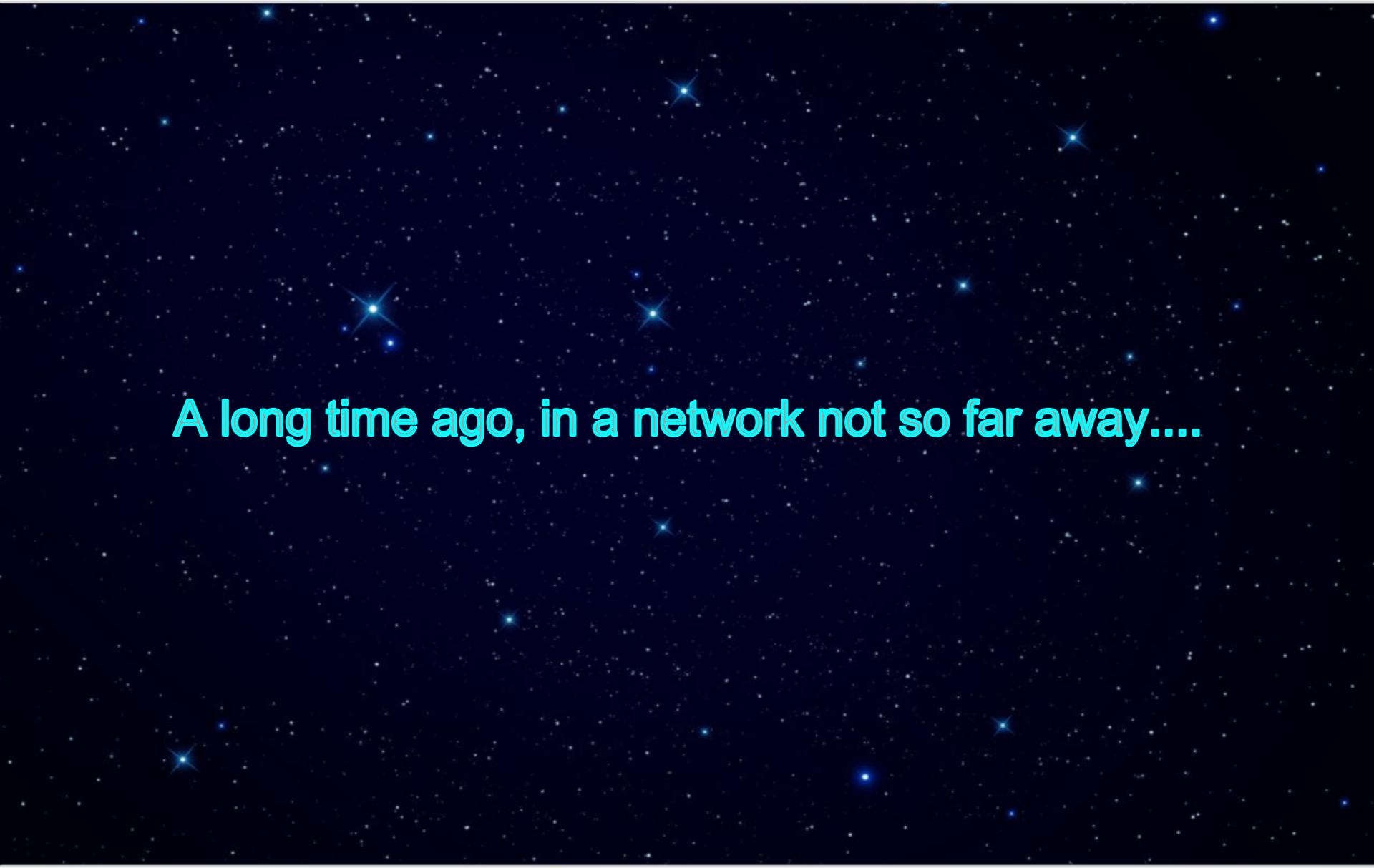
Multi-Platform ACL Generation and Testing

Capirca Network ACL Generation

Paul (Tony) Watson



Google

ACLs were written and (poorly) maintained by hand...

permit 6 any any range 33434 33534
permit 6 host 216.239.32.26 eq 49 any established
permit 6 host 216.239.34.26 eq 49 any established
permit 6 host 172.20.0.153 eq 53 any
permit 6 72.14.232.0 0.0.7.255 eq 22 any established
permit 6 any eq 179 host 114.123.56.1
permit 6 2620:0:1000::/40 any eq 22
permit 6 2620:15c::/36 any eq 22
permit 6 any 2001:4860:4000::/37 range 1024 65535 established
permit 6 any 2800:3f0:4000::/37 range 1024 65535 established
.....
.....
.....

