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)
pkts bytes target           prot opt  in     out  source     destination
740K  294M external_packets  0    --  eth2   *    0.0.0.0/0  0.0.0.0/0
190K   79M internal_packets  0    --  eth1   *    0.0.0.0/0  0.0.0.0/0
   0     0 internal_packets  0    --  eth0   *    0.0.0.0/0  0.0.0.0/0
   0     0 ACCEPT            0    --  *      *    0.0.0.0/0  127.0.0.1

Chain FORWARD (policy DROP 0 packets, 0 bytes)
pkts bytes target  prot opt in    out source          destination
 108  7296 DROP    icmp --  eth1  *   0.0.0.0/0       0.0.0.0/0       icmp type 8
   0     0 DROP    icmp --  eth0  *   0.0.0.0/0       0.0.0.0/0       icmp type 8
709K  499M ACCEPT  tcp  --  *     *   0.0.0.0/0       192.168.0.0/22  tcp dpt:22
   0     0 ACCEPT  tcp  --  *     *   0.0.0.0/0       192.168.0.0/22  tcp dpt:80
 936 47842 ACCEPT  tcp  --  *     *   0.0.0.0/0       192.168.0.0/22  tcp dpt:8080
   0     0 ACCEPT  tcp  --  *     *   101.92.26.68    192.168.5.15    tcp dpt:3306
 28M   38G ACCEPT  tcp  --  eth1  *   192.168.0.0/22  0.0.0.0/0       tcp dpt:22
9194 1849K ACCEPT  udp  --  eth1  *   192.168.0.0/22  0.0.0.0/0       udp dpt:53
   0     0 ACCEPT  tcp  --  eth0  *   192.168.0.0/22  0.0.0.0/0       tcp dpt:22
   0     0 ACCEPT  udp  --  eth0  *   192.168.0.0/22  0.0.0.0/0       udp dpt:53
 52M   26G ACCEPT  0    --  *     *   0.0.0.0/0       0.0.0.0/0       state RELATED,ESTABLISHED
   0     0 REJECT  0    --  eth1  *   192.168.5.121   0.0.0.0/0       reject-with icmp-port-unreachable
   0     0 REJECT  0    --  eth1  *   192.168.5.122   0.0.0.0/0       reject-with icmp-port-unreachable
   0     0 REJECT  0    --  eth1  *   192.168.5.123   0.0.0.0/0       reject-with icmp-port-unreachable
   0     0 REJECT  0    --  eth1  *   192.168.5.124   0.0.0.0/0       reject-with icmp-port-unreachable
   0     0 REJECT  0    --  eth1  *   192.168.5.125   0.0.0.0/0       reject-with icmp-port-unreachable
   0     0 REJECT  0    --  eth1  *   192.168.5.126   0.0.0.0/0       reject-with icmp-port-unreachable
   0     0 REJECT  0    --  eth1  *   192.168.5.101   0.0.0.0/0       reject-with icmp-port-unreachable
   0     0 REJECT  0    --  eth1  *   192.168.5.101   0.0.0.0/0       reject-with icmp-port-unreachable
```

Why do we need Formal Analysis?

