E Unum Pluribus

Google Network Filtering Management

(with apologies to the latin nerds about the conjugation)

Paul (Tony) Watson
&
Peter Moody

Google™

A Few Facts About Google's Edge

- Around 6,000 configured public services VIPs
  - Many of these are shared between multiple services
  - Each VIP and service has multiple backend systems
- Around 100 distinct services ports
  - We are not just web anymore!
- HTTP / HTTPS services run on over 4,600 of those VIPs
- Edge cache deployments

- Recently, Facebook announced it had over 30,000 servers
  - This amounts to a rounding error compared to the infrastructure we are required to protect at Google

- Unfortunately, we can't release specific numbers

Google™

Network Access Control Needs

- Specific Router ACL Needs
  - Production filtering
  - Edge cache filtering
  - Corporate filtering
  - Internal filtering (labs, contractors, special needs, etc)
  - Acquisition access filtering
  - Transit filtering
  - Individual host and network device filtering
- Stateful firewalls are used in many places as well, but are not alw
  ays necessary, or cannot meet the throughput requirements.

Historical ACL Management at Google

- Manual maintenance of hundreds of individual filters which were manually updated as needed
  - Often required duplication of large blocks of networks to multiple filters that had varying syntax and formats
  - Method was prone to human error and typos
  - Very time consuming to maintain
  - Extremely difficult to review and audit
  - Often required maintaining of identical filters for multiple platforms (Juniper, Cisco, F10, etc.)
- Several previous efforts helped simplify the process, but these project

E Unum Pluribus (Out of one, many)

- What was needed was a common language to describe security policies and a stan d ardized interconnect between the language and policy rules
- The language should define a policy and be clear and easy to read, but fle x ible enough to accommodate most common filtering formats
- Policies should be able to share common objects and definitions (ASNs, hosts, networks, grou p s of hosts and networks, services and service groups, etc.)

