Full-mesh IPsec network

10 Dos and 500 Don’ts

HostedGraphite

$ whoami

- Fran Garcia
- SRE @hostedgraphite
- “Break fast and move things”
- Absolutely no networking/cryptography background
- No, seriously, totally unqualified to give this talk

The questions I hope to answer

What’s this IPsec thing anyway?

Why would I want to use it in a full-mesh?

How many ways have you screwed up its implementation?

Prelude: our requirements

- Can’t depend on fancy networking gear
- Cluster spans multiple locations/providers
- We don’t trust the (internal) network!
- Must be efficient enough not to become a bottleneck!
- Simple security model (no complex/dynamic firewall rules)
- Can be implemented reasonably quickly

TL;DW

“IPsec is awful”

(trust me on this one)

IPsec for the super-impatient

So what’s this IPsec thing anyway?

Not a protocol, but a protocol suite

Open standard, which means lots of options for everything

66 RFCs linked from wikipedia page!

What IPsec offers

At the IP layer, it can:

- Encrypt your data (Confidentiality)
- Verify source of received messages (Data-origin authentication)
- Verify integrity of received messages (Data Integrity)

Offers your choice of everything to achieve this

Choices, choices everywhere

What protocol?

- Authentication Header (AH): Just data integrity/authentication*
- Encapsulating Security Payload (ESP): Encryption + integrity/auth (optional)
- AH/ESP

(TL;DR - You probably want ESP)

*Legend says AH only exists to annoy Microsoft

Second choice... Tunnel or Transport mode?

	Encapsulates header	Encapsulates payload	Works for host-to-host	Works for site-to-site
Tunnel Mode	YES	YES	YES	YES
Transport Mode	NO	YES	YES	NO

*Transport mode might incur a slightly smaller overhead and be a bit simpler to set up

IPsec: What's a SP (security policy)?

Consulted by the kernel when processing traffic (inbound and outbound)

> "From host A to host B use ESP in transport mode"
>
> "From host C to host D's port 443 do not use IPsec at all"

Stored in the SPD (Security Policy Database) inside the kernel

IPsec: What's a SA (Security Association)?

Secured unidirectional connection between peers:

- So need two for bidirectional communication (hosta->hostb, hostb->hosta)

Contains keys and other attributes like its lifetime, IP address of peer...

Stored in the SAD (Security Association Database) inside the kernel

IKE? Who’s IKE?

“Internet Key Exchange”

Negotiate algorithms/keys needed to establish secure channel between peers

A key management daemon does it in user space, consists of 2 phases

IPsec: IKE Phase 1

Lives inside the key management daemon (in user space)