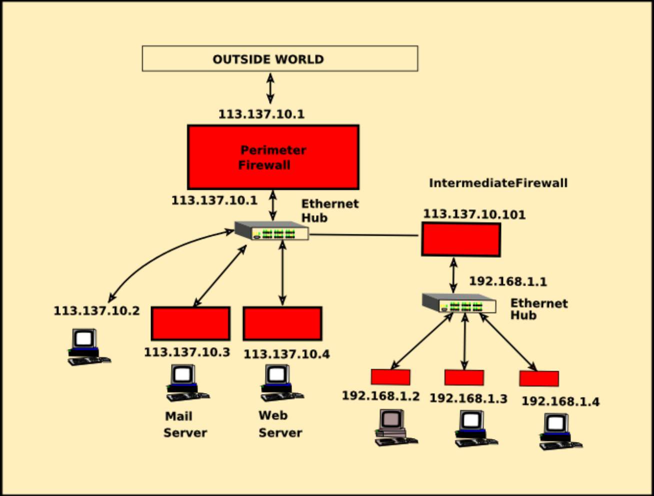
Reasons for Analysis

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

Results of Analysis

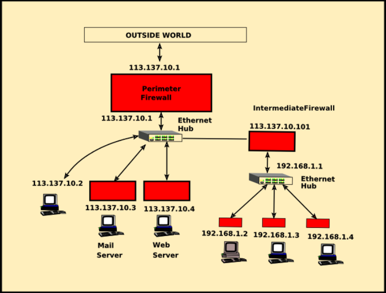
- Develop a better understanding of the policy
- Discover and repair errors in the policy
- Gain assurance in the policy

Firewall policy errors



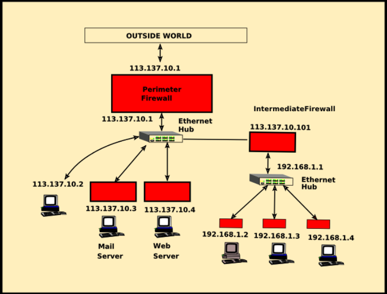
Topology of a large network

Firewall policy errors



Chain FORWARD (policy DROP):					
	target	**prot**	**source**	**destination**	**flags**
1	ACCEPT	all	113.137.10.0/24	113.137.10.4	
2	DROP	all	anywhere	113.137.10.4	

Firewall policy errors



Chain FORWARD (policy DROP):					
	target	**prot**	**source**	**destination**	**flags**
1	ACCEPT	tcp	192.168.1.0/24	113.137.10.0/24	tcp dpt:22
2	ACCEPT	all	113.137.10.0/24	113.137.10.4	
3	DROP	all	anywhere	113.137.10.4	

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

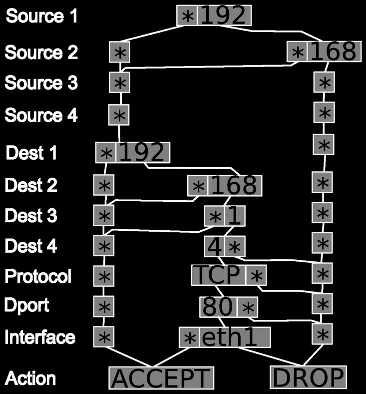
A Multiway decision diagram is:

- 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



Rule Set MDDs

- 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]).

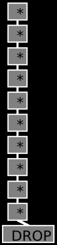
Example

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).

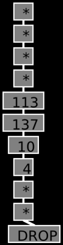
Ranges become nodes

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

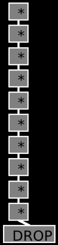
Constructing an MDD



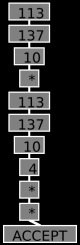
Default



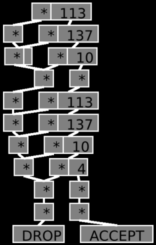
Rule 3



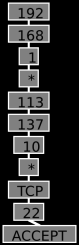
Default and Rule 3