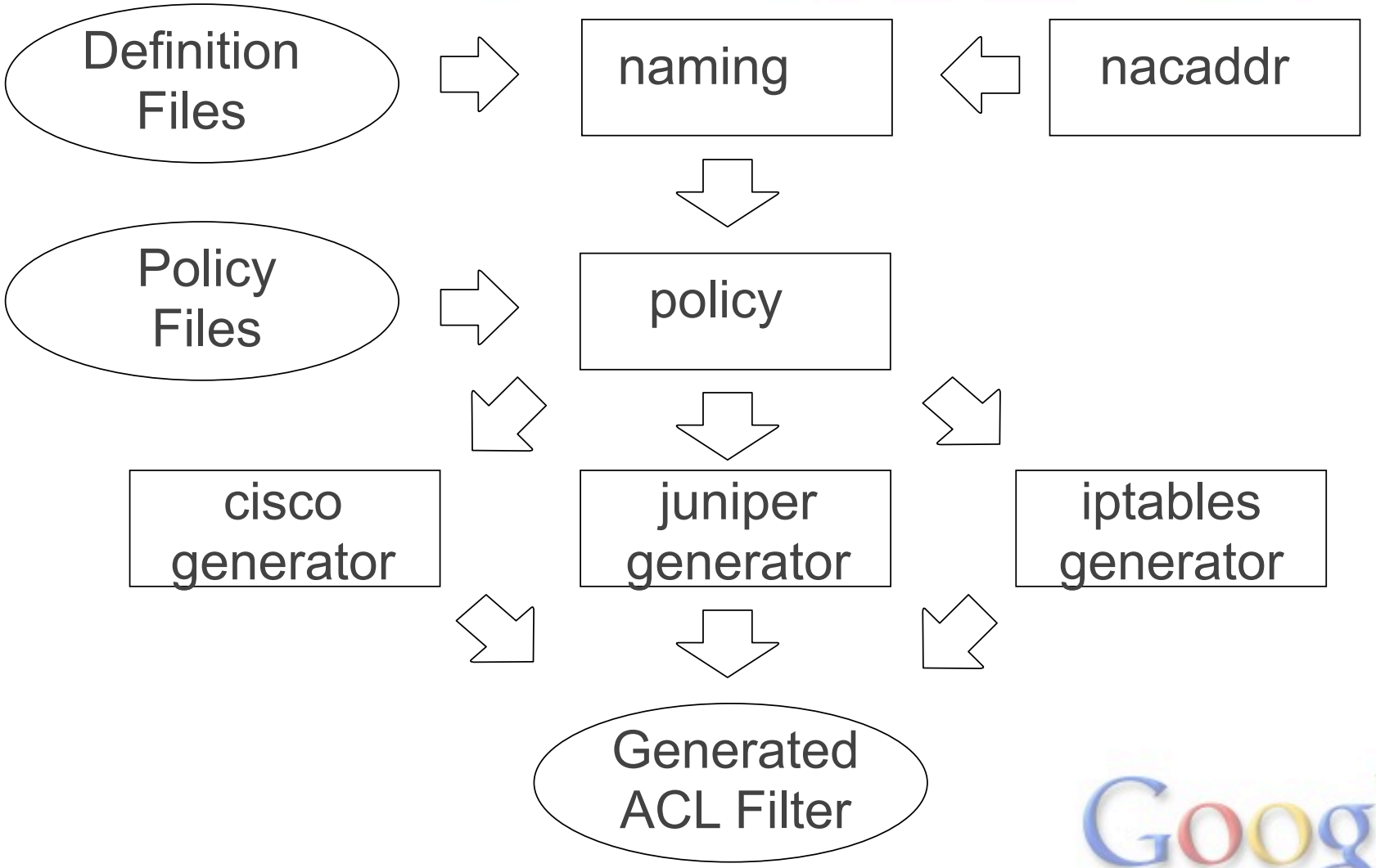
Initial Design Structure

The system was designed in a modular fashion to allow us to independently develop and test the various components and allow for reuse in later tools.

- IP Address library
  - ipaddr / nacaddr
- Naming library
- Policy library
- Generator libraries
  - Juniper
  - Iptables
  - Cisco standard
  - Cisco extended
  - Cisco named
  - others
- Compiler (aclgen)
- Unit tests

Google™

ACL Generation Process Flow



Google™

Overview of Libraries

The following slides provide a brief overview of the various libraries and components used in the ACL generation system.

The system is command line based, but designed such that it will easily allow overlay of various Web or other GUI interfaces

Release early, release often

The system we use in-house has several key differences:

- perforce integration for revision control and reviews
- iptables system with custom deployment and loader
- integration with other internal systems and processes
- more

IP Address Library

What it provides:

- lightweight, fast IP address manipulation.

To define an IP address object:

```
import nacaddr
ip = nacaddr.IP('10.1.1.0/24', 'text comment', 'token name')
```

The text comment and token name are optional, and provide extensions to the base IPaddr library that allow us to carry comments from the naming definitions to the final output.

Next, lets examine the methods available to the 'ip' object.

IP Address Library

ip.version	-> numeric value, 4 or 6	
ip.text	-> value of text comment	
ip.token	-> value of naming library token	
ip.parent_token	-> value of naming parent token, if nested	
ip.prefixlen	-> numeric prefix length of IP object (24)	
ip.numhosts	-> number of addresses within prefix (256)	
ip.ip_ext	-> IP address	10.1.1.0
ip.netmask_ext	-> netmask of address	255.255.255.0
ip.hostmask_ext	-> hostmask of address	0.0.0.255
ip.broadcast_ext	-> broadcast address	10.1.1.255
ip.network_ext	-> network address	10.1.1.0

* Non _ext methods also exist, that provide integer values.

Naming Library

The naming library provides an easy way to lookup addresses and services based on token names, which we refer to as definitions. We store definitions in a directory containing an arbitrary number of files. Files can be used to separate definitions based on roles or function, but this filename distinction does not carry into the object usage.

