Network Core Protection Best Practices

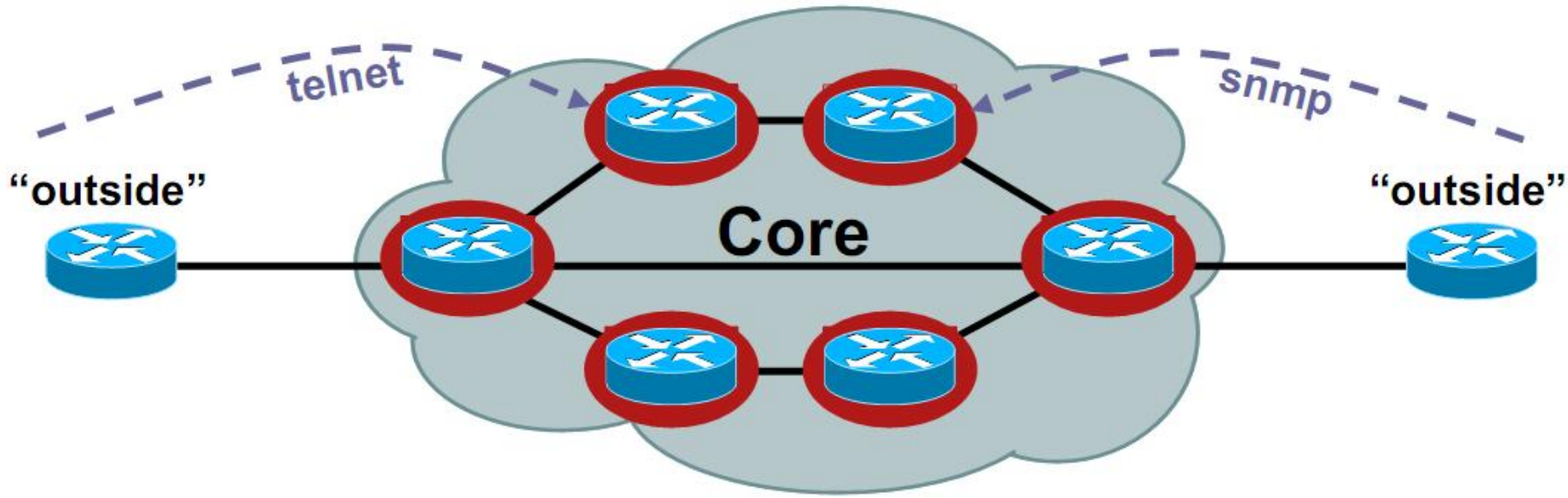
Thorsten Dahm
td@google.com
DENOG 1
05. November 2009

Agenda

- Protect your own infrastructure
- Overview of the current problems
- Methods to protect the own infrastructure:
  - traditional methods
  - protect the CPU
  - network hardening
- Not (!) part of this presentation: protect traffic which travels trough your network from customer to customer