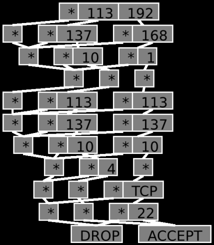
Rule 2



Rules 2 and 3



Rule 1



Complete rule set

Queries

QUERY SADDY TO 113.137.10.4 AND ACCEPTED forward;

Addresses: [192].[168].[1].* [137].[113].[10].*

QUERY SADDY NOT FROM 113.137.10.* AND TO 113.137.10.4 AND ACCEPTED forward;

Addresses: [192].[168].[1].*

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):

DROP	113.137.9.0/24	anywhere	
ACCEPT	113.137.8.0/24	anywhere	
DROP	anywhere	113.137.10.3	
ACCEPT	anywhere	anywhere	tcp dpt:80
ACCEPT	anywhere	anywhere	tcp dpt:53

Chain INPUT (default DROP):

ACCEPT	anywhere	anywhere	tcp dpt:22
ACCEPT	anywhere	anywhere	tcp dpt:993
ACCEPT	anywhere	anywhere	tcp dpt:25

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

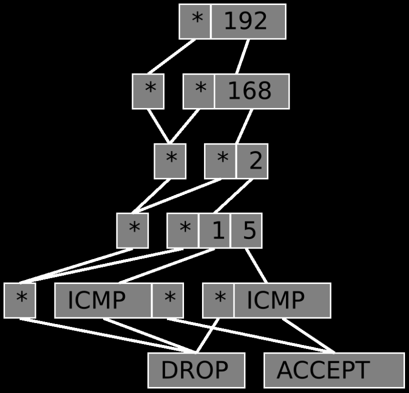
Chain FORWARD(policy DROP):

#	target	prot	source	destination	flags
1	ACCEPT	icmp	anywhere	192.168.2.5	
2	DROP	icmp	anywhere	192.168.2.1	

Chain DNAT(policy ACCEPT):

#	target	prot	source	destination	flags
1	DNAT	icmp	anywhere	192.168.1.0/24	to:192.168.2.5

Network Address Translation



Before After

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

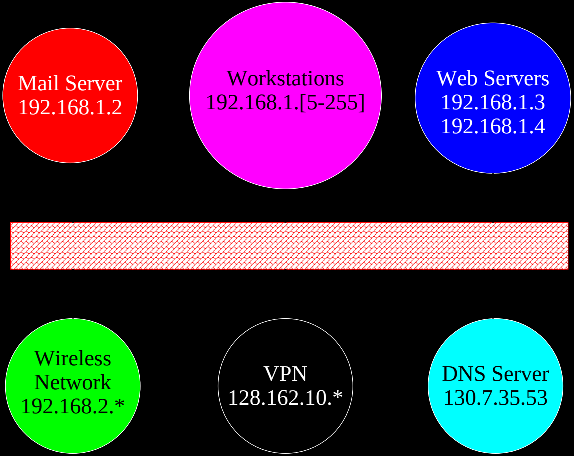
Convert the firewall policy into a new representation that is easier to understand and manipulate. Abstract away the details to create a policy map of the "distinguishable" hosts.

Advantage

No need for any input other than the firewall policy itself.

Policy Map

This allows us to generate a map of the policy:



Formal Definitions

Source Equivalent

