Google Network Filtering Management

Thorsten Dahm
td@google.com

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

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 always 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 projects were disjointed, awkward or too "need" or platform specific

E Unum Pluribus (Out of one, many)

- What was needed was a common language to describe security policies and a standardized interconnect between the language and policy rules
- The 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 (ASNs, hosts, networks, groups of hosts and networks, services and service groups, etc.)
- Automate as much of the process as possible to reduce potential for human error, speed time to delivery, and reduce expertise needed to manage changes

- Write once, output many

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.

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

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

Naming Library

- Provides an easy way to lookup addresses and services based on token names
- We call them 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
- Multiple groups can maintain individual .net or .svc files

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

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

OFFICES = NYC_OFFICE SFO_OFFICE
```

Naming Service Definitions Format

```
BGP = 179/tcp

SSH = 22/tcp

NTP = 123/tcp 123/udp

PORT_RANGE = 8000-8050/tcp

DNS_UDP = 53/udp

DNS_TCP = 53/tcp

DNS = DNS_UDP DNS_TCP
```

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.

* 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 Definition Format

```
header {
   comment:: "edge input filter for example network."
   target:: juniper edge-inbound
   target:: cisco edge-inbound extended
}

term discard-spoofs {
   source-address:: RFC1918
   action:: deny
}

term permit-ipsec-access {
   source-address:: REMOTE_OFFICES
   destination-address:: VPN_HUB
   protocol:: 50
   action:: accept
}
```

example rendered - pt. 1

```
$ 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
```

example rendered - pt. 2

```
$ cat example.ipt
# Speedway Iptables INPUT Policy
# edge input filter for sample network.
#
# $Id:$
# $Date:$
# inet
-N discard-spoofs
-A discard-spoofs -p all -s 10.0.0.0/8 -j DROP
-A discard-spoofs -p all -s 172.16.0.0/12 -j DROP
-A discard-spoofs -p all -s 192.168.0.0/16 -j DROP
-A INPUT -j discard-spoofs
-N permit-ipsec-access
-A permit-ipsec-access -s 1.1.1.0/24 -d 3.3.3.3/32 -j ACCEPT
-A permit-ipsec-access -s 1.1.2.0/24 -d 3.3.3.3/32 -j ACCEPT
-A permit-ipsec-access -s 2.1.1.0/24 -d 3.3.3.3/32 -j ACCEPT
-A INPUT -j permit-ipsec-access
```

example rendered - pt. 3

```
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 {
               ...
```

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-group (extended with object-groups)

Iptables can generate 2 output formats:

- IPv4
- IPv6

Juniper Generator

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]

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 them 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.

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.

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

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.

Assurance / Validation Development

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

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

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

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

IPTables Assurance - aka Pole Position

- The md5 hash of Google "Speedway" deployments needs to match the hash of the currently applied policies
- 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

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.

Core Code Released to the Public

We have open-sourced the 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"). One of my colleagues registered the misspelling, then later noticed the error, but the correct spelling was already taken.
So, for efficiency(?) we have kept the name Capirca.

Questions?

Thorsten Dahm
td@google.com