Formal Analysis of Firewalls

Robert Marmorstein

Dissertation Committee
Advisor: Dr. Phil Kearns
Dr. Weizhen Mao
Dr. David Coppit
Dr. Haining Wang
Dr. Jean Mayo

April 10, 2008
Some useful definitions

**Firewall**
A *firewall* is a software or hardware facility for implementing a security policy on the packets that enter a network.

**Firewall Policy**
A *firewall policy* is a set of rules that determines whether a packet should be allowed to enter the network (be accepted) or blocked from entering the network (be dropped).

**Firewall Chain**
On Linux, the firewall policy is organized into *chains* of rules. A *firewall chain* is a sequentially ordered list of rules which are grouped together into a logical unit.
Why do we need Formal Analysis?

Chain INPUT (policy DROP 373K packets, 41M bytes)

<table>
<thead>
<tr>
<th>pkts</th>
<th>bytes</th>
<th>target</th>
<th>prot</th>
<th>opt</th>
<th>in</th>
<th>out</th>
<th>source</th>
<th>destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>740K</td>
<td>294M</td>
<td>external_packets</td>
<td>0</td>
<td>--</td>
<td>eth2</td>
<td>*</td>
<td>0.0.0.0/0</td>
<td>0.0.0.0/0</td>
</tr>
<tr>
<td>190K</td>
<td>79M</td>
<td>internal_packets</td>
<td>0</td>
<td>--</td>
<td>eth1</td>
<td>*</td>
<td>0.0.0.0/0</td>
<td>0.0.0.0/0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>internal_packets</td>
<td>0</td>
<td>--</td>
<td>eth0</td>
<td>*</td>
<td>0.0.0.0/0</td>
<td>0.0.0.0/0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>ACCEPT</td>
<td>0</td>
<td>--</td>
<td>*</td>
<td>*</td>
<td>0.0.0.0/0</td>
<td>127.0.0.1</td>
</tr>
</tbody>
</table>

Chain FORWARD (policy DROP 0 packets, 0 bytes)