Let $F(X) : \mathcal{P} \rightarrow \{ACCEPT, DROP\}$ represent the filtering decision for any packet $X \in \mathcal{P}$. Let a and b be any two IP addresses. Then we say that $a =_{\mathrm{S}} b$ 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 $a =_{\mathrm{D}} b$ 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)

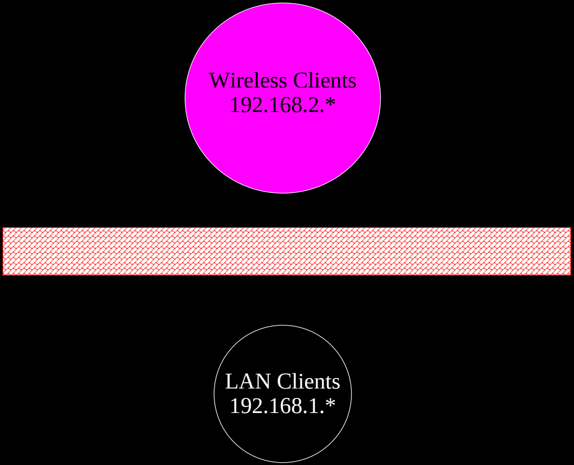
Equivalence Classes

If $a =_{\mathrm{S}} b$ and $a =_{\mathrm{D}} b$, then $a =_{\mathrm{SD}} b$. The relation $=_{\mathrm{SD}}$ 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 (Policy Map)



Typos

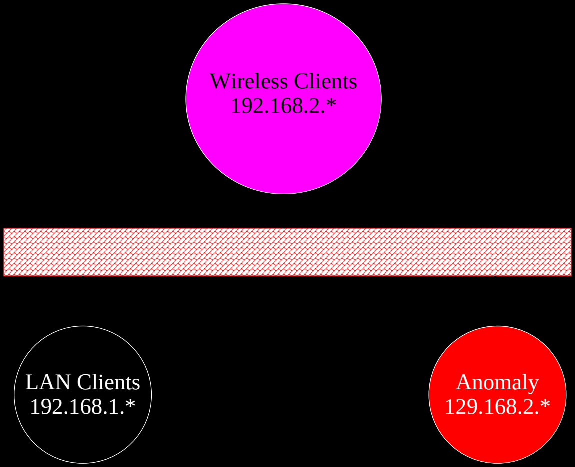
Forward(Default Drop)

1	DROP	129.168.2.0/24	192.168.1.0/24	SMTP
2	ACCEPT	192.168.1.0/24	192.168.2.0/24	

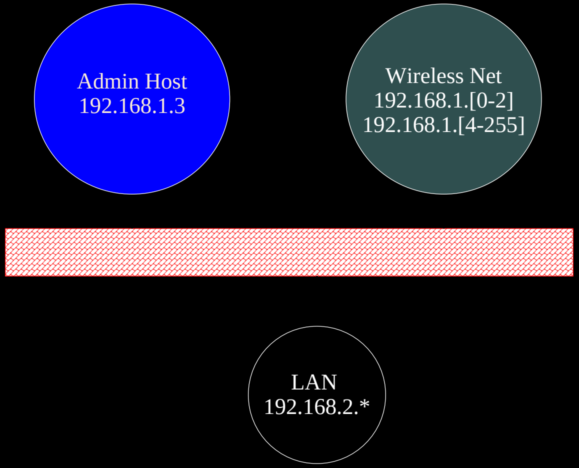
Classes

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)



Out of Order Rules (Policy Map)



Out of Order Rules

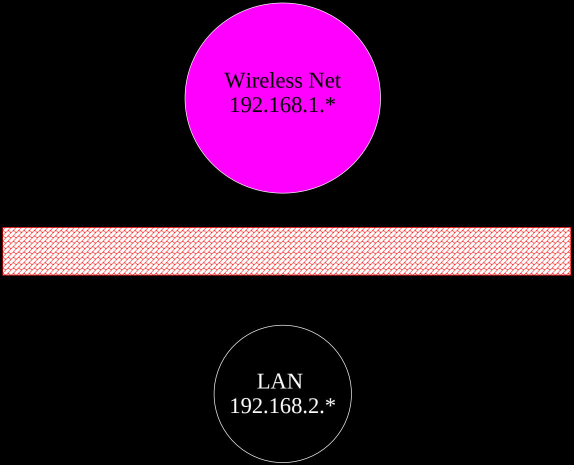
Forward(Default Drop)

1	DROP	192.168.1.0/24	192.168.2.0/24	
2	ACCEPT	192.168.1.3	192.168.2.0/24	SSH

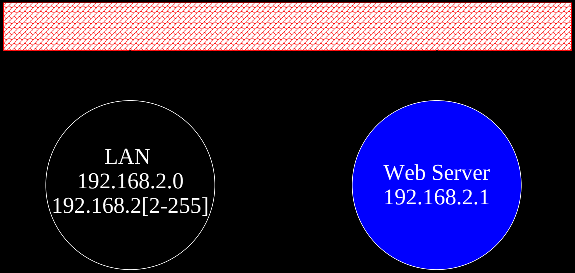
Classes

Class 1: 192.168.1.[0–255]
Class 2: 192.168.2.[0–255]
Class 3: [Everything Else]

Out of Order Rules (Actual Map)



Shadowed Rules (Policy Map)



Shadowed Rules

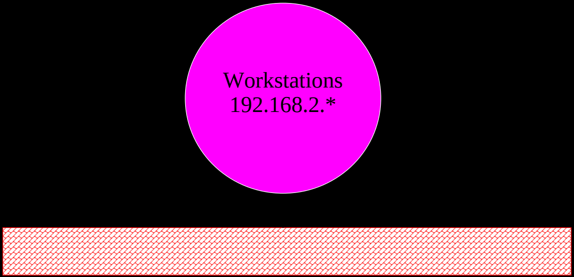
Forward(Default Drop)

1	ACCEPT	192.168.2.0/24	192.168.2.0/24	
2	ACCEPT	192.168.2.0/24	192.168.2.1	SSH

Classes

Class 1: 192.168.2.[0–255]
Class 2: [Everything Else]

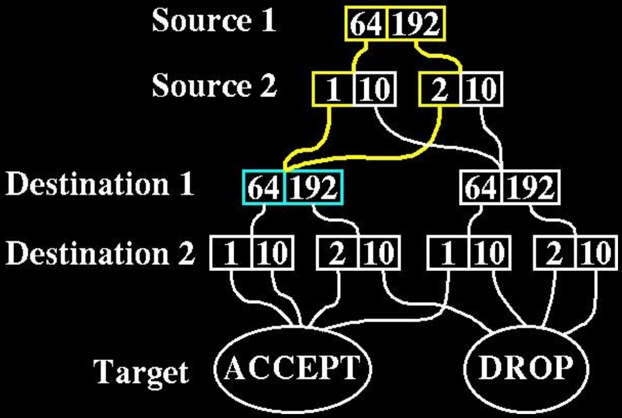
Shadowed Rules (Actual Map)



Calculating Equivalence Classes

- Step 1: Generate a multi-valued decision diagram representation of the Firewall Policy.