Network defintions files must end in '.net'
Service definitions files must end in '.svc'

Multiple groups can maintain individual .net or .svc files
Definitions can then be easily used by other tools or teams

**creating a naming standard is always encouraged*

Naming Network Definitions Format

```
RFC1918 = 10.0.0.0/8          # non-public
          172.16.0.0/12    # non-public
          192.168.0.0/16  # non-public

INTERNAL = RFC1918

LOOPBACK = 127.0.0.1/32  # loopback
           ::1/128          # ipv6 loopback

NYC_OFFICE = 100.1.1.0/24  # new york office
SFO_OFFICE = 100.2.2.0/24  # san francisco office
CHI_OFFICE  =  100.3.3.0/24  # chicago office

OFFICES = NYC_OFFICE SFO_OFFICE
          CHI_OFFICE
```

Google™

Naming Service Definitions Format

WHOIS = 43/udp

SSH = 22/tcp

TELNET = 23/tcp

SMTP = 25/tcp

MAIL_SERVICES = SMTP
ESMTP
SMTP_SSL
POP_SSL

DNS = 53/tcp 53/udp

Naming Library Usage

```
>>> import naming
>>> definitions = naming.Naming('/my/definitions/directory')
>>> dir(definitions)
['GetIpParents', 'GetNet', 'GetNetAddr', 'GetService',
'GetServiceB
yProto', 'GetServiceParents', 'ParseNetworkList', 'ParseServiceList', ...]

>>> definitions.GetNet('INTERNAL')
[IPv4('10.0.0.0/8'), IPv4('172.16.0.0/12'), IPv4('192.168.0.0/16')]
```

**note that this returns NacAddr objects, allowing easy IP address manipulation.*

```
>>> definitions.GetService('DNS')
['53/tcp', '53/udp']

>>> definitions.GetServiceByProto('DNS','tcp')
['53']
```

Policy Library

- The policy library is intended to read and interpret high-level network policy files
- Uses the naming library to convert tokens to networks and services
- Creates an object that is suitable for passing to any of the output generators
- Each policy definition file contains 1 or more filters, each with 1 or more terms
  - Header sections - defines the filter attributes
  - Term sections - defines the rules to be implemented
- There is no support for NAT at this time
  - You can add support and submit patches
- Policy language has both required and optionally supported keyword - generators must support required keywords

Policy Definition Format

```
header {
    comment:: "edge input filter for sample network."
    target:: juniper edge-inbound
}
term discard-spoofs {
    source-adress:: RFC1918
    action:: deny
}
term permit-ipsec-access {
    source-address:: REMOTE_OFFICES
    destination-address:: VPN_HUB
    protocol:: 50
    action:: accept
}
....
```

Google™

Generator Libraries

There are current 3 generator libraries, more are desired

- Juniper
- Cisco
- Iptables

Juniper can generate 3 output formats:

- IPv4
- IPv6
- Bridge

Cisco can generate 3 output formats:

- extended
- standard
- object-object (extended with object-groups)

Iptables can generate 2 output formats:

- IPv4
- IPv6

Cisco Generator

- Renders policy objects into Cisco network ACL filters
- Defaults to generating "extended" ACL filters
- Supports several output formats:
  - Extended
  - Standard
  - Object-Group
- Does not currently support IPv6 filter generation
- Output text begins with "no ip access-list...", then defines replacement with "ip access-list..."
  - Provides for easy cut-paste deployment
- Each policy term is identified in remark text

Cisco Generator

Defining Cisco output in the Policy "header" section:

```
header {
    comment:: "cisco filter header"
    target:: cisco [filter name] {extended|standard|object-group}
}
```

For standard ACLs, the format is:

```
header {
    comment:: "cisco filter header"
    target:: cisco [number] standard
}
```

Google™

Juniper Generator

The most fully featured generator, since Google has a long history as a Juniper partner

Supports most "optional" policy definition keywords:

- *destination-prefix::* currently only supported by the juniper generator
- *ether-type::* currently on used by juniper generator to specify arp packets
- *fragement-offset::* currently only used by juniper generator to specify a fragment offset of a fragmented packet
- *icmp-type::* [echo-reply|echo-request|port-unreachable]
- *logging::* specify that this packet should be logged
- *loss-priority::* juniper only, specify loss priority
- *packet-length::* juniper only, specify packet length
- *policer::* juniper only, specify which policer to apply to matching packets
- *precedence::* juniper only, specify precendence
- *qos::* apply quality of service classification to matching packets
- *routing-instance::* juniper only, specify routing instance for matching packets
- *source-prefix::* juniper only, specify source-prefix matching
- *traffic-type::* juniper only, specify traffic-type
  - [broadcast|multicast|unknown_unicast]

Google™

Juniper Generator

Defining Juniper output in the Policy "header" section:

```
header {
    comment:: "juniper filter header"
    target:: juniper [filter name] {inet|inet6|bridge}
}
```

Iptables Generator

- Used within Google as component of a host based security system.
- The current output format is not suitable for 'iptables-restore'
  - This is planned for the open-source version shortly
  - Until then, each line can be passed to /sbin/iptables
  - Internally, Google uses its own specialized loader (speedway)
- Supports both IPv4 and IPv6 filter generation
- Terms are rendered as jumps in the base filters
  - Optimization algorithm desirable, especially for large filters
- Permits setting of default policy on filters

Iptables Generator

Defining Iptables output in the Policy "header" section:

```
header {
    comment:: "iptables filter header"
    target:: iptables [INPUT|OUTPUT|FORWARD] {ACCEPT|DROP} {inet|inet6}
}
```

Internally, we generate multiple smaller Iptables filters that each provide a specific function, then chain then together for create policies.

For example: we have a base policy that is always applied, and may include one or more additional 'modules' to enable functionality such as web-services, mail-services, etc.

Google™

Compiler (AclGen)

Located in parent directory: aclgen.py

Arguments:
  -d [definitions]
  -p [policy source file]
  -o [output directory]
  --poldir [policy source directory]
  --help -h

The -p option is generates output for a single policy source file

The --poldir option allows you to generate ACLs for an entire directory of source policies

Assurance / Validation Development

The following slides provide a brief overview of the various libraries and components used in our ACL assurance and validation processes.

These tools are essential parts of the ACL process at Google.

We do not want our customers to suffer an outage due to an error or accident in our ACL management.

Assurance / Validation Development

Once the initial system was built, it allowed us to easily do things that were previously very difficult or impossible.

Regular reports are now generated advising us of potential problems or issues.

Other code and projects have also integrated components of our system into their own code, such as naming library & definitions.

- AclCheck library
  - NacParser libary
  - AclTrace library
- Netflow validation
  - aka "snackle"
- Netscaler validation
  - aka "crackle"
- Policy Reader library
- Term Occlusion library
- Iptables assurance
  - aka "Pole Position"

AclCheck Libary

- Having all the various flavors of ACLs in a single policy format allows us the ability to easily analyze filters
- Allow verification of specific packets against a policy to determine what matches will occur
- Pass in policy, src, dst, dport, sport, proto and it returns and aclcheck object
- Methods:
  - ActionMatch(action) - matched terms for this exact action
  - DescribeMatches() - text descriptions of matches
  - ExactMatches() - excludes 'next' actions
  - Matches() - list of matched terms
- AclCheck is the basis for most of our ACL validation tools that we describe in the following slides

Google™

Netflow Validation (aka Snackle)

- We cannot tolerate accidental outages due to ACL errors

- "Snackle" compares huge amounts of previous netflow data against proposed ACL changes
- Alerts us whenever a new ACL is built, but before it is pushed out if a possible conflict is detected
- Allows us to detect errors before they might affect our users
  - accidentally blocking POP3 to gmail servers for example
- Obviously, it cannot identify problems that result from "new" services that did not exist in previous netflow sessions