Google™

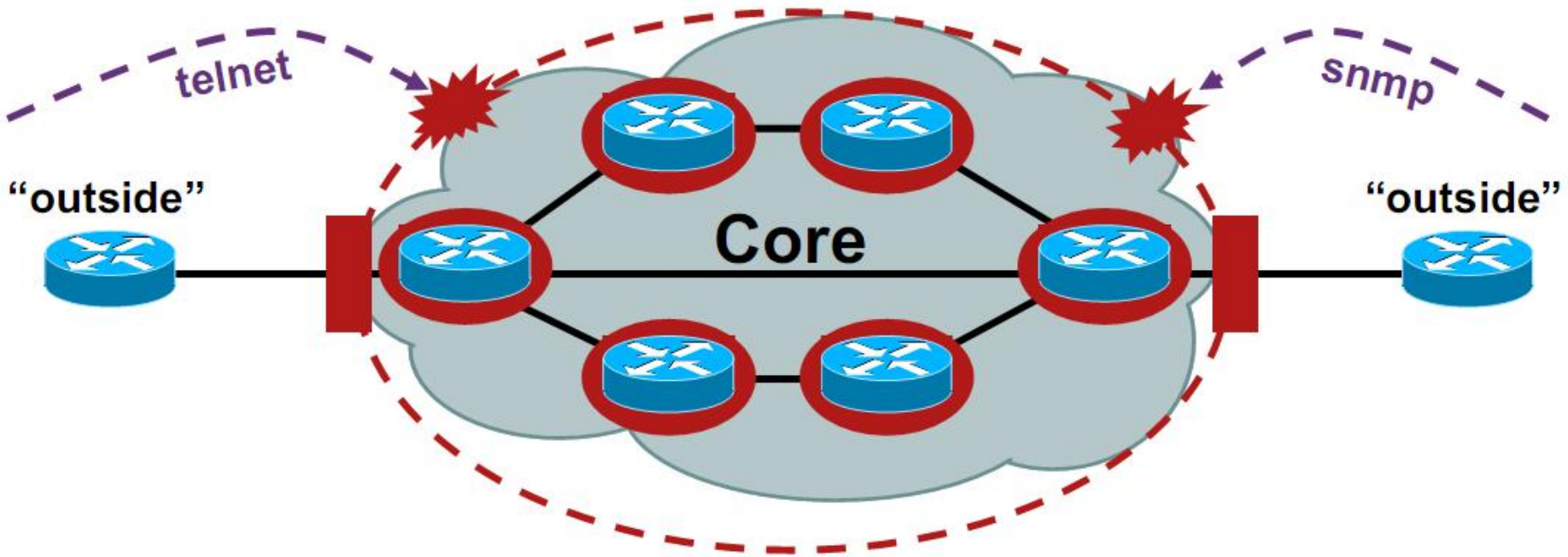
Traditional Network design



- all core routers are protected individually
- all routers are reachable from outside the own AS

Google™

Network hardening



- keep unwanted IP packets away from your core

Google™

The three security characteristics

- availability
- reliability
- integrity

-> our goal is to keep and maintain this three characteristics

Availability: protect the infrastructure

- security is a key factor in networks
- Internet changed from a trusted to an untrusted network
- do not trust any IP packet!
- develop trust by filters and policies
- fundamental: protect your own infrastructure
- must be part of every network design
- secure and stable network = basis of business operations

An adequate approach

- The presented features are useful, but have to fit into your own network design!
- Should never break your connectivity
- Do not implement all features at the same time
- Better: implement one feature that you understand
- Most important feature according to your assessment
- First roll out in the real network only in a controlled manner and in a limited part of the network
- Write down the lessons you learned
- Create documentation
- It's not a problem if it's taking one year and more!

Distinction of attacks

- Internal:
  - human error
  - internal attacker
- External:
  - worms
  - packet floods (what we looking at for now)
  - vulnerabilities
  - penetration
  - route hijacking
  - attacking services like DNS

Google™

Possible attacks

- control plane (ARP, BGP, OSPF)
- management plane (telnet, SSH, SNMP, NTP)
- filling up the queue between the interface and the RP
- overload of the RP input buffer
- overload the RP itself
- plus a few more

Protecting routers - Best Practices

- Many guidelines from cymru, NSA, ...
- Most of the proposals are out of the scope of this talk
- For every practice exists a reason (e. g. SSH version 2)
- Sometimes there is an interesting story behind this best practices :-)

Traditional methods of protecting routers

- turn off all services you don't need, like CDP
- turn off all features you don't need, like proxy ARP or ip redirects
- VTY ACLs
- SNMP community ACLs
- turn off SNMP rw (or use v3)
- AAA
- logging
- uRPF
- prefix filter
- MD5 for routing protocols
- ...