<table>
<thead>
<tr>
<th>pkts</th>
<th>bytes</th>
<th>target</th>
<th>prot</th>
<th>opt</th>
<th>in</th>
<th>out</th>
<th>source</th>
<th>destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>108</td>
<td>7296</td>
<td>DROP</td>
<td>icmp</td>
<td>--</td>
<td>eth1</td>
<td>*</td>
<td>0.0.0.0/0</td>
<td>icmp type 8</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>DROP</td>
<td>icmp</td>
<td>--</td>
<td>eth0</td>
<td>*</td>
<td>0.0.0.0/0</td>
<td>icmp type 8</td>
</tr>
<tr>
<td>709K</td>
<td>499M</td>
<td>ACCEPT</td>
<td>tcp</td>
<td>--</td>
<td>*</td>
<td>*</td>
<td>0.0.0.0/0</td>
<td>192.168.0.0/22 tcp dpt:22</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>ACCEPT</td>
<td>tcp</td>
<td>--</td>
<td>*</td>
<td>*</td>
<td>0.0.0.0/0</td>
<td>192.168.0.0/22 tcp dpt:80</td>
</tr>
<tr>
<td>936</td>
<td>47842</td>
<td>ACCEPT</td>
<td>tcp</td>
<td>--</td>
<td>*</td>
<td>*</td>
<td>0.0.0.0/0</td>
<td>192.168.0.0/22 tcp dpt:8080</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>ACCEPT</td>
<td>tcp</td>
<td>--</td>
<td>*</td>
<td>*</td>
<td>101.92.26.68</td>
<td>192.168.5.15 tcp dpt:3306</td>
</tr>
<tr>
<td>28M</td>
<td>38G</td>
<td>ACCEPT</td>
<td>tcp</td>
<td>--</td>
<td>eth1</td>
<td>*</td>
<td>192.168.0.0/22</td>
<td>tcp dpt:22</td>
</tr>
<tr>
<td>9194</td>
<td>1849K</td>
<td>ACCEPT</td>
<td>udp</td>
<td>--</td>
<td>eth1</td>
<td>*</td>
<td>192.168.0.0/22</td>
<td>udp dpt:53</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>ACCEPT</td>
<td>tcp</td>
<td>--</td>
<td>eth0</td>
<td>*</td>
<td>192.168.0.0/22</td>
<td>tcp dpt:22</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>ACCEPT</td>
<td>udp</td>
<td>--</td>
<td>eth0</td>
<td>*</td>
<td>192.168.0.0/22</td>
<td>udp dpt:53</td>
</tr>
<tr>
<td>52M</td>
<td>26G</td>
<td>ACCEPT</td>
<td>0</td>
<td>--</td>
<td>*</td>
<td>*</td>
<td>0.0.0.0/0</td>
<td>state RELATED,ESTABLISHED</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>REJECT</td>
<td>0</td>
<td>--</td>
<td>eth1</td>
<td>*</td>
<td>192.168.5.121</td>
<td>reject-with icmp-port-unreachable</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>REJECT</td>
<td>0</td>
<td>--</td>
<td>eth1</td>
<td>*</td>
<td>192.168.5.122</td>
<td>reject-with icmp-port-unreachable</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>REJECT</td>
<td>0</td>
<td>--</td>
<td>eth1</td>
<td>*</td>
<td>192.168.5.123</td>
<td>reject-with icmp-port-unreachable</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>REJECT</td>
<td>0</td>
<td>--</td>
<td>eth1</td>
<td>*</td>
<td>192.168.5.124</td>
<td>reject-with icmp-port-unreachable</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>REJECT</td>
<td>0</td>
<td>--</td>
<td>eth1</td>
<td>*</td>
<td>192.168.5.125</td>
<td>reject-with icmp-port-unreachable</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>REJECT</td>
<td>0</td>
<td>--</td>
<td>eth1</td>
<td>*</td>
<td>192.168.5.126</td>
<td>reject-with icmp-port-unreachable</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>REJECT</td>
<td>0</td>
<td>--</td>
<td>eth1</td>
<td>*</td>
<td>192.168.5.101</td>
<td>reject-with icmp-port-unreachable</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>REJECT</td>
<td>0</td>
<td>--</td>
<td>eth1</td>
<td>*</td>
<td>192.168.5.101</td>
<td>reject-with icmp-port-unreachable</td>
</tr>
</tbody>
</table>
Why do we need Formal Analysis?

Reasons for Analysis

- Firewall policies are complex
- Firewall policies are dynamic
- Firewall policies are critical

Results of Analysis

- Develop a better understanding of the policy
- Discover and repair errors in the policy
- Gain assurance in the policy
Firewall policy errors

Topology of a large network
Firewall policy errors

<table>
<thead>
<tr>
<th>target</th>
<th>prot</th>
<th>source</th>
<th>destination</th>
<th>flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCEPT</td>
<td>all</td>
<td>113.137.10.0/24</td>
<td>113.137.10.4</td>
<td></td>
</tr>
<tr>
<td>DROP</td>
<td>all</td>
<td>anywhere</td>
<td>113.137.10.4</td>
<td></td>
</tr>
</tbody>
</table>
Firewall policy errors

<table>
<thead>
<tr>
<th>target</th>
<th>prot</th>
<th>source</th>
<th>destination</th>
<th>flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ACCEPT</td>
<td>tcp</td>
<td>192.168.1.0/24</td>
<td>113.137.10.0/24</td>
<td>tcp dpt:22</td>
</tr>
<tr>
<td>2 ACCEPT</td>
<td>all</td>
<td>113.137.10.0/24</td>
<td>113.137.10.4</td>
<td></td>
</tr>
<tr>
<td>3 DROP</td>
<td>all</td>
<td>anywhere</td>
<td>113.137.10.4</td>
<td></td>
</tr>
</tbody>
</table>
### Existing Work

#### Active Testing tools
- Port Scanners (nmap, hping2)
- Vulnerability Scanners (nessus, SATAN, ISS)
- Ftester

#### Passive Testing tools
- FANG/Lumeta/Algosec
- Redseal
- ITVal

#### Other tools
- Hazelhurst BDD model, Gounda/Liu Firewall MDDs
- Expert Systems, Model Checking Systems
- Graph Algorithms (Lumeta, SUNY project)
Multiway Decision Diagrams

