A Secure BGP Implementation

Henning Brauer <henning@openbsd.org>

BGP - The Protocol

- Border Gateway Protocol, RFC 1771
- ISPs talk BGP to each other to announce reachability of their networks

BGP - The Protocol

- Networks are subsummarized into Autonomous Systems (AS)
- One ISP is typically one AS

BGP - The Protocol

- Network reachability is announced with so-called AS-Pathes, describing the path to the final network through intermediate ASes
- A BGP speaker usually announces directly connected networks, and prefixes with their pathes it learned from its neighbors
- An AS Path looks like "13237 174 3602 22512", listing the AS numbers we cross on the way to the destination, in this case, cvs.openbsd.org

BGP - Messages

- OPEN
  - Sent once at establishment of the tcp session. contains parameters such as the AS number.
- KEEPALIVE
  - Sent periodically to test wether the session is still alive.
- UPDATE
  - These messages carry the actual routing information.
- NOTIFICATION
  - Sent on fatal errors. After sending a notification the session is reset.

BGP - Existing Implementations

- Zebra: GPL, makes heavy use of cooperative threads. Suffers from loosing sessions while busy. Documentation and error messages in japanese or missing. Commercialized, thus mostly dead since about 2 years.
- Quagga: frustrated zebra users try to fix the worst bugs
- gated: became unfree, then died. Nothing really usable left.

BGP - Existing Implementations

- Cisco: proprietary, only works on their overpriced routers. Usually works ok, unless you happen to hit one of its countless bugs, or the tiny CPUs they use are swamped with work.

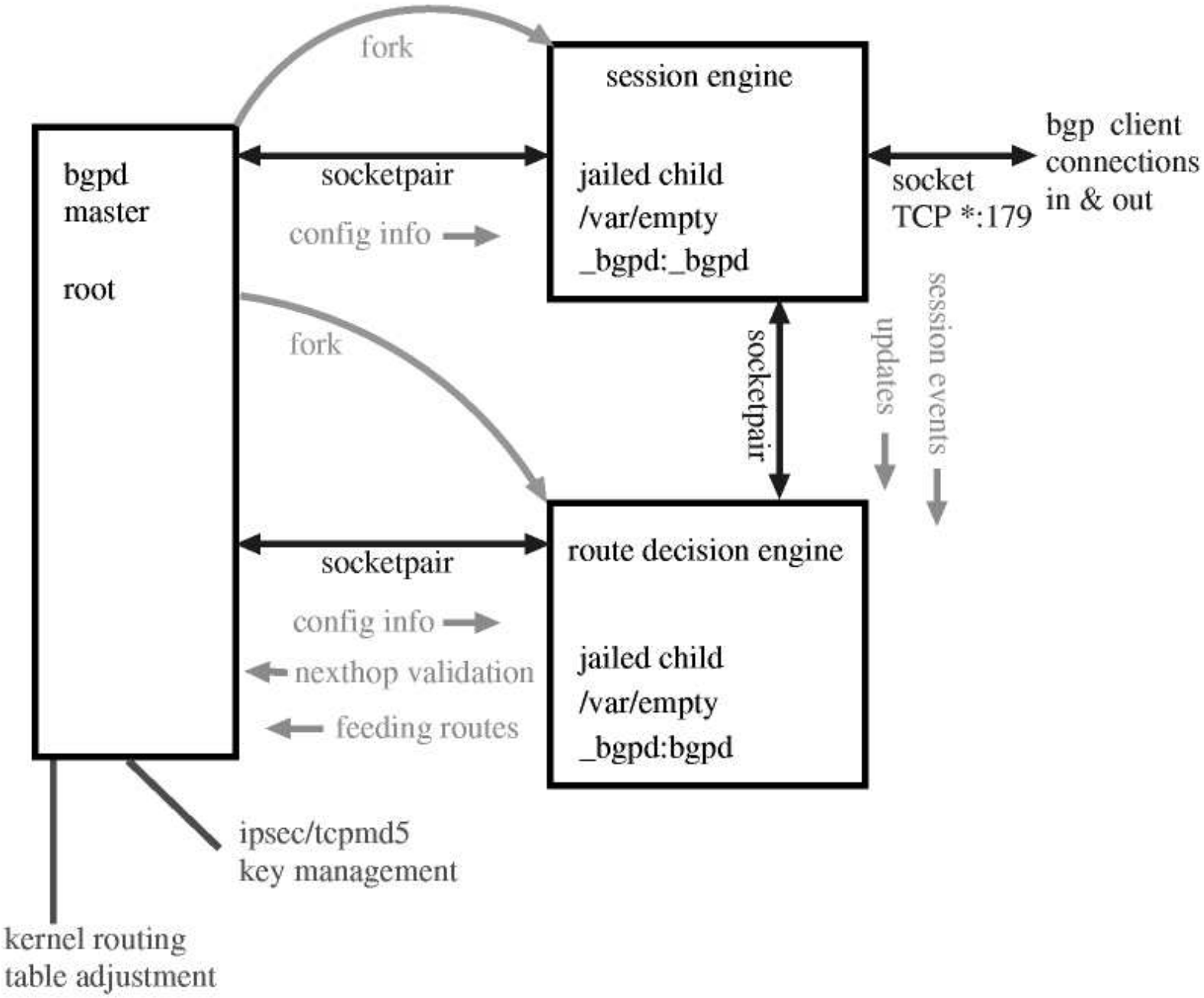
- Juniper's JunOS: apparently works ok, but not free either.

