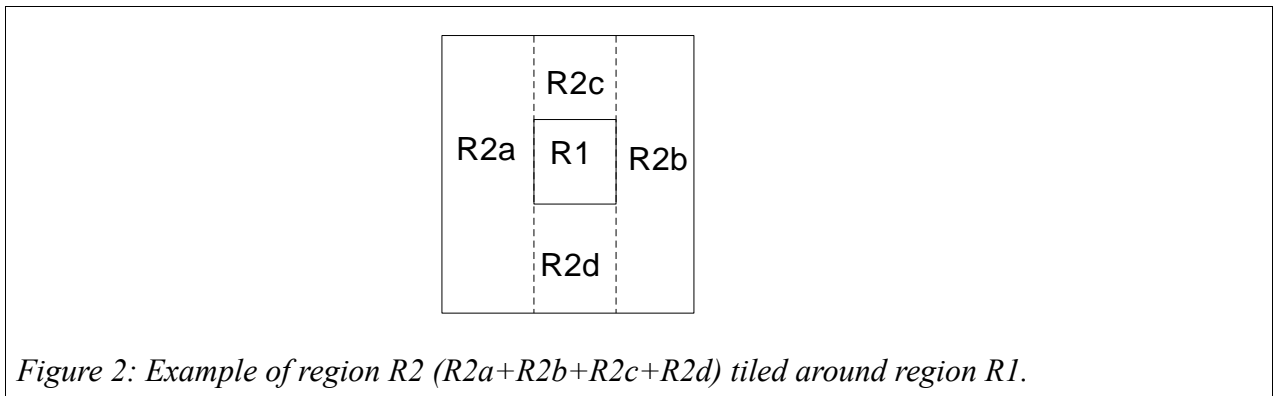
decision. Then the regions of the flowspace are decorated with features from overlapping regions of the other features rule entries.

## 2.4 Flowspace Tiling

As described in section 2.1, one can efficiently add regions into a d-dimensional interval tree. The interval tree itself does not eliminate overlapping regions within the d-dimensional interval tree, but it does make the identification of such overlaps very efficient.

However, much analysis is simplified if it can be performed over a *tiled* space. That is, any portion of the space is covered by one and only one region. For example, given two tiled trees representing two different configuration versions of the same device, computing the change comparison is as straightforward and iterating over the tiles in one tree and checking the actions of conflicting tiles in the other tree.

Tiling can be performed by slicing later regions around earlier regions. A two dimensional example is shown in Figure 2. This tiling or disambiguation can be performed in  $O(n^2)$  time using a naïve algorithm that simply takes each rule region and slices it around all earlier rule regions.



One can create a tiled d-dimensional interval tree more efficiently as follows.

- 1) For each region R in the original rule list (traversed from first to last)
  - 2) Put R in bag B
  - 3) While B is not empty
    - 4) Pull b from B
    - 5) If there is an area C in T that conflicts with b
      - 6) Split b around hull(C). Store the remainders in bag B
    - 7) else
      - 8) insert b into T
      - 9) set hull(b) = R

This procedure will eventually stop because for each rule, the region R is subdivided until it disappears completely or conflicts with no previous rules and can be inserted into T. This will create a new tree T with  $n'$  regions.  $n' \leq n * S * C$ , where S is the number of sub regions each region can be split into and C is the number of conflicts for each original rule.

If  $n'$  is more than a constant factor larger than  $n$ , this means that the effective lookup time in the new tree is far worse as a factor of the original  $n$ . First let us examine a bound on  $S$ .

Claim: Splitting a  $d$ -dimensional region  $R1$  around another  $d$ -dimension region  $R2$  will result in at most  $2d$  remainder regions.

Proof by induction:

For  $d=1$ , worst case is if  $\min(R1[1]) < \min(R2[1])$  and  $\max(R1[1]) > \max(R2[1])$ . In this case, there are two remainder lines  $[\min(R1[1]), \min(R2[1])-1]$  and  $[\max(R2[1])+1, \max(R1[1])]$ .