A Multiway decision diagram is:

- A directed, acyclic graph in which:
- The nodes are organized into levels
- Level 0 is a special terminal level
- All arcs from a node at level \( k > 0 \) point to nodes at level \( k - 1 \).
- Edges are labeled with positive integer values

Quasi-Reduced MDDs

- Redundant nodes, which have all arcs pointing to the same child, are allowed
- Duplicate nodes are not allowed
Multiway Decision Diagrams

- Levels correspond to attributes of a packet or rule
- Terminal nodes correspond to the actions of a rule
- Non-terminal nodes correspond to sets of packets that share some common attributes
- Arcs correspond to choices of value for a packet or rule
- Every path through the MDD represents a firewall rule
Constructing a rule MDD

Processed Rules

We convert each condition of a rule to a list of ranges of values. For instance, the criterion that the source address be from the subnet 192.168.1.0/24 becomes the list ([192–192].[168–168].[1–1].[0–255]).

Example

ACCEPT tcp 192.168.1.0/24 113.137.10.0/24 tcp dpt:22 becomes: ([192–192], [168–168], [1–1], [0–255], [113–113], [137–137], [10–10], [0–255], [3–3], [22–22], ACCEPT).

Ranges become nodes

Each range becomes a node, with arcs for each of the values in the range.
Constructing an MDD

- Default Rule 3
- Rule 3
- Default and Rule 3
- Rule 2

Rules 2 and 3

Rule 1

Complete rule set
**Queries**

<table>
<thead>
<tr>
<th>Query</th>
<th>Description</th>
<th>Addresses</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUERY SADDY TO 113.137.10.4 AND ACCEPTED forward;</td>
<td></td>
<td>[192].[168].[1].* [137].[113].[10].*</td>
</tr>
<tr>
<td>QUERY SADDY NOT FROM 113.137.10.* AND TO 113.137.10.4 AND ACCEPTED forward;</td>
<td></td>
<td>[192].[168].[1].*</td>
</tr>
</tbody>
</table>
Composition

Motivation
- Distributed firewalls allow fine-grained control
- But they are harder to analyze
- Need to adjust MDD model to combine two firewall policies

Solution
- Build rule sets for each firewall
- Construct a “meta-firewall”
- Perform analysis on the meta-firewall
Meta-firewalls

- FORWARD chain of meta-firewall is the intersection of all the FORWARD chains
- INPUT chain of meta-firewall is the intersection of the INPUT chain of the innermost firewall with the FORWARD chains of the remaining firewalls
- OUTPUT chain is the intersection of the OUTPUT chain of the innermost firewall with the FORWARD chains of the remaining firewalls
Firewall Composition

**Chain FORWARD (default ACCEPT):**

<table>
<thead>
<tr>
<th>Action</th>
<th>Source</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>DROP</td>
<td>113.137.9.0/24</td>
<td>anywhere</td>
</tr>
<tr>
<td>ACCEPT</td>
<td>113.137.8.0/24</td>
<td>anywhere</td>
</tr>
<tr>
<td>DROP</td>
<td>anywhere</td>
<td>113.137.10.3</td>
</tr>
<tr>
<td>ACCEPT</td>
<td>anywhere</td>
<td>anywhere</td>
</tr>
<tr>
<td>ACCEPT</td>
<td>anywhere</td>
<td>anywhere</td>
</tr>
</tbody>
</table>

**Chain INPUT (default DROP):**

<table>
<thead>
<tr>
<th>Action</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCEPT</td>
<td>anywhere</td>
<td>anywhere</td>
<td>tcp dpt:22</td>
<td></td>
</tr>
<tr>
<td>ACCEPT</td>
<td>anywhere</td>
<td>anywhere</td>
<td>tcp dpt:993</td>
<td></td>
</tr>
<tr>
<td>ACCEPT</td>
<td>anywhere</td>
<td>anywhere</td>
<td>tcp dpt:25</td>
<td></td>
</tr>
</tbody>
</table>
Composition

QUERY SADDY FROM 113.137.8.5 AND FOR TCP 80 AND ACCEPTED FORWARD;
# Addresses:
# 0 results.