Google

Challenges of Network ACL Maintenance...

- Dealing with large numbers of varied ACLs across an organization
- Repeated use of changing CIDR blocks and groups of hosts
- Duplication of CIDR blocks and host IPs across multiple filters with varying platform syntax and formats
- Extremely difficult to review and audit
- Very time consuming
- Prone to human error and typos
- Often requires maintaining identical policies for multiple platforms
- Revision / change control for ACL modifications
- Endless...

Google

Old School Methods....

- Using Editor tools to "search & replace" ACL text files
- Memorizing massive numbers of IPs and Netblocks
- Lots and lots of 'remarks' to keep track of what groups of rules do
- Unix tools like 'grep' and 'sed' to find and replace
- Expensive Commercial Packages offered some help, but limited in capabilities and scope and no ability to customize as needed
- And so on...

Google

What Was Needed...

- A common language to describe security policies and a standardized interconnect between language and platform syntax
- Language should define a policy and be clear and easy to read, but flexible enough to accommodate most common filtering formats
- Policies should be able to share common objects and definitions (Hosts, CIDR blocks, Services and groups of nested items)
- Automate as much of the process as possible to reduce the potential for human error, speed the time to delivery, and reduce the expertise needed to manage changes
- Easy use of #includes in network filtering policies

- *Write once, output many...*

Google

Real World Deployment Scenarios...

- Need to easily copy policies to a new platform
- Need to apply router edge policy on host machines
- Defense in depth strategies
- Ability to create automated unit tests for ACL policies
  - Ensure XXX traffic is always permitted before changes are pushed
  - Ensure YYY traffic is always blocked before changes are pushed
  - Ensure ZZZ traffic matches on a specific term
- Avoid time-consuming duplication of terms, address sets, or service groups across multiple devices and platforms (#includes)
- Easy generation of both IPv4 and IPv6 rules from a single policy
- Verify newly modified policies against "recent" traffic scenarios
- Modifiable to support new devices/platforms quickly