For  $d=k$ , the number of regions created by splitting the  $d-1$  dimensions of  $R1$  and  $R2$  in the worst case is  $2(d-1)$ . Add the final dimension. In the worst case, the  $\min(R1[d]) < \min(R2[d])$  and  $\max(R1[d]) > \max(R2[d])$ . For each  $d-1$  dimensional remainder subregions, we need to create two more remainder regions. One with the  $d$ 'th dimension set to  $[\min(R1[d]), \min(R2[d])-1]$  and the other with the  $d$ 'th dimension set to  $[\max(R2[d])+1, \max(R1[d])]$ .

Therefore  $S$  has a bound of  $2d$ . Unfortunately,  $C$  does not have a nice bound. There exist cases where the average number of conflicts for each rule is  $O(n)$ . This gives us a  $n'$  bounded by  $O(2d*n*n)$ , which means that  $n^2$  comes into our lookup and construction bounds for the tiled tree.

A  $O(n)$  checkerboard or basket weave conflict case is shown in Figure 3. We have not encountered such cases in practice. Most firewall configurations follow an exceptions then general rule pattern as shown in Figure 4. While the general region has many subregions conflicting with it, the subregions only conflict with the general region. In these exceptions then general patterns, the average conflict size  $C$  is a constant.

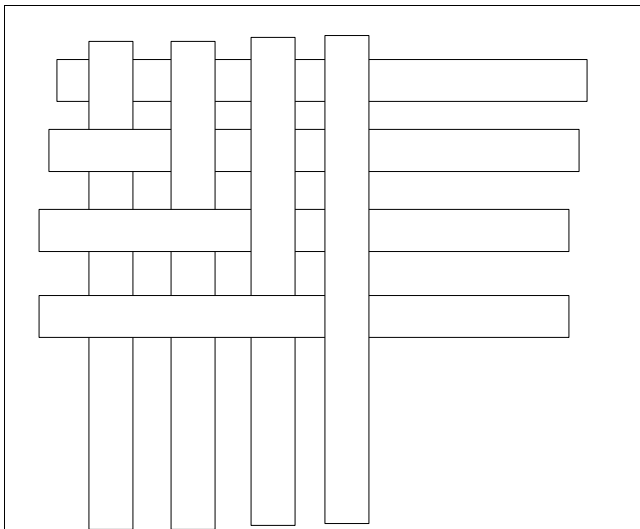


Figure 3: Example of a 2D flowspace showing a basket weave pattern. Each region conflicts with  $O(n)$  other regions.

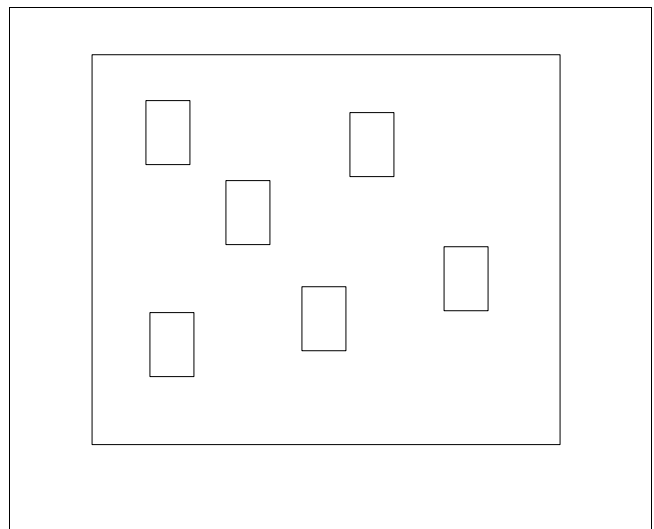


Figure 4: Example of a exceptions then general 2D flowspace. The average number of conflicts is constant.

### 3 Security Traffic Constraints

One can specify constraints in terms of basic expressions. For InfoSector we define two expression clause types: *Traffic* types and *Action* types. The traffic type clauses are combined to identify a set of

packets. We use the notation “attribute ^ value” to indicate that attribute should include the specified value. The Traffic type attributes include:

- Source or destination address – A set of IP addresses (single address, range, or network) that specify the source or destination addresses for the traffic under consideration. e.g., Source Address ^ 192.168.1.0/24
- Source or destination service – The set of services that specify the source and destination components of the protocols and auxiliary information for the traffic under consideration. e.g., Destination Service ^ TCP:80
- Protocol – The set of protocols for the traffic under consideration. e.g., Protocol ^ ICMP
- Scope – The pair of interfaces or zones that the traffic under consideration traverses. e.g., Scope ^ traffic crossing from Trust to Untrust