bgpd - Design Prerequisites

- Security. Use privilege separation.
- Don't loose sessions. There should be a fairly independent session engine.
- Performance and memory efficiency, of course.
- Well designed config and filter language.

bgpd - Design

- 3 processes
  - Session Engine (SE): manages bgp sessions
  - Route Decision Engine (RDE): holds the bgp tables, takes routing decisions
  - Parent: enters routes into the kernel, starts SE and RDE


fork
session engine
bgp client
connections
in & out
bgpd
master
root
socketpair
jailed child
/var/empty
_bgpd:_bgpd
socket
TCP *:179
config info
fork
socketpair
updates
session events
socketpair
route decision engine
config info
nexthop validation
feeding routes
jailed child
/var/empty
_bgpd:bgpd
ipsec/tcpmd5
key management
kernel routing
table adjustment

bgpd - Design

- Obviously, the Session Engine needs to be nonblocking, and use nonblocking sockets.
  - We need to handle all buffering ourselves.
- Invent an easy to use Buffer API
- For the internal messaging, invent an "imsg" API as well.
  - internal messaging is a core component in privilege separation
  - 40 message types now

bgpd - Session Engine

- Maintains a listening tcp socket
- Opens tcp connections to neighbors
- Negotiates parameters with neighbors via OPEN messages
- Once a session is established, it sends KEEPALIVE messages regularily, and receives ones from the neighbors

bgpd - Session Engine

- Finite State Machine for each neighbor

- States:
  - None: new, uninitialized neighbor, internal state
  - Idle: no connections accepted, none attempted
  - Connect: trying to open a tcp connection via connect(2)
  - Active: accepting tcp connection from neighbor, not trying to open one
  - OpenSent: tcp connection established, OPEN message sent
  - OpenConfirm: received OPEN message from peer, waiting for first KEEPALIVE
  - Established: fully established BGP session, KEEPALIVEs are exchanged regularily, routes are exchanged

bgpd - Session Engine

- Several Timers per neighbor:
  - IdleHoldTimer
    - A START event is generated when it expires. A session in Idle state transforms to Connect on expiration. Conditionally started when a session transforms to Idle state, depending on the cause of the session going back to Idle.
  - ConnectRetryTimer
    - Sessions in Active state transform to Connect and retry to open a tcp session on expiration. Started when a session transforms from Connect to Active.
  - HoldTimer
    - Started when a sessions reaches the Established state, and restarted on reception of a KEEPALIVE packet. When the HoldTimer expires, the session is assumed dead and is reset to Idle state.
  - KeepAliveTimer
    - Every time the KeepaliveTimer expires, a KEEPALIVE message is sent and the timer is restarted. Its start value is usually 1/3 of the HoldTime.

bgpd - Session Engine

- UPDATEs received from a neighbor are passed to the RDE.
- Outgoing UPDATEs are generated in the RDE and the SE just relays them.