Google

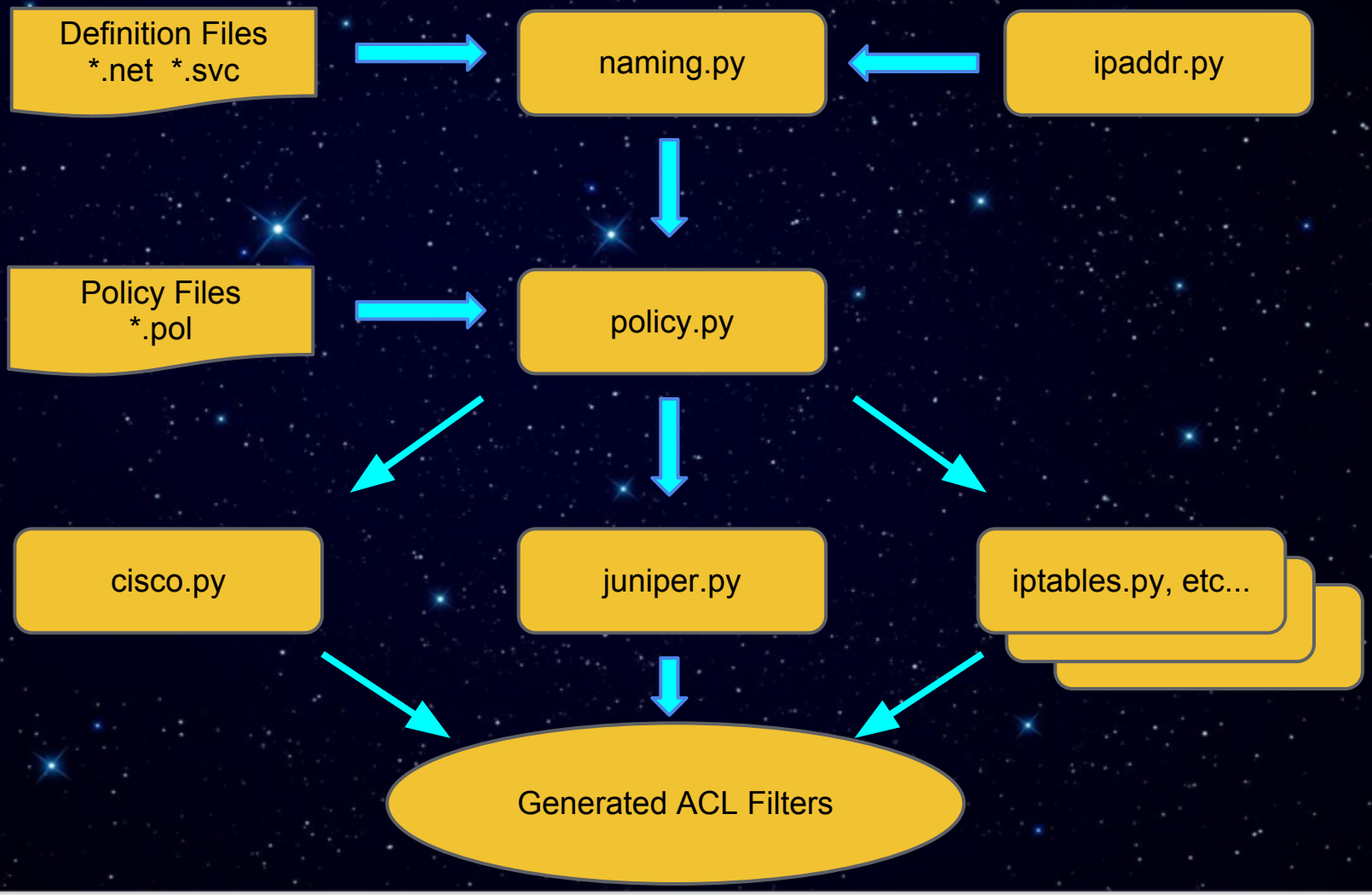
Capirca Design Structure

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.

- Naming library
- IP Address library
- Policy library
- Generator libraries
  - Juniper / JuniperSRX
  - Iptables / Speedway
  - Cisco / Cisco ASA
  - PacketFilter
  - Silverpeak
  - others
  - easily extensible…
- Compiler (aclgen.py)
- Unit tests

Google

ACL Generations Process Flow



Google

Overview of Libraries...

The following slides provide a brief overview of the various libraries and components built for the Capirca 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 management
- Automated "push" tools for router ACLs
- Separate code-tree for development (most changes ported in/out)

Google

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 definitions 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*

Google

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

Google

Naming Library Usage

```
>>> import naming
>>> definitions = naming.Naming('/my/definitions/directory')
>>> dir(definitions)
['GetIpParents', 'GetNet', 'GetNetAddr', 'GetService', 'GetServiceByProto',
'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']
```

Google

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.

Google

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.

* Logical changes in this library are pending, stay tuned.

Policy Library

- The policy library is intended to read and interpret high-level network policy definition files
- Uses the naming library which converts 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

Google

Example Policy Definition

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

Google

Rendered Example - Cisco

```
$ cat example.acl

remark $Id:$
remark $Date:$
no ip access-list extended edge-inbound
ip access-list extended edge-inbound
remark edge input filter for sample network.

remark discard-spoofs
 deny ip 10.0.0.0 0.255.255.255  any
 deny ip 172.16.0.0 0.15.255.255  any
 deny ip 192.168.0.0 0.0.255.255  any

remark permit-ipsec-access
 permit 50 1.1.1.0 0.0.0.255  host 3.3.3.3
 permit 50 1.1.2.0 0.0.0.255  host 3.3.3.3
 permit 50 2.1.1.0 0.0.0.255  host 3.3.3.3
```