Once the traffic is specified, the action type is used to indicate how the specified traffic should be processed. The action type values include:

- Permit or Deny – The device will forward or drop the specified traffic.
- Translation – The device will perform address translation on the specified traffic.
- Tunnel – The device will tunnel the specified traffic. This action should have additional attributes to indicate the characteristics of the desired tunnel.
- Inspect – The device will perform deep packet inspection on the specified traffic.
- URL Filter – The device will pass the specified traffic through a URL filter.

These primitive expression clauses can be combined using the following operators: NOT, AND, OR, and OTHERWISE. These operators are expressed using the following notation: ! for NOT, & for AND, | for OR, and \* for OTHERWISE.

By combining traffic expressions with the AND operator, the expression narrows the set of packets under consideration. By AND'ing in an action clause, the expression identifies a set of desired actions for the traffic specified by the traffic clauses. For example, consider the following expression:

```
Source Address ^ 192.168.1.0/24 & Destination Service ^ TCP:80 &  
Action ^ Translation
```

This expression has two type clauses which indicate that the traffic of interest is the set of TCP packets with a source in the 192.168.1.0/24 network to any destination address from any source port to destination port 80. For these packets, the firewall should perform address translation.

Similarly OR'ing elements creates multiple sets of packets. Our previous example can be expanded as follows.

```
(Source Address ^ 192.168.1.0/24 | Source Address ^ 10.10.10.10) &  
Destination Service ^ TCP:80 & Action ^ Translation
```

This gives us two sets of packets, one with a source address in the 192.168.1.0/24 network and the other from 10.10.10.10. In both cases, the device should apply address translation to the specified packets.

In many cases, policies and standards enumerate several sets of cases where specific actions should be

applied, and then for all other packets there is a default action (generally a deny in a security scenario). One may initially think of using a final OR clause to express the default action, e.g.,

```
(Source Address ^ 192.168.1.0/24 & Action ^ Permit) | (Action ^ Deny)
```

However, this is not accurate. In the example above, the packets with a source address in the 192.168.1.0/24 network will also match the second clause.

This could be accurately expressed with only AND, OR and NOT by adding in the appropriate NOT clauses, .e.g,

```
(Source Address ^ 192.168.1.0/24 & Action ^ Permit) |  
(! (Source Address ^ 192.168.1.0/24) & Action ^ Deny)
```

However, for sizable expressions this solution is cumbersome and difficult for the end user to create and understand. We could apply ordering and special cases to the clauses. This again complicates the constraint language and makes it more difficult to understand what will happen for expressions combined outside of a disjunctive normal form.

Instead, we add a new operator called OTHERWISE.

```
Previous cases OTHERWISE otherwise case
```

The otherwise case includes an action which specifies how traffic not specified in the “Previous cases” should be processed by the configuration. The previous cases are at the same nesting level as the OTHERWISE, so you could have nested other OTHERWISE operators, e.g.,

```
(Source Address ^ 192.168.1.0/24 &  
((Destination Service ^ TCP:SSH & Action ^ Permit) * (Action ^ Deny)))
```

In this case, the otherwise only applies to traffic to other services and destinations from the source network 192.168.1.0/24. In our previous example we have,

```
(Source Address ^ 192.168.1.0/24 & Action ^ Permit) * (Action ^ Deny)
```

For any expression, the OTHERWISE clauses can be translated into normal NOT, AND and OR, but for the human editor and reader, the expression that includes the OTHERWISE operator is far simpler and easier to understand.

### **3.1 Verifying Constraint Expressions**

Using these constraint expressions, the user can specify which actions should be applied to sets of packets. InfoSector places no restrictions on the nesting of AND, OR, and NOT operator. However, to simplify constraint analysis InfoSector first translates the expression to disjunctive normal form before processing. It also replaces the OTHERWISE operators with an equivalent expression that uses only AND, OR, and NOT operators.

After the expression is normalized, each conjunctive clause represents a set of packets and a desired operation. There may be overlap between the conjunctive clauses. Therefore, we create a d-dimensional interval tree and encode the constraint areas in that tree. Then we tile the tree to eliminate duplicate regions. The algorithm can report on overlap constraint regions at this point if it was of interest to the end user.