bgpd - Session Engine

- Maintains a Unix-Domain socket for the bgpctl program
- very lightweight: typically under 1 MB RAM on i386
- runs as unprivileged user _bgpd, chroots to /var/empty

bgpd - Route Decision Engine

- Maintains the Routing Information Base (RIB)
  - prefix table
  - AS path table
- BGP Filters run here
- Calculates the best path per prefix
- Generates UPDATE messages as needed

bgpd - Route Decision Engine

- RIB Layout
  - Split into many tables
  - Heavily linked
  - Avoid table walks
- UPDATE messages are processed to completion
- Generated UPDATEs are queued to use piggy-back optimization
- RIB Table and sessions can be dumped to mrt files

bgpd - Route Decision Engine

- Memory efficent
  - 1 full view needs around 20 MB
  - 2 full views need around 25 MB
- Fast
  - Around 10s to load a full view on a PIII 1GHz
  - Less than 5s to dump a full view to another router
- Runs as unprivileged user _bgpd, chroots to /var/empty

bgpd - Parent process, kernel interface

- Responsible for getting the routes into the kernel
- Maintains its own copy of the kernel routing table
- Fetches the kernel routing table and interface list on startup
- Does nexthop validation for the RDE

bgpd - Parent process, kernel interface

- Listens to the routing socket
  - Internal view of the kernel routing table is held in sync
    - If you fiddle with the routing table manually, we notice that and cope with it
  - Internal list of interfaces and their status is kept in sync
    - We know about interfaces' link status and use it for nexthop verification
    - Yes, we notice when you pull the cable!
- We don't need periodic nexthop table walks

bgpd - Parent process, kernel interface

- The internal view of the routing table can be coupled and decoupled from the kernel
  - Damn fast! With a full table (about 140000 entries), less than 3 seconds on a PIII 750.
- Needs about 5 MB in full-mesh configurations

bgpd - tcp md5 signatures

- bgp sessions are not really authenticated - just IP based access control
- An attacker could send a bgp notification message with a faked source address, resetting the connection -> DoS

bgpd - tcp md5 signatures

- RFC 2385 defines tcp md5 signatures
- An md5 hash of parts of the header and a shared secret is added to the tcp header and verified on the receiving side
  - (unless you happen to run FreeBSD, they don't bother verifying the signatures)
- Attacker has to know the shared secret

bgpd - tcp md5 signatures

- Very old code for tcp md5 signatures existed, but didn't work. We used it as starting point.
- We implemented tcp md5 signatures as Security Association within the IPsec framework
- bgpd got a pfkey interface to interact with the IPsec framework
- tcp md5sig is extremly easy to configure, works with ciscos and junipers, too: USE IT!

bgpd - tcp md5 signatures

- Keep in mind that tcp md5 sigs are rather weak
- Take care for the key length - use at least 12 bytes
- Make sure to read RFC 3562, "Key Management Considerations for the TCP MD5 Signature Option"

bgpd - ipsec integration

- As we had the pfkey interface already, it was not too hard to do real IPsec
  - bgpd loads the SAs into the kernel
  - bgpd sets up the flows
- Juniper can do static-keyed IPsec as well, we're compatible.
- Cisco cannot, of course
  - (could cause CPU load after all!)

bgpd - ipsec integration

- We can use isakmpd to do the keying for us
  - keys are changed on a regular basis
- bgpd asks the kernel for an unused pair of SPIs and uses them
- bgpd sets up the flows
  - it knows the endpoints and ports already
- isakmpd only needs to handle the keying
  - almost NO configuration needed!
  - copy key files (generated at first boot on OpenBSD 3.6) over
  - run "isakmpd -Ka"

bgpd - pf integration

- The BGP protocol is an efficient way to distribute lists of network prefixes, so we integrated bgpd with our pf packet filter

- bgpd can add prefixes learned from neighbors into a pf table
  - prefixes are selected using the bgpd filter language
  - tables use a radix tree, very fast even with lots of entries

- pf tables can be used for pretty much anything:
  - packet filtering
  - redirection to spamd (BGP distributed spam blacklists)
  - QoS processing

bgpd - configuration

- Split into 5 sections
  - Macro definitions - just like in pf
  - Global settings
  - Networks to announce
  - Neighbor definitions
  - Filter

bgpd - macros, global config, networks

```
#macros
peer1="10.0.0.2"
peer2="10.0.0.3"
myip="127.0.0.1"

# global configuration
AS 65001
router-id $myip
listen on $myip
holdtime 180
holdtime min 3
fib-update no

# networks we announce
network 10/8
network 192.168.2/23
```

bgpd - neighbor definition

```
neighbor 10.0.1.0 {
        remote-as       65003
        descr           upstream
        multihop        2
        local-address   10.0.0.8
        passive
        holdtime        180
        holdtime min    3
        announce        self
        tcp md5sig key  deadbeef
}
```