Google

Rendered Example - Speedway (iptables)

```
$ cat example.ipt

*filter
# $Id: ./filters/x.ipt $
# $Date: 2013/05/27 $
# inet
:INPUT DROP
-N I_discard-spoofs
-A I_discard-spoofs -p all -s 10.0.0.0/8 -j DROP
-A I_discard-spoofs -p all -s 172.16.0.0/12 -j DROP
-A I_discard-spoofs -p all -s 192.168.0.0/16 -j DROP
-A INPUT -j I_discard-spoofs
-N I_permit-ipsec-access
-A I_permit-ipsec-access -p 50 -s 100.1.1.0/24 -d 3.3.3.3/32 -m state --state NE
W,ESTABLISHED,RELATED -j ACCEPT
-A I_permit-ipsec-access -p 50 -s 100.2.2.0/24 -d 3.3.3.3/32 -m state --state NE
W,ESTABLISHED,RELATED -j ACCEPT
-A I_permit-ipsec-access -p 50 -s 100.3.3.0/24 -d 3.3.3.3/32 -m state --state NE
W,ESTABLISHED,RELATED -j ACCEPT
-A INPUT -j I_permit-ipsec-access

COMMIT
```

Google

Rendered Example - Juniper

```
firewall {
    family inet {
        replace:
        /*
        ...
        ** edge input filter for sample network.
        */
        filter edge-inbound {
            interface-specific;
            term discard-spoofs {
                from {
                    source-address {
                        10.0.0.0/8; /* non-public */
                        172.16.0.0/12; /* non-public */
                        192.168.0.0/16; /* non-public */
                    }
                }
                then {
                    discard;
                }
            }
            term permit-ipsec-access {
                from {
...
```

```
...

*/

}}}
```

```
        source-address {
            1.1.1.0/24; /* Remote Office 1 */
            1.1.2.0/24; /* Remote Office 1 - annex

            2.1.1.0/24; /* Remote Office 2 */
        }
        destination-address {
            3.3.3.3/32; /* vpn concentrator */
        }
        protocol 50;
    }
    then {
        accept;
    }
```

Google

Generator Libraries

There are currently 7+ generator libraries, more are desired

- Juniper, SRX, Cisco, Cisco ASA, Iptables, Speedway, PacketFilter, SilverPeak

Juniper can generate 3 output formats:

- IPv4, IPv6, Bridge

Cisco can generate 3 output formats:

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

Iptables can generate 2 output formats:

- IPv4, IPv6 *(speedway outputs both in one policy)*
- Slightly odd output format - use "speedway" for most uses
- For *'iptables-restore'* output format, use "speedway"

Google

Cisco Generator

- Renders policy objects into Cisco network ACL filters
- Defaults to generating "extended" ACL filters
- Supports several output formats:
  - Extended
  - Standard
  - Object-Group
- 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
- Object-Group is essentially what we've done in the framework for hosts and services

Google

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
- *fragment-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}
}
```

Google

Iptables Generator

- Used within Google as component of a host based security system
  - **Most people should use *"Speedway"* instead**
- The current output format is not suitable for 'iptables-restore'
  - Each line must be individually passed to /sbin/iptables
  - Internally, Google uses its own specialized loader
- Supports both IPv4 and IPv6 filter generation
- Terms are rendered as jumps in the base filters
  - Optimization algorithm desirable, especially for large filters
- Allows setting of default policy on filters

Google

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 them together to 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

Speedway Generator

Defining Speedway output in the Policy "header" section:

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

}
```

Policy terms allow for interface specification, if desired, to apply individual terms

"nostate" specifies no state-tracking for packet flows

Google

Compiler (AclGen)

Located in parent directory: aclgen.py

Arguments:
  -h, --help (Show this help message and exit)
  -d [definitions]
  -p [policy source file] (mutually exclusive with --poldir)
  -o [output directory]
  --poldir [policy source directory] (mutually exclusive with -p)
  -s, --shade_checking (Enable shade checking)
  -e EXP_INFO, --exp_info=EXP_INFO
     (this applies to terms with "expiration::" keyword)

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

Google

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 Network Filter management processes at Google.

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

* Unfortunately, most of these tools aren't being released at this time.