QUERY SADDY FROM 113.137.8.5 AND FOR TCP 25 AND ACCEPTED FORWARD;
# Addresses: 113.137.8.5
# 1 result.
Network Address Translation

DNAT

Destination NAT modifies destination of a packet, usually before filtering, so that it will be routed to a new host.

SNAT

Source NAT modifies source address of a packet, usually after filtering, so that it appears to have come from a different host.
## Network Address Translation

### Chain FORWARD(policy DROP):

<table>
<thead>
<tr>
<th>#</th>
<th>target</th>
<th>prot</th>
<th>source</th>
<th>destination</th>
<th>flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ACCEPT</td>
<td>icmp</td>
<td>anywhere</td>
<td>192.168.2.5</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>DROP</td>
<td>icmp</td>
<td>anywhere</td>
<td>192.168.2.1</td>
<td></td>
</tr>
</tbody>
</table>

### Chain DNAT(policy ACCEPT):

<table>
<thead>
<tr>
<th>#</th>
<th>target</th>
<th>prot</th>
<th>source</th>
<th>destination</th>
<th>flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DNAT</td>
<td>icmp</td>
<td>anywhere</td>
<td>192.168.1.0/24</td>
<td>to:192.168.2.5</td>
</tr>
</tbody>
</table>
Network Address Translation

Before

After

* 192
* 168
* 2
* 1 5

ICMP

DROP

ACCEPT
Network Address Translation

Before
QUERY SADDY TO 192.168.2.1 AND FOR ICMP 8 AND ACCEPTED FORWARD;
# Addresses:
# 0 results.

After
QUERY SADDY TO 192.168.2.1 AND FOR ICMP 8 AND ACCEPTED FORWARD;
# Addresses: *.*.*.*;
# 4294967296 results.
Equivalence Classes

Problem
How does a system administrator know what queries to ask?

Possible Solution:
Download generic queries from the web.

Possible Solution:
Anticipate every possible threat to the network.

A Better Solution:
Look for anomalies in the firewall policy.
Policy-Based Host Classification

Host Classification
Group hosts into equivalence classes for easier visualization and analysis.

Policy-Based
Use information from the firewall policy to distinguish hosts.
Policy-Based Host Classification (II)

Key idea
Convert the firewall policy into a new representation that is easier to understand and manipulate. Abstract away the details to create a policy map of the “distinguishable” hosts.

Advantage
No need for any input other than the firewall policy itself.
This allows us to generate a map of the policy:
Formal Definitions

Source Equivalent

Let $F(X) : \mathcal{P} \rightarrow \{\text{ACCEPT}, \text{DROP}\}$ represent the filtering decision for any packet $X \in \mathcal{P}$. Let $a$ and $b$ be any two IP addresses. Then we say that $a =_S b$ if and only if for all packets $X_a$ and $X_b$ such that $a$ is the source address of $X_a$ and $b$ is the source address of $X_b$, then $F(X_a) = F(X_b)$. 


Formal Definitions (II)

**Destination Equivalent**

Let $a$ and $b$ be any two IP addresses. Then we say that $a = _D b$ if and only if for all packets $X_a$ and $X_b$ such that $a$ is the destination address of $X_a$ and $b$ is the destination address of $X_b$, then $F(X_a) = F(X_b)$. 
Formal Definitions (III)

Equivalence Classes

If $a =_S b$ and $a =_D b$, then $a =_{SD} b$. The relation $=_{SD}$ is an equivalence relation which divides the set of addresses into equivalence classes.