Hosts negotiate proposals on how to authenticate and secure the channel

Negotiated session keys used to establish actual (multiple) IPsec SAs later

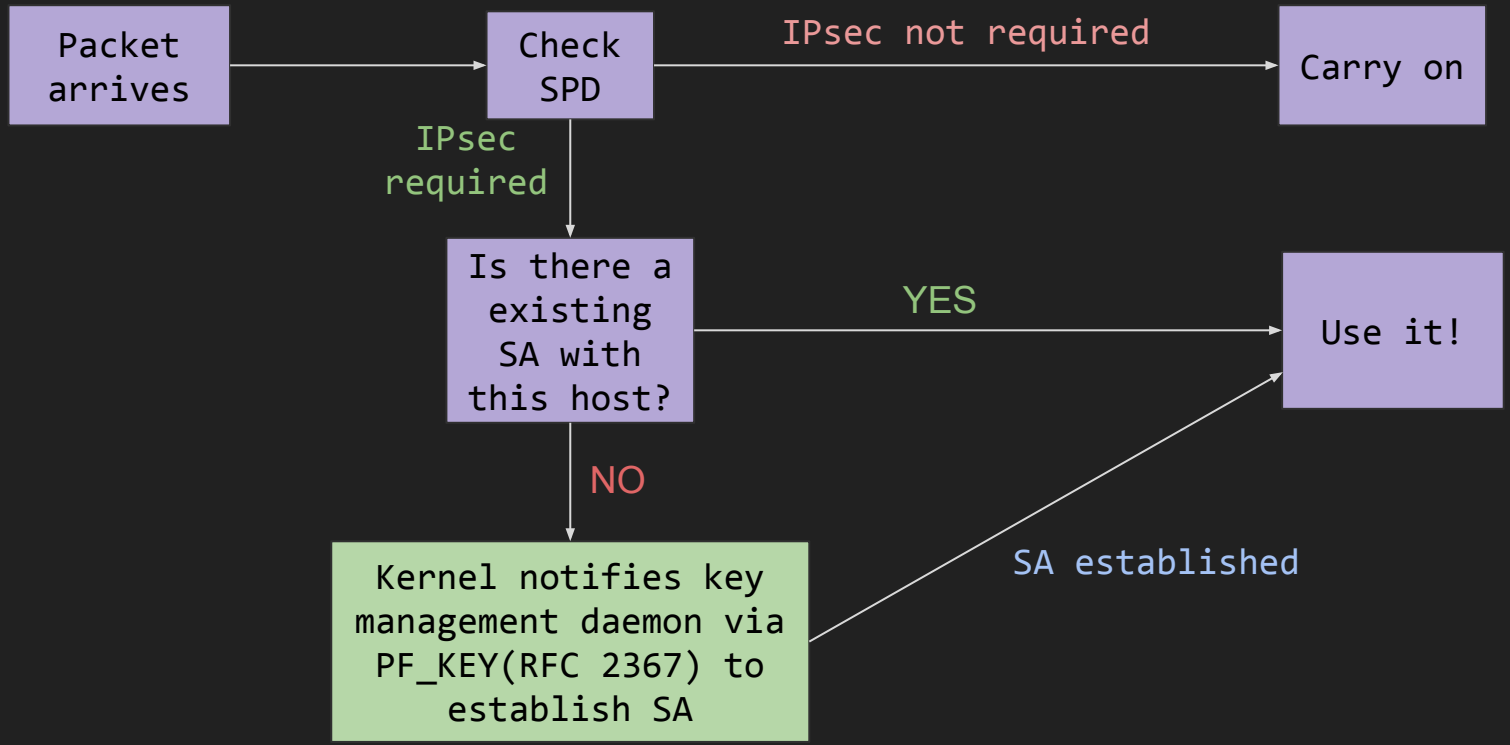
IPsec: Phase 2

Negotiates IPsec SA parameters (protected by IKE SA) using phase 1 keys

Establishes the actual IPsec SA (and stores it in SADB)

Can renegotiate when close to end of lifetime

Life of an IPsec packet



So what has IPsec ever done for us?

Encryption happens inside the kernel, so it’s fast!

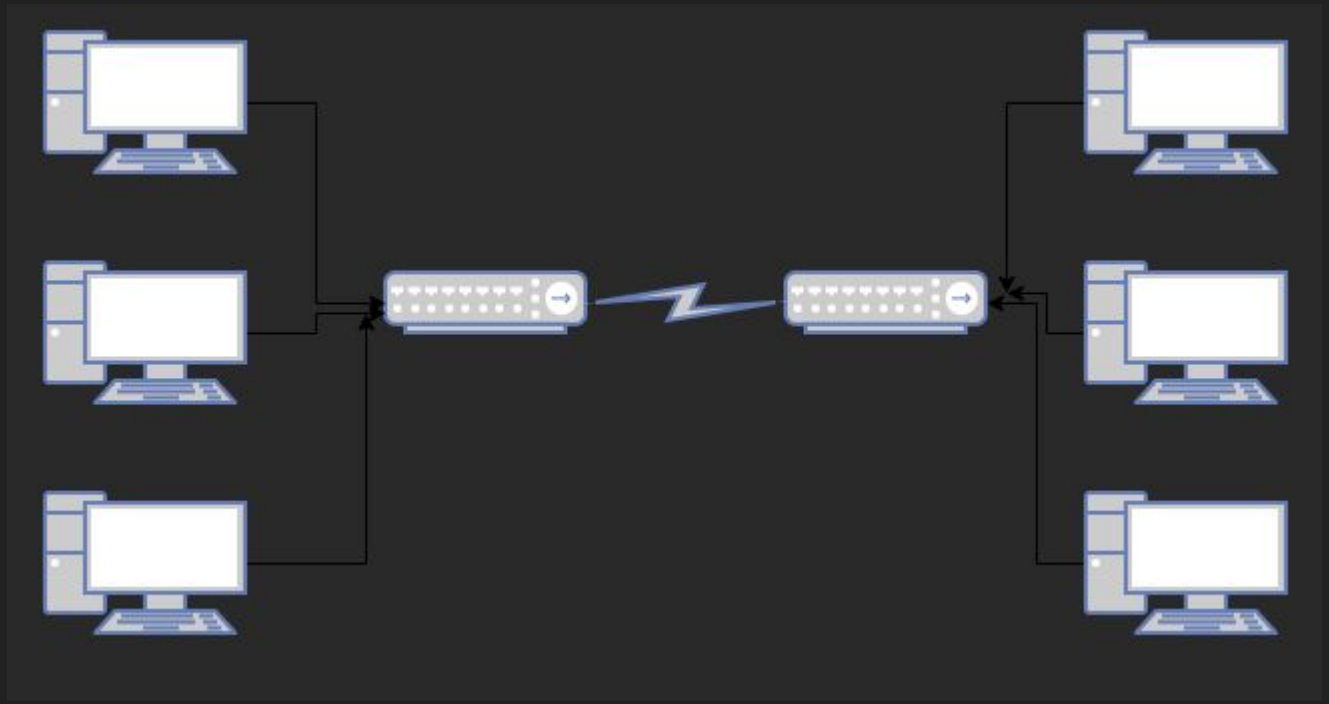
Using the right algorithms/settings it can be fairly secure