*This tool is not being released at this time

Google™

Netflow Validation (aka Snackle)

Example Snackle Report Text:

```
deny->accept
id=1003,64.81.47.74:34609,216.73.86.153:80(global-discard-reserved)(global-accept-gtransit-customer-af4)
id=1035232,98.171.189.17:52555,209.62.189.11:80(global-discard-reserved)(global-accept-gtransit-customer-af4)
id=1036450,66.74.106.59:1989,209.62.176.153:80(global-discard-reserved)(global-accept-gtransit-customer-af4)
...
```

Or

```
accept->deny
id=1003,64.81.47.74:34609,216.73.86.153:80(global-accept-gtransit-customer-af4)(global-discard-reserved)
id=1035232,98.171.189.17:52555,209.62.189.11:80(global-accept-gtransit-customer-af4)(global-discard-reserved)
id=1036450,66.74.106.59:1989,209.62.176.153:80(global-accept-gtransit-customer-af4)(global-discard-reserved)
...
```

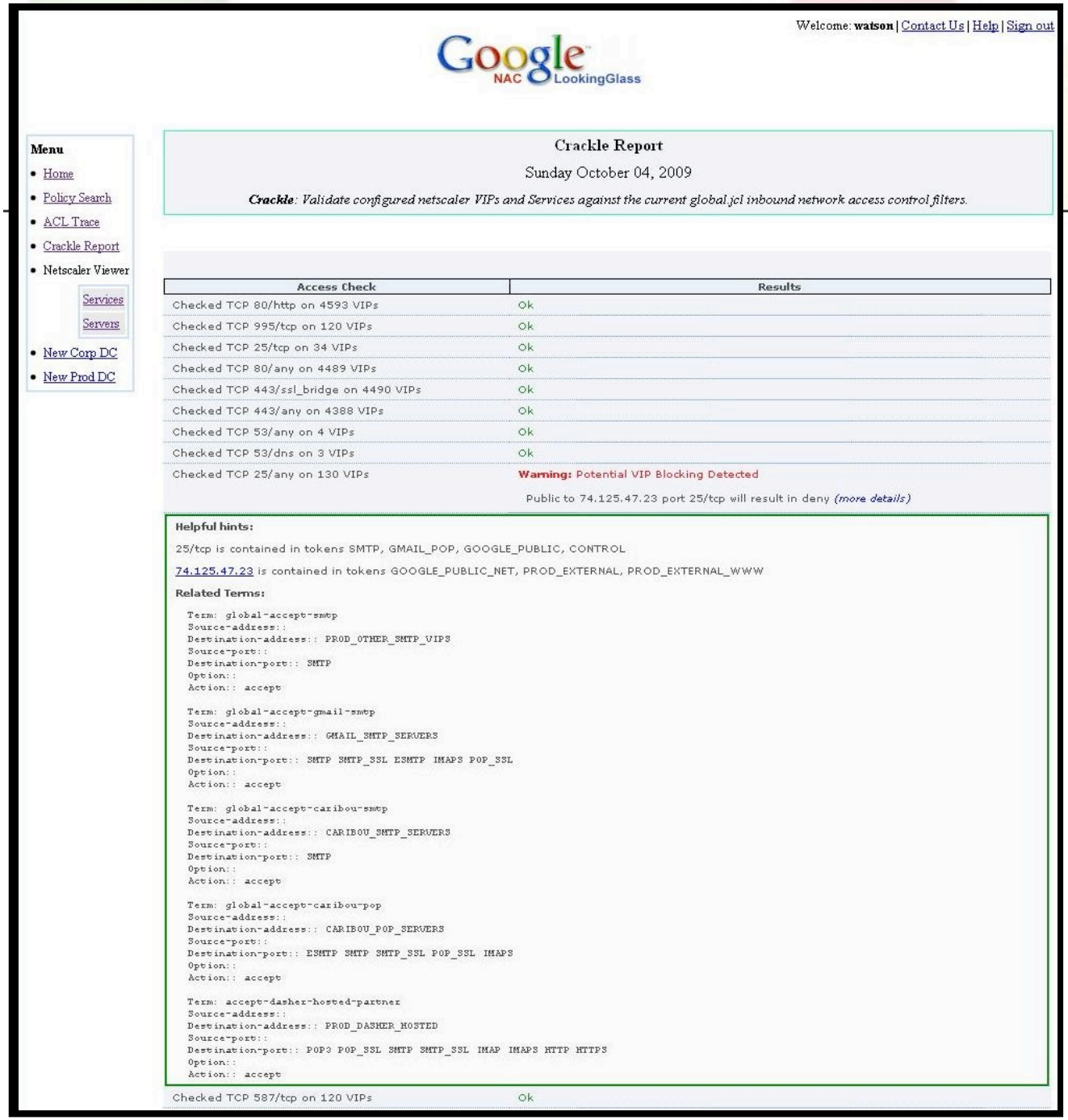
VIP Validation (aka Crackle)