Control Plane Policing (CoPP)

- Cisco-version of a "real" loopback-interface
- not only permit/deny, also rate-limits available
- exists on all Cisco platforms
- same syntax everywhere
- flexible, can also deal with ARP etc.
- be careful with the decision what you want to deny or rate limit

Google™

CoPP: define ACLs (example)

- Critical - absolutely necessary (e. g. OSPF Hello)
- Important - daily work (e. g. SSH)
- Normal - needed, not essential needed
- Undesirable - “evil” oder “undesirable”

- Catch-All - all other IP traffic towards the RP which didn't get identified yet
- Default - all other non-IP traffic towards the RP which didn't get identified yet

Google™

CoPP - a configuration example

```
access-list 121 permit tcp host 10.1.1.2 eq bgp host 10.1.1.1 gt 1024
access-list 121 permit tcp host 10.1.1.2 gt 1024 host 10.1.1.1 eq bgp

class-map match-any CoPP-critical
      match access-group 121

policy-map CoPP
 class CoPP-critical
      police 5000000 2500 4470 conform-action transmit  exceed-action
      transmit

Router(config)# control-plane
Router(config-cp)# service-policy [input | output] <policy-map-name>
```

Monitoring CoPP

- show access-list
- show log (if the log keyword is used in the ACL)
- show policy-map control-plane
- SNMP Queries