Given a configuration, we can build a tiled d-dimensional interval tree T that encodes how that configuration will process packets. Given a constraint expression, we can build a tiled d-dimensional

interval tree C that encodes the desired actions for each specified packet region. Given these two trees, we can execute the following algorithm to determine whether the configuration meets the requirements of the constraint expression.

- 1) For each constraint region c in C
  - 2) For each region r that conflicts with c in T
    - 3) If action(r) is not consistent with action(c)
      - 4) report constraint mismatch

## 4 PCI-DSS Example Constraint Analysis

The PCI Data Security Standards (PCI-DSS) [16] have emerged as a set of concrete compliance requirements for organizations handling payment cardholders information. The PCI-DSS covers all aspects of the computer system as it applies to cardholder information including networking, disk, and application security. In particular, requirement 1 “Install and maintain a firewall configuration to protect card holder data” directly applies constraints to the design and implementation of an organization's firewalls. A number of these requirements can be completely or partially verified by evaluating constraints against the firewall configurations. We examine some of those requirements in this section.

### 4.1 Requirement 1.1.5b

This requirement covers the use of insecure network protocols. That is, protocols that can pass sensitive information like passwords in the clear such as ftp and telnet. Specifically, the requirement is stated as:

Identify insecure services, protocols, and ports allowed; and verify they are necessary and that security features are documented and implemented by examining firewall and router configuration standards and settings for each service.

The auditor will want to identify the use of any commonly used insecure protocols, and determine whether the organization fully understands the impact of the use of these protocols and has adding the appropriate mitigating controls into their design.

The identification of the use of these insecure protocols can be expressed as a constraint applied to traffic allowed into the organization from the Internet and out of the organization to the Internet as follows:

```
(Scope ^ $Inbound | Scope ^ $Outbound) &  
Destination Service ^ $InsecureServices & Action ^ Deny
```

Tokens that begin with the \$ character are macros. They can be bound to specific values that make sense for a specific environment. E.g., \$InsecureServices may be bound to TCP:23 and TCP:21 (telnet and ftp). This expression states that any traffic entering the firewall from the Internet or leaving the firewall to the Internet should be denied if the service is telnet or ftp.

### 4.2 Requirements 1.3.2 and 1.3.5

These requirements both express a limit on addressability within the network architecture.

Requirement 1.3.2 states:

Verify that inbound Internet traffic is limited to IP addresses within the DMZ.

Requirement 1.3.5 is similar but applies to outbound traffic:

Verify that outbound traffic from the cardholder data environment to the Internet can only access IP addresses within the DMZ.

The requirement is not stating that all traffic to the DMZ network should be allowed, but rather it is stating if the traffic is not destined to the DMZ network it should not be allowed. This can be expressed in the following constraints. For 1.3.2:

```
(Scope ^ $Inbound) &  
  ((Destination Address ^ $DMZNetwork) * (Action ^ Deny))
```

or

```
(Scope ^ $Inbound & !(Destination Address ^ $DMZNetwork) & Action ^ Deny)
```

For 1.3.5:

```
(Scope ^ $Outbound & Destination Address ^ $DMZNetwork) * (Action ^ Deny)
```

or

```
(Scope ^ $Outbound & !(Destination Address ^ $DMZNetwork) & Action ^ Deny)
```

### **4.3 Requirement 1.3.8**

Requirement 1.3.8 specifies how far addresses can be distributed. Specifically this requirement states:

For the sample of firewall and router components, verify that NAT or other technology using RFC 1918 address space is used to restrict broadcast of IP addresses from the internal network to the Internet (IP masquerading).

Once you understand which firewall(s) is implementing the address translation, you can apply the following constraint:

```
(Scope ^ $Outbound & (Action ^ Translate | Action ^ Deny))
```

This constraint check will catch packets that are allowed but not translated. The constraint analysis will need to be followed up by periodic review of the specific translation rules to ensure they are in fact hiding the details of the internal network addressing.

### **4.4 Tests**

We verified our constraints on the basic n-tier network architecture shown in Figure 5. The configuration of the Internet facing firewall is in Appendix A.