Key Feature

Two hosts belong to the same class if and only if they are treated the same by the firewall.
### Typos

#### Forward (Default Drop)

<table>
<thead>
<tr>
<th></th>
<th>Action</th>
<th>Source</th>
<th>Destination</th>
<th>Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DROP</td>
<td>129.168.2.0/24</td>
<td>192.168.1.0/24</td>
<td>SMTP</td>
</tr>
<tr>
<td>2</td>
<td>ACCEPT</td>
<td>192.168.1.0/24</td>
<td>192.168.2.0/24</td>
<td></td>
</tr>
</tbody>
</table>

#### Classes

- **Class 1:** 192.168.2.[0–255]
- **Class 2:** 192.168.1.[0–255]
- **Class 3:** 129.168.2.[0–255]
- **Class 4:** [Everything Else]
Typos (Actual Map)

Wireless Clients
192.168.2.*

LAN Clients
192.168.1.*

Anomaly
129.168.2.*
Out of Order Rules (Policy Map)

- **Admin Host**: 192.168.1.3
- **Wireless Net**: 192.168.1.[0-2], 192.168.1.[4-255]
- **LAN**: 192.168.2.*
### Out of Order Rules

#### Forward (Default Drop)

<table>
<thead>
<tr>
<th></th>
<th>Action</th>
<th>Source IP Address 1</th>
<th>Source IP Address 2</th>
<th>Destination IP Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DROP</td>
<td>192.168.1.0/24</td>
<td>192.168.2.0/24</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>ACCEPT</td>
<td>192.168.1.3</td>
<td>192.168.2.0/24</td>
<td>SSH</td>
</tr>
</tbody>
</table>

#### Classes

- **Class 1**: 192.168.1.[0–255]
- **Class 2**: 192.168.2.[0–255]
- **Class 3**: [Everything Else]
Out of Order Rules (Actual Map)

Wireless Net
192.168.1.*

LAN
192.168.2.*
Shadowed Rules (Policy Map)
Shadowed Rules

Forward (Default Drop)

<table>
<thead>
<tr>
<th></th>
<th>Action</th>
<th>IP Source</th>
<th>IP Destination</th>
<th>Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ACCEPT</td>
<td>192.168.2.0/24</td>
<td>192.168.2.0/24</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>ACCEPT</td>
<td>192.168.2.0/24</td>
<td>192.168.2.1</td>
<td>SSH</td>
</tr>
</tbody>
</table>

Classes

- Class 1: 192.168.2.[0–255]
- Class 2: [Everything Else]
Shadowed Rules (Actual Map)

Workstations
192.168.2.*
Calculating Equivalence Classes

- Step 1: Generate a multi-valued decision diagram representation of the Firewall Policy.
- Step 2: Calculate \( =_S \) classes from the MDD.
- Step 3: Reorder the MDD to calculate \( =_D \) classes from the MDD.
- Step 4: Combine \( =_S \) elements and \( =_D \) elements to form \( =_{SD} \) classes.
Equivalence Classes in the MDD

Source 1

Source 2

Destination 1

Destination 2

Target

ACCEPT

DROP
Guided Repair

Disadvantages of query tools

- Identify the error, but not its cause.
- Still a lot of work to repair the policy.
- Do not provide a context for the error.
- Some errors may have multiple potential causes.
Why we can’t automate repair

Automated Repair

It would be nice to have a tool that identified problems and then automatically fixed them.

Why not?

- There are multiple ways to “fix” a problem.
- Some of them create new problems.
- No algorithm can decide which “solutions” are valid ones.

Should SMTP packets be allowed to the server? If it’s a mail server, maybe so.
Key problem

To enable fully automated repair, we must give the tool a very narrow specification — but if we can do that, we might as well just write the policy correctly in the first place!
Enabling Repair

A partial solution

Give the tool a partial specification and provide extensive information about violations of this specification to guide the user in repairing the policy.

Assertions

ASSERT FROM 192.168.2.* IS TO 192.168.1.* OR DROPPED forward;
Assertion Held.

ASSERT FROM 192.168.2.* SUBSET OF FOR TCP 22 OR DROPPED forward;
Assertion Failed.
Example

Forward (Default DROP):

<table>
<thead>
<tr>
<th>#</th>
<th>Target</th>
<th>Source</th>
<th>Destination</th>
<th>Interface</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DROP</td>
<td>192.168.1.0/24</td>
<td>anywhere</td>
<td>!eth2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>DROP</td>
<td>192.168.3.0/22</td>
<td>192.168.2.0/24</td>
<td>any</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ACCEPT</td>
<td>anywhere</td>
<td>192.168.2.4</td>
<td>any</td>
<td>dpt:tcp 80</td>
</tr>
<tr>
<td>4</td>
<td>DROP</td>
<td>anywhere</td>
<td>192.168.2.0/24</td>
<td>any</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>ACCEPT</td>
<td>192.168.1.0/24</td>
<td>anywhere</td>
<td>any</td>
<td></td>
</tr>
</tbody>
</table>