- We cannot tolerate accidental outages due to ACL errors

- "Crackle" parses configurations of our public VIPs to determine what IPs and services should be available
- Alerts us whenever a new ACL is built, but before it is pushed out, if a possible conflict is detected
- Allows us to detect errors before they might affect our users
  - inadvertently blocking POP3 to Gmail servers for example
- Also identifies stale or misconfigured load balancers
- This has saved us from inadvertent outages on several occasions

*This tool is not being released at this time

Google™



In this example, we see that 25/tcp is being blocked to a public IP that was configured to receive SMTP.

The "details" dropdown advises us which service tokens contain 25/tcp, and which network tokens contain the public IP.

Then it shows us likely related ACL terms.

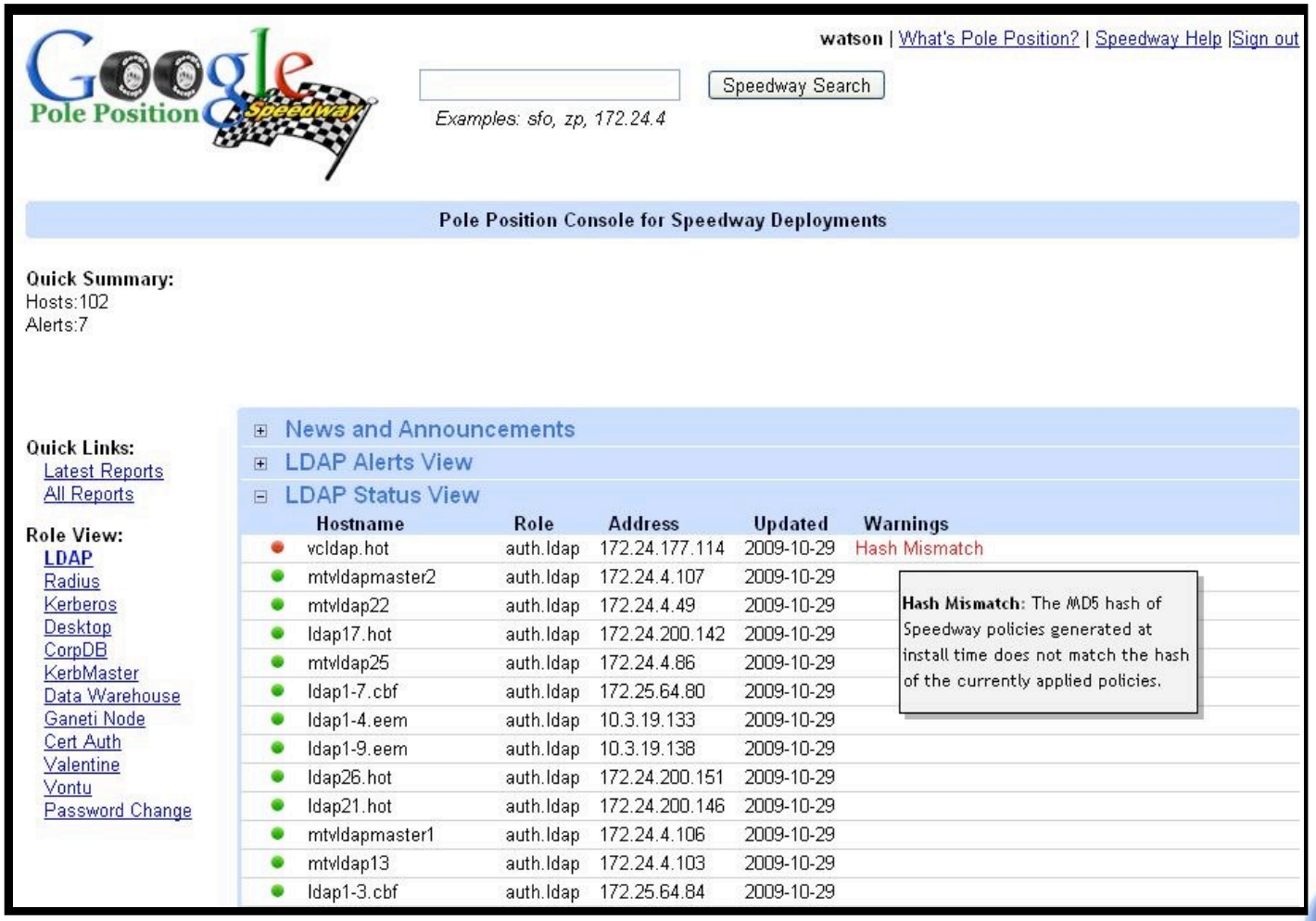
Google™

Iptables Assurance - aka Pole Position

- Adds deployment tracking to Google "Speedway" deployments
- All deployment report back to central collector at regular intervals
  - install hash, current hash, role, modules, interface stats
- Collector performs variety of functions on data
  - validates reports
  - stores valid data in database
  - analyzes data for issues
  - reports in real-time though Web UI
    - all hosts
    - per role reports

*This tool is not being released at this time

Google™



Simple search box allows us to find hosts by DNS or IP matching.

The "Recent Alerts" (closed) shows only the hosts reporting errors.

The "Recent Reports" shows all hosts in the selected role.

Google

Policy Reader library

- The policy reader library is a recent addition that allows other code to easily examine policy source files
- The policy library only reads policies for the purpose of rendering objects for passing to generators