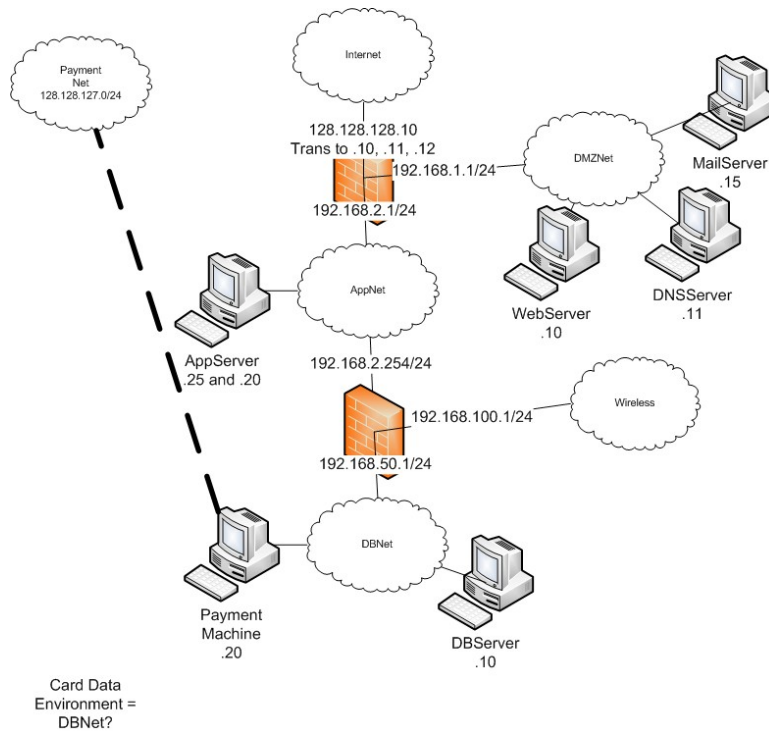


Figure 5: Test PCI environment network architecture.

We applied all the constraints outlined in the previous sections. The constraint report results are highlighted in the table below.

Constraint	Mismatch Areas	Due to Lines	Comment
1.1.5b	Scope = Cross outsidexdmz Source Address = ANY Destination Address = 192.168.1.15 Destination Service = TCP:telnet	40,66	Access to telnet server from Internet to DMZ
1.3.2	Scope = Cross outsidexinside Source Address = ANY Destination Address = 192.168.2.10 Destination Service = TCP:ssh	42, 67	Firewall and translation rules allow access to SSH server on the application network.
1.3.5	None		Only traffic from the application network to the DMZ servers is allowed.
1.3.8	None		For PIX 6.3 translation is required to enable access in addition to the firewall rule.

The example firewall configuration is very short to make this example understandable. The constraint analysis ran instantaneously from the human perspective. Our algorithms do scale well. We have run

this analysis over very large configurations (over 10,000 lines) in the matter of a few minutes.

#### **4.5 Benefits and Limitations**

Constraint analysis over firewall configurations can be quite useful to continually ensure that the firewall implementations meet the requirements of standards such as PCI-DSS. Once set up, the constraints can be automatically applied via scripts to every change to the firewall configuration. Depending on the organization's change control model, the constraint results can be included in the configuration change review, or alternatively the constraint results can be emailed to a managing system administrator only if there are constraint mismatches. Since the constraint analysis does not require injection of traffic into a production network, it can be performed more frequently with little human resource overhead, and the analysis can be performed even before the configuration changes are deployed.

While the constraint analysis can be quite useful, it cannot completely replace direct human review or runtime traffic scanning. Certain requirements cannot be automatically verified, e.g., Requirement 1.1.1 “Verify that there is a formal process for testing and approval of all network connections and changes to firewall and router configurations.”

While traffic scanning and constraint analysis both check on the behavior of traffic flowing through the network, the traffic scanning is sending real traffic and not just simulating traffic. Traffic scanning should still be performed periodically to ensure that the constraint analysis engine does not have errors in its model, or that the analysis engine is being fed incorrect information. Testing using multiple techniques ensures more accurate results.