- Very cool: the annnounce keyword
  - none: don't announce any networks
  - self: announce only our own networks
  - all: announce everything we know
  - default-route: announce a default-route and nothing else
- On cisco/zebra you need filters for this

bgpd - neighbor groups

```
group "peering AS65002" {
        remote-as       65002
        passive
        holdtime        180
        holdtime min    3

        neighbor $peer1 {
                descr    "AS 65001 peer 1"
                announce self
                tcp md5sig password mekmitasdigoat
        }
        neighbor $peer2 {
                descr    "AS 65001 peer 2"
                announce all
        }
}
```

bgpd - ipsec configuration, static keying

```
neighbor 10.2.1.1 {
        remote-as 65023
        local-address 10.0.0.8
        ipsec esp in  spi 10 \
                sha1 0a4f1d1f1a1c4f3c9e2f6f0f2a8e9c8c5a1b0b3b \
                aes 0c1b3a6c7d7a8d2e0e7b4f3d5e8e6c1e
        ipsec esp out spi 12 \
                sha1 0e9c8f6a8e2c7d3a0b5d0d0f0a3c5c1d2b8e0f8b \
                aes 4e0f2f1b5c4e3c0d0e2f2d3b8c5c8f0b
}
```

bgpd - ipsec configuration, using IKE

```
neighbor 10.2.1.1 {
        remote-as 65023
        local-address 10.0.0.8
        ipsec esp ike
}

neighbor 10.2.1.2 {
        remote-as 65024
        local-address 10.0.0.8
        ipsec ah ike
}
```

filter language

```
# filter out prefixes longer than 24 or shorter than 8 bits
deny from any
allow from any prefixlen 8 - 24

# do not accept a default route
deny from any prefix 0.0.0.0/0

# filter bogus networks
deny from any prefix 10.0.0.0/8 prefixlen >= 8
deny from any prefix 172.16.0.0/12 prefixlen >= 12
deny from any prefix { 192.168.0.0/16 169.254.0.0/16 } \
        prefixlen >= 16
deny from any prefix 192.0.2.0/24 prefixlen >= 24
deny from any prefix { 224.0.0.0/4 240.0.0.0/4 } prefixlen >= 4
```

filter language

```
allow from $someuplink transit-as { $dfn }     set localpref 114
allow from $someuplink source-as  { $viag }    set localpref 112
allow from $someuplink transit-as { $telekom } set localpref 110

allow from group peerings set localpref 200

allow to $someuplink set community 13129:1911
allow to group uplinks set prepend-self 1
```

filter language

- Last match

- Rule consists of 3 parts:
  - Action: allow, deny or match
  - Match: based on prefix, prefixlen or AS path
  - Set: add prepends, modify localpref, metric etc

bgpctl

- Client connecting to bgpd via unix domain socket
  - query runtime information
  - reload configuration
  - (de-)couple kernel routing table
  - take specific sessions up/down

bgpctl

Neighbor	AS	MsgRcvd	MsgSent	OutQ	Up/Down	State/PrefixRcvd
192.168.133.46	64639	4333	4332	0	3d00h10m	1/100
192.168.133.47	64686	33618	33585	0	5d19h57m	12/100
192.168.133.85	64847	6768	6756	0	1d18h28m	3/100
192.168.133.86	64847	8693	8689	0	6d00h46m	3/100
192.168.133.49	64918	9096	9582	0	6d15h40m	80/200
192.168.133.28	64586	9113	9581	0	2d23h00m	1/100
192.168.133.65	64902	19158	19161	0	6d15h40m	2/100
192.168.133.48	64956	35310	9588	0	1d15h41m	45/100
192.168.133.95	64727	9585	9582	0	6d15h40m	5/100
192.168.133.22	65126	9589	9585	0	6d13h28m	1/100
192.168.133.17	64562	9405	9582	0	6d15h40m	142
192.168.133.173	64785	361006	9582	0	6d15h40m	143006
192.168.133.169	64562	77441	9582	0	6d15h40m	35987

bgpctl

```
<henning@cr10>  $ bgpctl show fib connected static
        flags: * = valid, B = BGP, C = Connected, S = Static
               N = BGP Nexthop reachable via this route

        flags destination           gateway
        *C    80.86.162.24/30       link#2
        *SN   80.86.164.16/32       80.86.162.25
        *S    80.86.181.0/24        80.86.183.4
        *S    80.86.182.0/23        80.86.183.4
        *C    80.86.183.0/29        link#5
        *C    80.86.183.16/28       link#7
        *S    80.86.183.30/32       127.0.0.1
        *S    81.209.180.0/22       80.86.183.4
        *S    81.209.196.0/22       80.86.183.4
        *C    127.0.0.1/8           link#0
        *S    127.0.0.1/32          127.0.0.1
        *S    192.168.214.0/24      80.86.183.17
        *SN   212.20.158.0/30       212.20.158.201
        *C    212.20.158.200/29     link#3
     [ ... ]
```