- Step 2: Calculate $=_{\mathrm{S}}$ classes from the MDD.
- Step 3: Reorder the MDD to calculate $=_{\mathrm{D}}$ classes from the MDD.
- Step 4: Combine $=_{\mathrm{S}}$ elements and $=_{\mathrm{D}}$ elements to form $=_{\mathrm{SD}}$ classes.

Equivalence Classes in the MDD



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):

#	Target	Source	Destination	Interface	Flags
1	DROP	192.168.1.0/24	anywhere	!eth2	
2	DROP	192.168.3.0/22	192.168.2.0/24	any	
3	ACCEPT	anywhere	192.168.2.4	any	dpt:tcp 80
4	DROP	anywhere	192.168.2.0/24	any	
5	ACCEPT	192.168.1.0/24	anywhere	any	

Why are HTTP packets from 192.168.1.0/24 to the web server (192.168.2.4) blocked by the firewall?

Implementation

```
bool testSubsetAssertion(cond A, cond B):
mddA = condition_to_MDD(A);
mddB = condition_to_MDD(B);
notB = MDD_complement(mddB);
result = MDD_intersect(mddA, notB);
if notEmpty(result) then:
   return ASSERTION_FAILED;
else:
   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):

#	Target	Source	Destination	Interface	Flags
1	ACCEPT	anywhere	192.168.1.0/24	eth0	dpt:tcp 22
2	ACCEPT	anywhere	131.106.3.253	eth1	
3	DROP	63.118.7.16	anywhere	eth0	
4	DROP	192.168.2.0/24	anywhere	any	
5	ACCEPT	anywhere	anywhere	any	dpt:tcp 80

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

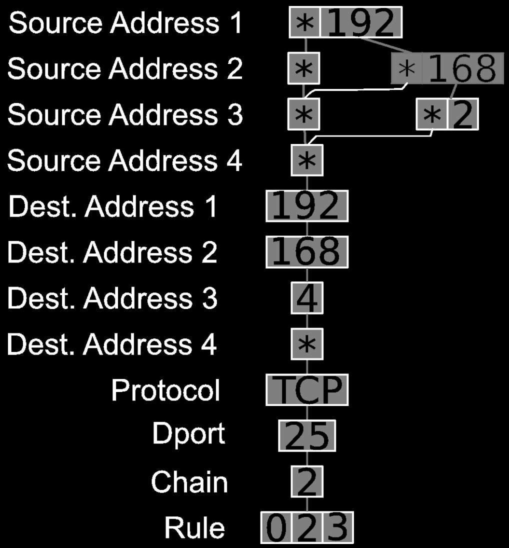
Key Idea

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):

#	Target	Source	Dest	Flags
1	DROP	192.168.2.0/22	anywhere	
2	ACCEPT	anywhere	192.168.3.0/24	
3	ACCEPT	anywhere	anywhere	dpt:tcp 25

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!