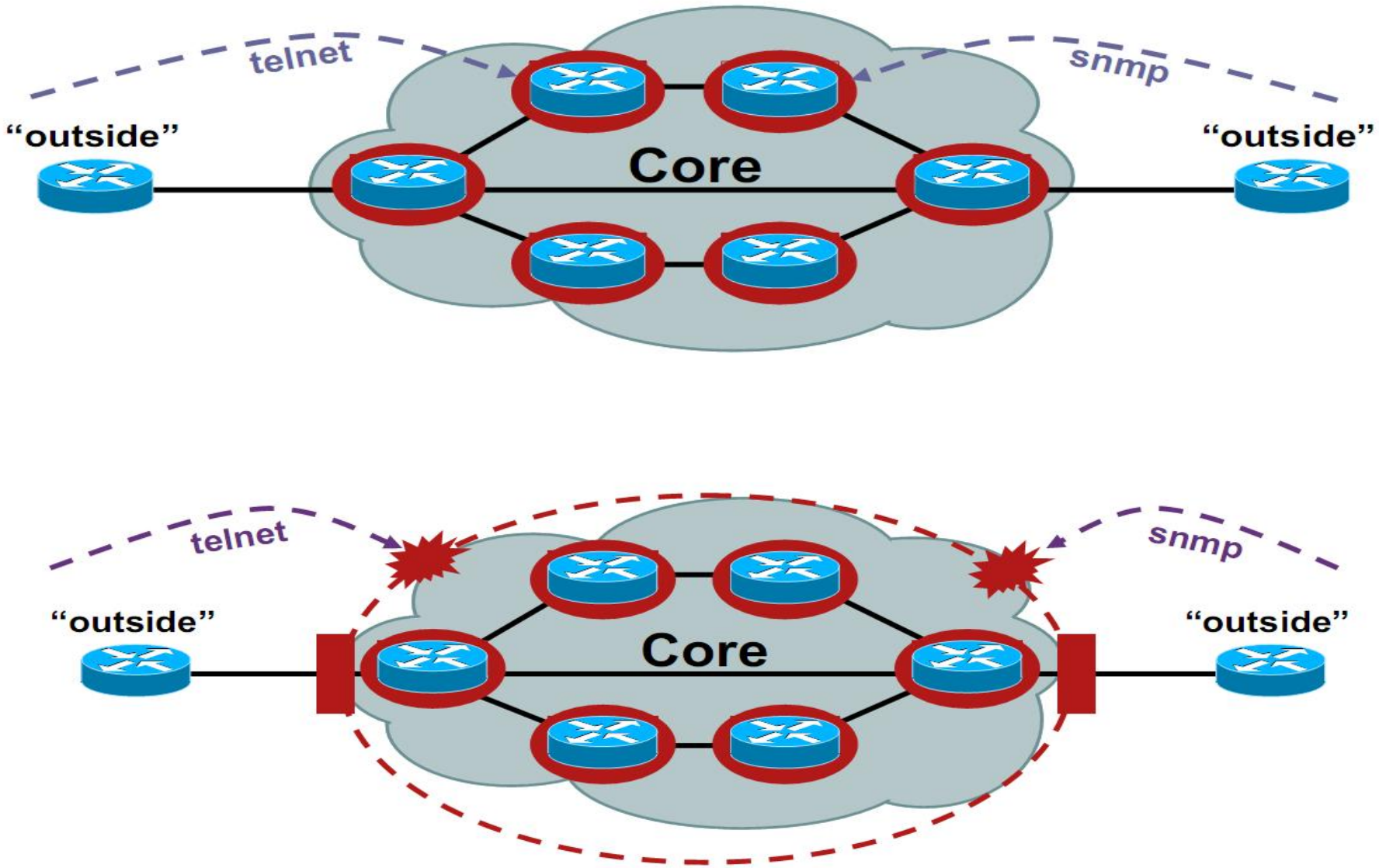
Google™

Network hardening

- In case of an DoS attack it is already too late if the packet reaches the router
  - CoPP helps in this case, but doesn't solve the problem
  - better: deny the undesirable packets at the network border

- One method to sove this problem:
  - Infrastructure ACLs

Comparison before - after



Infrastructure ACLs

- prerequisite: filter traffic to your own core routers
- create a list of protocols you need with a source outside your own AS and which have to reach your core routers (e. g. eBGP, IPSec, ...)
- the (preferably aggregated) address block of your core routers are the IP range you want to protect
  - summarization keeps your ACLs small
  - bad summarization makes your ACL less manageable

Infrastructure ACLs

- allow only the protocols and connections you really need
- should also do the anti-spoofing filtering:
  - RFC3330 defines IPv4 addresses for special use
  - deny your own IPs as a source from outside
  - deny RFC1918 addresses
  - deny multicast source addresses (224/4)
- have to allow transit
  - IP traffic which has to be forwarded trough the core routers must be permitted with "permit ip any any" in the end
- apply incoming at the ingress interface

Infrastructure ACLs

but ... RFC1918 addresses don't get routed in the Internet anyway, or do they?

Router#sh ip access <name>

100 deny ip 10.0.0.0 0.255.255.255 any (12 matches)
120 deny ip 169.254.0.0 0.0.255.255 any (15 matches)
130 deny ip 172.16.0.0 0.15.255.255 any (753 matches)
140 deny ip 192.168.0.0 0.0.255.255 any (24 matches)

Implementation step by step

- usually, you need just a few protocols
- even less of them will have a source IP from outside your own AS
- the necessary access will be definied by an ACL
- configure and test this ACL gradually

Google™

Define what is allowed

- every IP packet to the backbone must be classified
- NetFlow can help
- "log" keyword (be careful)
- investigate unexpected events with care
- No protocol / IP packet must be allowed that you can't explain!

Google™

An example

```
! Deny our internal space as a source of external packets
  access-list 101 deny ip core_CIDR_block any
! Deny src addresses of 0.0.0.0 and 127/8
  access-list 101 deny ip host 0.0.0.0 any
  access-list 101 deny ip 127.0.0.0 0.255.255.255 any
! Deny RFC1918 space from entering our AS
  access-list 101 deny ip 10.0.0.0 0.255.255.255 any
  access-list 101 deny ip 172.16.0.0 0.0.15.255 any
  access-list 101 deny ip 192.168.0.0 0.0.255.255 any
!Permit eBGP from outside
  access-list 101 permit tcp host peerA host peerB eq 179
  access-list 101 permit tcp host peerA eq 179 host peerB
! Deny all other access to infrastructure
  access-list 101 deny ip any core_CIDR_block
! Permit all transit traffic
  access-list 101 permit ip any any
```

Google™

Infrastructure ACLs - summarization

- Infrastructure ACLs are very useful if designed well and used everywhere
- being used since years from many ISP
- address summary is essential for an successful deployment
- Infrastructure ACLs have some weaknesses as well:
  - can collide with other (customer-) ACL for example

Google™

The End :-)

Questions?
td@google.com