bgpctl

```
<henning@cr10>  $ bgpctl s nei 10.0.0.16
BGP neighbor is 10.0.0.16, remote AS 13237
 Description: lnc
  BGP version 4, remote router-id 10.0.0.16
  BGP state = Established, up for 02:53:52
  Last read 00:00:15, holdtime 90s, keepalive interval 30s
  Neighbor capabilities:
    Multiprotocol extensions: IPv4 Unicast
    Route Refresh

  Message statistics:
                  Sent       Received
  Opens              1              1
  Notifications      0              0
  Updates            1          78260
  Keepalives       348              1
  Route Refresh      0              0
  Total            350          78262

  Local host:          10.0.0.26, Local port:    179
  Remote host:         10.0.0.16, Remote port:  2667
```

bgpctl

```
<henning@cr10>  $ bgpctl s nei 10.0.0.16 timers
BGP neighbor is 10.0.0.16, remote AS 13237
 Description: lnc
  BGP version 4, remote router-id 10.0.0.16
  BGP state = Established, up for 02:57:16
  Last read 00:00:18, holdtime 90s, keepalive interval 30s
  Neighbor capabilities:
    Multiprotocol extensions: IPv4 Unicast
    Route Refresh

  IdleHoldTimer:       not running          Interval:    30s
  ConnectRetryTimer:   not running          Interval:   120s
  HoldTimer:           due in 00:01:12      Interval:    90s
  KeepaliveTimer:      due in 00:00:30      Interval:    30s

  Local host:          10.0.0.26, Local port:    179
  Remote host:         10.0.0.16, Remote port:  2667
```

bgpctl

```
<henning@cr10>  $ bgpctl sh nex
Nexthop                 State
80.86.164.16            valid
213.128.128.6           valid
212.20.158.2            valid
```

bgpctl

```
lyn # bgpctl reload
reload request sent.

lyn # bgpctl fib couple
couple request sent.

lyn # bgpctl fib decouple
decouple request sent.

lyn # bgpctl nei 213.128.133.5 up
request sent.

lyn # bgpctl nei 213.128.133.5 down
request sent.
```

bgpd - status quo

- Very stable
- In use at quite some sites, including setups with many many many many many many many many many peers.
  - Quite some operators mail me, expressing that they are very happy with bgpd's performance, reliability and ease of use
    - That makes me happy ;)
- Some statistics...
  - bgpd: 16879 lines of code
  - bgpctl: 1356 lines of code
  - manpages: 2582 lines

bgpd - 6 months old future plans

- IPv6 transport
  - was added somewhen in April
- Multiprotocol support, including IPv6 (RFC 2858)
  - partly done. kroute6 is a major PITA, I won't do it.
- Route refresh (RFC 2918)
  - implemented
- Route flap dampening (RFC 2439)
  - claudio's working on that

bgpd - evil future plans

- Give pf access to some information from bgpd
- allow for freetext labels attached to a route
  - 32 bytes we can use to attach arbitary information
  - implemented in route(8) and the kernel routing table
  - pf can't filter based on the label yet, and bgpd can't set it - will be there soonish...
- This is really evil:

```
pass in proto tcp keep state route-label swisscom queue reallyslow
```

Thanks

- Claudio Jeker <claudio@openbsd.org>, who's writing most of the RDE and helped enormously with getting these slides to you in time
- Andre Oppermann <andre@freebsd.org>, who designed the RDE with claudio and is paying most of his work on bgpd
- Theo de Raadt for kicking my lazy butt so that I eventually started bgpd after thinking about it for at least 2 years, helping with basic design and many many many McNally's we had while discussing bgpd
- Wim Vandeputte, for his continued support and beer supply
  - (we're far from your house this time, your fridge is safe)

The unavoidable last page, 2004 edition

- We have cool shirts and posters for sale outside, as well as OpenBSD CDs
- Money is running out, donations can be made at http://www.openbsd.org/donations.html or outside at our booth
- Beer donations for the hackers are always welcome!