It’s a standard, so there are good practices to use it securely

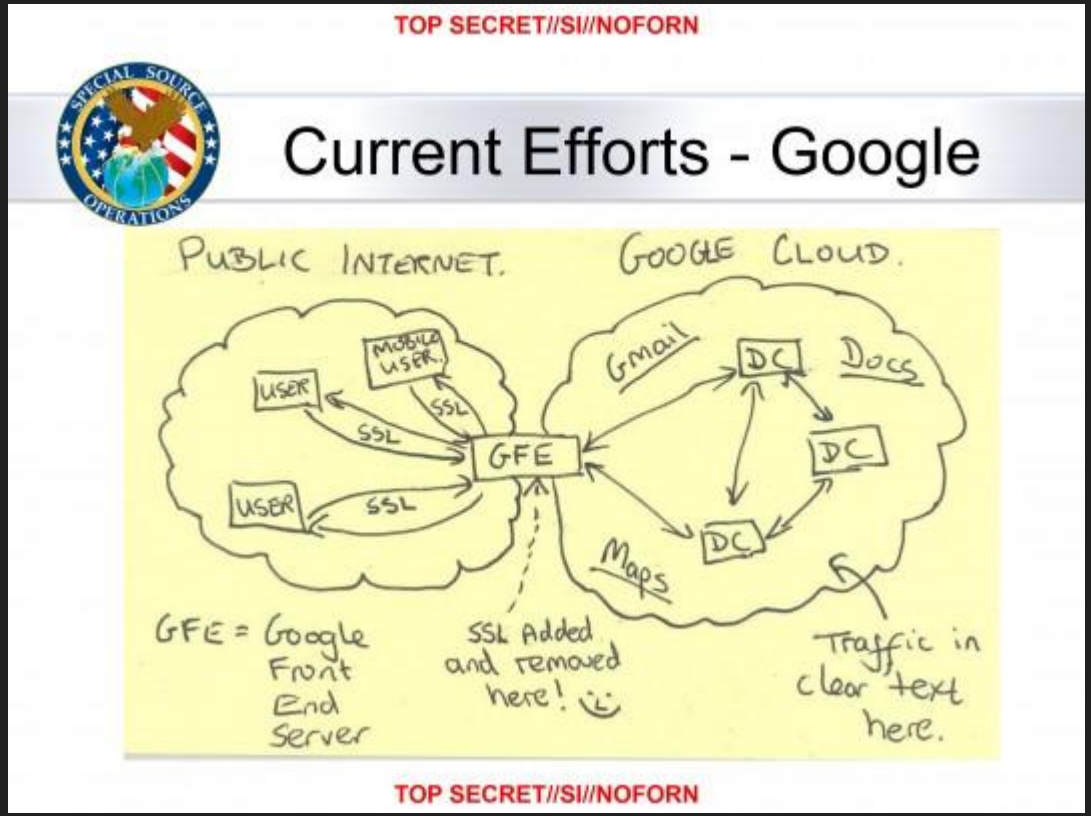
Very flexible, which is useful if you have:

- Hardware distributed across different datacenters/providers
- No real control over your network infrastructure

Traditional model vs Full Mesh



Why a full mesh?



Hosted Graphite does IPsec!

Our migration to IPsec

Big time constraints: previous VPN was unreliable and preventing us from scaling

We had trouble finding reports of people using IPsec in the same way*…

...So we had to improvise a bit.

After careful planning and testing we rolled it out to our production cluster...