### **5 Summary and Moving Forward**

Firewall Configuration analysis can be a useful additional tool for policy and standards compliance verification. Since configuration analysis can be performed with little computation overhead and little human time overhead, the configuration analysis can be performed more often than some of the other more resource intensive compliance verification techniques such as log analysis, human review, or traffic scanning. By performing some form of compliance verification on every configuration change, an organization is more likely to catch compliance errors sooner, potentially even before the faulty changes are deployed.

With the expression language presented in this paper, one can create formal constraints on how traffic should be processed based on standards such as PCI-DSS and NIST-800-41. Different stakeholders in an organization may have different policies they need enforced in a central firewall. Each stakeholder can write their own constraint expressions, and the constraints can be set up to run against every change of configuration for that firewall.

Firewalls and other network security appliances are evolving. Many features are being specified by attributes beyond the basic address, protocol, and port 5-tuple. For example, current enterprise firewalls enable the specification of HTTP application-level attributes to determine whether the HTTP traffic should have deep inspection applied or not. Efficiently modeling specification of traffic at the application level for constraint analysis is an area that needs to be explored.

The current version of InfoSector only analyzes one firewall configuration at a time. The flowspaces should be composable across a path of firewalls. Given network topology, one should be able to

generate flowspace for all interesting paths as a means of doing a system-wide constraint analysis.

Routing integration is another area for future work. The current InfoSector implementation uses static routing information to determine how packets can flow between interfaces. If dynamic routing is used, the InfoSector analysis will make overly broad assumptions about how traffic can flow between interfaces. To be more accurate, the constraint analysis should include current routing tables.

Potentially the analysis should return different results depending on whether the packets were explicitly denied or simply not routed through that interface due to the current routing tables.

Constraint and configuration analysis is a rich area to improve the policy and standards compliance for an organization. The constraint analysis as outlined in this paper can be used to catch errors sooner and to focus the human auditors efforts to the areas of real problems.

## References

1. *InfoSector*, <http://network-geographics.com/infosector>.
2. J. D. Guttman, "Filtering Postures: local enforcement for global policies", In *IEEE Security and Privacy Symposium*, Oakland, CA, 1997.
3. Ehab Al-Shaer and Hazem Hamed, "Modeling and Management of Firewall Policies". In *IEEE Transactions on Network and System Management*, Volume 1-1, April 2004.
4. A. Mayer, A. Wool, and E. Ziskind. "Offline firewall analysis". *International Journal of Information Security*, 5(3):125-144, 2006.
5. S. Hinrichs, "Policy-Based Management: Bridging the Gap". In *Proc. of the 15th Annual Security Applications Conference*, IEEE, Phoenix, AZ, December, 1999.
6. *Solsoft Policy Server*, <http://www.exaprotect.com/products/solsoft.php>.
7. Y. Bartal, A. Mayer, K. Nissim, and A. Wool. "Firmato: A novel firewall management toolkit". *ACM Transactions on Computer Systems*, 22(4):381-420, November 2004.
8. J. D. Guttman and A. L. Herzog, "Rigorous automated network security management". *International Journal of Information Security*, 4(1-2), 2005.
9. Ehab Al-Shaer and Hazem Hamed, "[Taxonomy of Conflicts in Network Security Policies](#)", *IEEE Communications Magazine*, Vol. 44, No. 3, March 2006
10. A. Wool. "[A quantitative study of firewall configuration errors.](#)" *IEEE Computer*, 37(6):62-67, 2004
11. L. Yuan, J. Mai, Z. Su, H. Chen, C-N. Chuah, and P. Mohapatra, "FIREMAN: A Toolkit for Firewall Modeling and Analysis," *IEEE Symposium on Security and Privacy*, May 2006
12. A. Mayer, A. Wool, and E. Ziskind. "Offline firewall analysis". *International Journal of Information Security*, 5(3):125-144, 2006.
13. D. M. Nicol, W. H. Sanders, S. Singh, M. Seri, "Usable Global Network Access Policy for Process Control Systems," *IEEE Security and Privacy*, vol. 6, no. 6, pp. 30-36, Nov./Dec. 2008, doi:10.1109/MSP.2008.159
14. J. Qian, S. Hinrichs, K. Nahrstedt, "ALCA: A Framework for Access Control List (ACL) Analysis and Optimization". In *Proceedings of Communications and Multimedia Security*, 2001.
15. K. Mehlhorn, 1984 *Data Structures and Algorithms 3: Multi-Dimensional Searching and Computational Geometry*. Springer-Verlag New York, Inc.
16. Payment Card Industry (PCI) Data Security Standards Requirements and Security Assessment Procedures version 1.2. October, 2008. [https://www.pcisecuritystandards.org/security\\_standards/pci\\_dss.shtml](https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml)