- For some tools, we needed to be able to easily examine the various filters and terms for programatically
  - where certain tokens are used
  - where specific options are used
  - etc.
- Policy reader renders simple objects that allow us to do this
- Handy for a variety of tools, such as rendering policies in a Web UI for example

Google™

Term Occlusion library

Another library built onto this system examines complex ACLs to identify when a term will block or overlap subsequent terms

This library helps us to identify common errors such as:

- overly broad terms
- mismatched QoS accepts (more specific before more general terms)

*This is not being released at this time

Summary - Do Know Evil!

- ACLs are highly prone to human error
- Manually auditing and reviewing large and complex ACLs is very difficult and time consuming
- Keeping large blocks of networks in sync between large numbers of ACLs is time consuming and error prone
- Automating these tasks reduces manual labor, helps eliminate typos, and helps identify logical errors

Without this system, we would be overwhelmed today due to the size, complexity and large number of ACLs in the Google environment.

We have open sourced much of this code hoping to help other large and small business.

Google™

Core Code Released to the Public

We have open-sourced software under the Apache2 license

http://code.google.com/p/capirca/

** Detailed help and documentation is available on the wiki **

If you use it and modify it, please contribute your patches back.

The name, "capirca", was intended to be "caprica" from BattleStar galactica (the "new world"). I registered the misspelling, then later noticed the error, but the correct spelling was already taken.

So, for efficiency(?) we have kept the name Capirca.