Why are HTTP packets from 192.168.1.0/24 to the web server (192.168.2.4) blocked by the firewall?
bool testSubsetAssertion(cond A, cond B):
[1] mddA = condition_to_MDD(A);
[2] mddB = condition_to_MDD(B);
[3] notB = MDD_complement(mddB);
[4] result = MDD_intersect(mddA, notB);
[5] if notEmpty(result) then:
[6]    return ASSERTION_FAILED;
[7] else:
[8]    return ASSERTION_HELD;

The algorithm for the IS operator is similar.
Witnesses and Counterexamples

Key idea

If the assertion is violated, generate an example that gives a context for the error.
## Example

### Forward (Default Drop):

<table>
<thead>
<tr>
<th>#</th>
<th>Target</th>
<th>Source</th>
<th>Destination</th>
<th>Interface</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ACCEPT</td>
<td>anywhere</td>
<td>192.168.1.0/24</td>
<td>eth0</td>
<td>dpt:tcp 22</td>
</tr>
<tr>
<td>2</td>
<td>ACCEPT</td>
<td>anywhere</td>
<td>131.106.3.253</td>
<td>eth1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>DROP</td>
<td>63.118.7.16</td>
<td>anywhere</td>
<td>eth0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>DROP</td>
<td>192.168.2.0/24</td>
<td>anywhere</td>
<td>any</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>ACCEPT</td>
<td>anywhere</td>
<td>anywhere</td>
<td>any</td>
<td>dpt:tcp 80</td>
</tr>
</tbody>
</table>
Example

Subset assertion

ASSERT EXAMPLE (FROM 192.168.2.*
   AND NOT FOR TCP 22)
   SUBSET OF DROPPED FORWARD;
Assertion failed.
Counterexample: TCP packet
   from 192.168.2.1:6362[eth1]
   to 131.106.3.253:25[eth1]
   in state NEW
   with flags[    ].
Rule History

Key Idea

Consider the packets that violate the assertion. Create a list of rules that match those packets. This allows the system administrator to narrow down the problem to only a few rules, rather than dozens or even hundreds of rules.
Implementation

- Create a new MDD that has additional levels.
- Instead of mapping packets to ACCEPT or DROP, map to the firewall id, chain id, and index.
- New intersection operator that finds the rules that match a packet.
A History MDD
Constructing a History MDD

Forward (Default DROP):

<table>
<thead>
<tr>
<th>#</th>
<th>Target</th>
<th>Source</th>
<th>Dest</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DROP</td>
<td>192.168.2.0/22</td>
<td>anywhere</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>ACCEPT</td>
<td>anywhere</td>
<td>192.168.3.0/24</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ACCEPT</td>
<td>anywhere</td>
<td>anywhere</td>
<td>dpt:tcp 25</td>
</tr>
</tbody>
</table>

Packets sent to host 192.168.4.5 should be able to receive mail.
Using a History MDD

Example

```
ASSERT HISTORY FOR TCP 25
   AND TO 192.168.4.5
   AND NOT FROM 192.168.2.*
   SUBSET OF ACCEPTED FORWARD;
```

Assertion failed.

Critical Rules:

- Firewall 0 Chain 1 Rule 1:
  DROP all -- * * 192.168.2.0/22
  0.0.0.0/0

- Firewall 0 Chain 1 Rule 3:
  ACCEPT tcp -- * * 0.0.0.0/0
  0.0.0.0/0 tcp dpt:25
Ongoing Work

Extending ITVal to other firewall systems

- Redseal XML format
- BSD ipfilters
Future Work

**Theory Extensions**
- Filtering at higher layers
- Support for dynamic filtering
- Filtering on additional matches

**Application Extensions**
- More applications of equivalence classes
- New interface for visualizing output
- More work on guided repair
Thank You!