## Appendix A – PCI Firewall Configuration

```
1. PIX Version 6.3(2)
2.
3.
4. route outside 0.0.0.0 0.0.0.0 128.128.128.50 1
5. route inside 192.168.10.0 255.255.255.128 192.168.2.50 1
6. route inside 192.168.100.0 255.255.255.128 192.168.2.254 1
7. route inside 192.168.50.0 255.255.255.128 192.168.2.254 1
8.
9. hostname outer-fw
10.     domain-name example.com
11.     fixup protocol http 80
12.     fixup protocol smtp 25
13.
14.     names
15.     name 192.168.1.11 DNS-server
16.     name 192.168.1.15 Mail-server
17.     name 192.168.1.10 Web-server
18.     name 128.128.128.12 Ext-server
19.
20.     nameif eth1 inside 100
21.     nameif eth2 dmz 50
22.     nameif eth0 outside 0
23.
24.     ip address inside 192.168.2.1 255.255.255.0
25.     ip address outside 128.128.128.10 255.255.255.0
26.     ip address dmz 192.168.1.1 255.255.255.0
27.
28.     global (outside) 1 128.128.128.11
29.     global (outside) 1 interface
30.
31.     nat (inside) 1 192.168.2.0 255.255.255.0 dns
32.     nat (inside) 1 192.168.50.0 255.255.255.0 dns
33.     nat (inside) 1 192.168.100.0 255.255.255.0 dns
34.     : nat (dmz) 1 192.168.1.0 255.255.255.0 dns
35.
36.     static (dmz,outside) tcp Ext-server 80 Web-server 80 netmask
37.     255.255.255.255
38.     static (dmz,outside) tcp Ext-server 443 Web-server 443 netmask
39.     255.255.255.255
40.     static (dmz,outside) udp Ext-server 53 DNS-server 53 netmask
41.     255.255.255.255
42.     static (dmz,outside) tcp Ext-server 25 Mail-server 25 netmask
43.     255.255.255.255
44.     static (dmz,outside) tcp Ext-server telnet Mail-server telnet netmask
45.     255.255.255.255
46.     static (inside,outside) tcp 128.128.128.13 22 192.168.2.10 22 netmask
47.     255.255.255.255
48.
49.     static (inside,dmz) 192.168.2.0 192.168.2.0 netmask 255.255.255.0
50.
51.     object-group service web-services tcp
52.     port-object eq web
53.     port-object eq https
```

```

50.    object-group service inbound-svcs tcp
51.        port-object eq smtp
52.        port-object eq telnet
53.        port-object eq ssh
54.        group-object web-services
55.
56.    object-group service inbound-udp-svcs tcp
57.        port-object eq domain
58.
59.    object-group network internal
60.        description The networks behind the dmz
61.        network-object 192.168.2.0 255.255.255.0
62.        network-object 192.168.10.0 255.255.255.0
63.        network-object 192.168.50.0 255.255.255.0
64.        network-object 192.168.100.0 255.255.255.0
65.
66.    access-list inbound permit tcp any host Ext-server object-group
inbound-svcs
67.    access-list inbound permit tcp any host 128.128.128.13 eq ssh
68.    access-list inbound permit udp any host Ext-server object-group
inbound-udp-svcs
69.
70.    access-list outbound permit udp object-group internal host DNS-server
eq domain
71.    access-list outbound permit tcp object-group internal host Mail-server
eq smtp
72.    access-list outbound permit tcp object-group internal host Web-server
object-group web-services
73.
74.    access-list dmz-out-list permit udp host DNS-server any eq domain
75.    access-list dmz-out-list permit tcp host Mail-server any eq smtp
76.
77.    access-group outbound in interface inside
78.    access-group inbound in interface outside
79.    access-group dmz-out-list in interface dmz

```