Google

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"
- Load balancer validation
  - aka "crackle"
- Policy Reader library
- Occlusion detection
- Iptables assurance
  - aka "Pole Position"
- Definate
  - Object defs from authoritative sources

Google

AclCheck Library

- 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
  - such as accidentally blocking POP3 to gmail servers
- Obviously, it cannot identify problems that result from "new" services that did not exist in previous netflow sessions

*This tool is not 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-transit-customer)
id=1035232,98.171.189.17:52555,209.62.189.11:80(global-discard-reserved)(global-accept-transit-
customer)
id=1036450,66.74.106.59:1989,209.62.176.153:80(global-discard-reserved)(global-accept-transit-
customer)
...
```

Or

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

Google

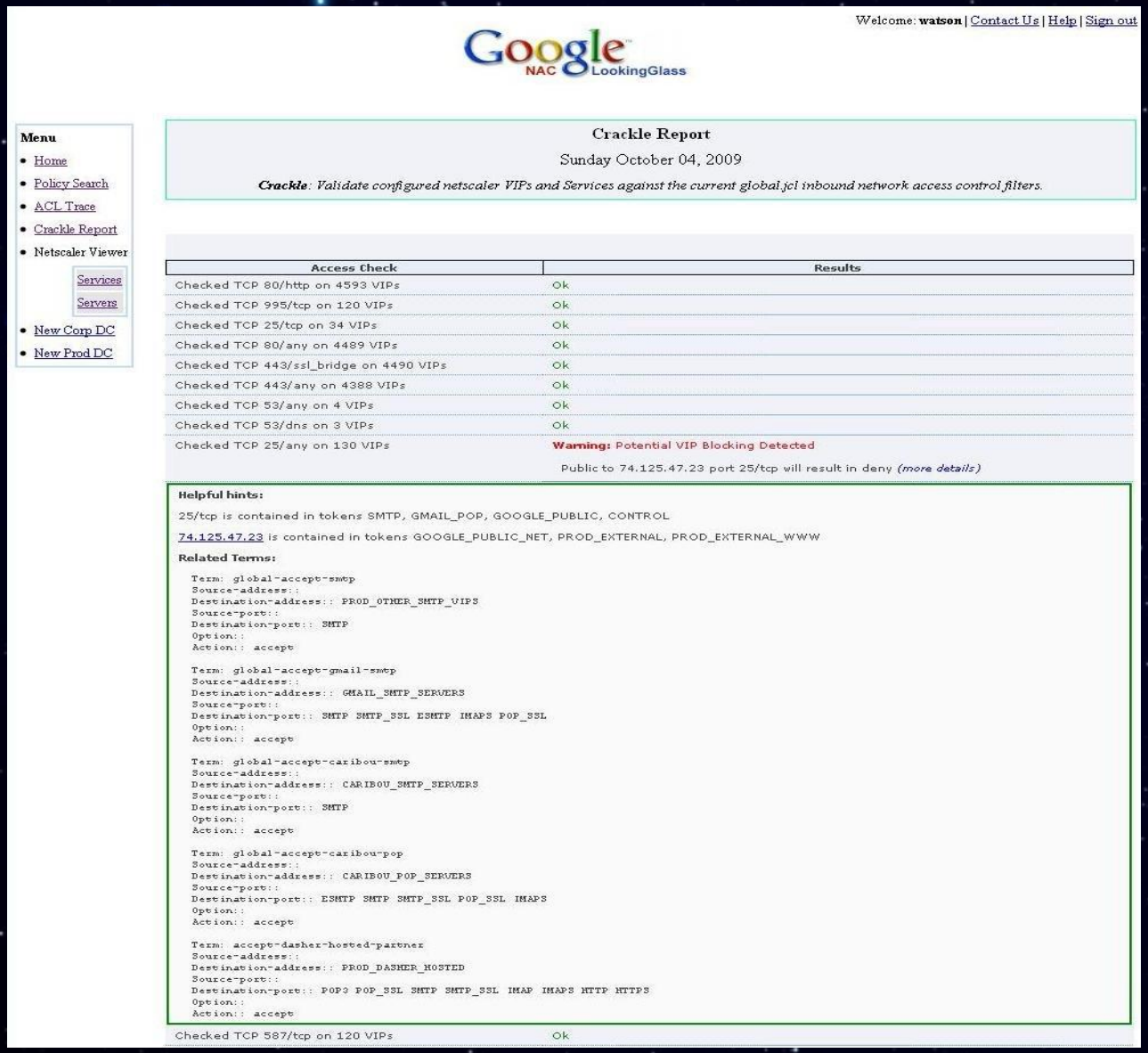
VIP Validation (aka Crackle)

- 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
  - such as inadvertently blocking POP3 to Gmail servers
- This has saved us from inadvertent outages on several occasions

*This tool is not 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.

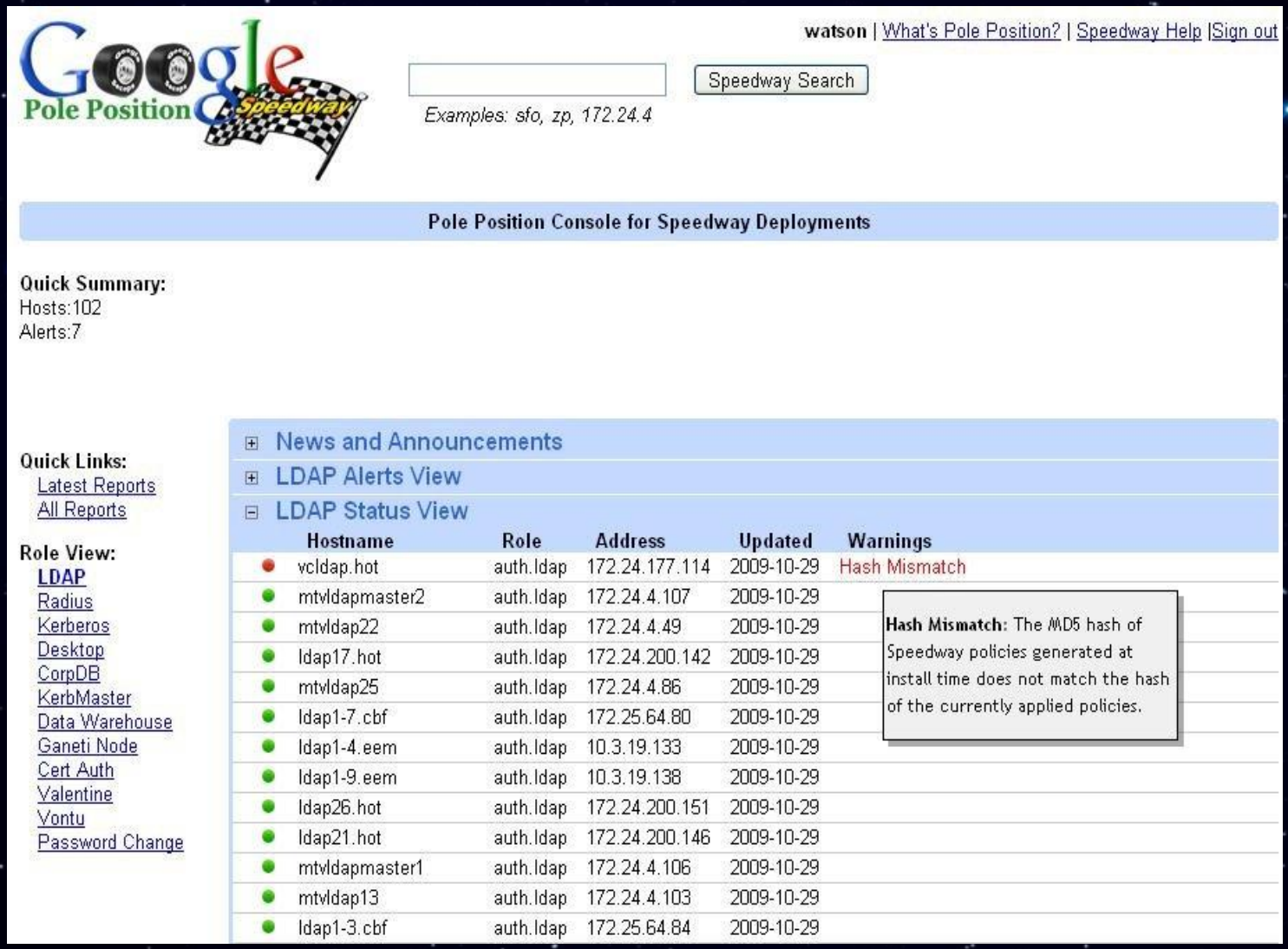
Google

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 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 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 programmatically
  - 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

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 to help other organizations in the management of complex network filtering

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.