* Notable exception: pagerduty's Doug Barth at Velocity 2015
http://conferences.oreilly.com/velocity/devops-web-performance-2015/public/schedule/detail/41454

WORST

MIGRATION

EVER

WORST. MIGRATION. EVER

Migration attempt resulted in multi-day incident:

http://status.hostedgraphite.com/incidents/gw2v1rhm8p5g

Took two days to stabilize, a full week to resolve the incident.

Lots of issues not found during testing

IPsec rollout aftermath

Back to drawing board, came up with another plan

Spent almost 3 months slowly rolling it out and fixing bugs:

- Also known as “the worst three months of my life”
- Big team effort, everybody pitched in

Still worth it, things are stable now and we’ve learned a lot

Our stack: present day

Our IPsec stack: present day

Hundreds of hosts using ESP in transport mode (full-mesh)

Several clusters, isolated from each other

Using ipsec-tools with racoon as key management daemon

Our config: iptables

```
# Accept all IKE traffic, also allowing NAT Traversal (UDP 4500)
-A ufw-user-input -p udp --dport 500 -j ACCEPT
-A ufw-user-input -p udp --dport 4500 -j ACCEPT


# Allow  all ESP traffic, if it has a formed IPsec SA we trust it
-A ufw-user-input -p esp -j ACCEPT
```

Our config: Security Policies (`/etc/ipsec-tools.conf`)

Node1 = 1.2.3.4 Node2 = 5.6.7.8

On node1:

```
# require use of IPsec for all other traffic with node2
spdadd 1.2.3.4 5.6.7.8 any -P out ipsec esp/transport//require;
spdadd 5.6.7.8 1.2.3.4 any -P in ipsec esp/transport//require;
```

On node2:

```
# require use of IPsec for all other traffic with node1
spdadd 5.6.7.8 1.2.3.4 any -P out ipsec esp/transport//require;
spdadd 1.2.3.4 5.6.7.8 any -P in ipsec esp/transport//require;
```

Our config: Security Policies (`/etc/ipsec-tools.conf`)

What about management hosts (configuration management, syslog…)?

Node1 = 1.2.3.4 PuppetMaster = 5.6.7.8

On node1:

```
# Only require IPsec for port 8140 on the puppet master
spdadd 1.2.3.4 5.6.7.8[8140] any -P out ipsec esp/transport//require;
spdadd 5.6.7.8[8140] 1.2.3.4 any -P in ipsec esp/transport//require;
```

Everything else will get dropped by the firewall

Our config: Security Policies (`/etc/ipsec-tools.conf`)

```
# Exclude ssh traffic:
spdadd 0.0.0.0/0[22] 0.0.0.0/0 tcp -P in prio def +100 none;
spdadd 0.0.0.0/0[22] 0.0.0.0/0 tcp -P out prio def +100 none;
spdadd 0.0.0.0/0 0.0.0.0/0[22] tcp -P in prio def +100 none;
spdadd 0.0.0.0/0 0.0.0.0/0[22] tcp -P out prio def +100 none;


# Exclude ICMP traffic (decouple ping and the like from IPsec):
spdadd 0.0.0.0/0 0.0.0.0/0 icmp -P out prio def +100 none;
spdadd 0.0.0.0/0 0.0.0.0/0 icmp -P in prio def +100 none;
```

Our config: racoon (/etc/racoon.conf)

Phase 1:

```
remote anonymous {
    exchange_mode main;
    dpd_delay 0;
    lifetime time 24 hours;
    nat_traversal on;
    proposal {
        authentication_method pre_shared_key;
        dh_group modp3072;
        encryption_algorithm aes;
        hash_algorithm sha256;
    }
}
```

Our config: racoon (/etc/racoon.conf)

Phase 2:

```
sainfo anonymous {
    pfs_group modp3072;
    encryption_algorithm aes;
    authentication_algorithm hmac_sha256;
    compression_algorithm deflate;

    lifetime time 8 hours;
}
```

10 DOS AND 500 DONT'S

Disclaimer:

(We couldn't find 10 dos)

Don’t use ipsec-tools/racoon! (like we did)

Not actively maintained (Last release on early 2014)

Buggy

But the only thing that worked for us under time/resource constraints

LibreSwan seems like a strong alternative, but haven’t tested it in production

Don’t blindly force all traffic to go through IPsec

Account for everything that needs an exception:

- SSH, ICMP, etc

You’ll need to be able to answer these two questions:

- “Is the network broken?”
- “Is IPsec broken?”

“Yo dawg, I heard you like encrypted traffic…”

If migrating from an existing VPN, make sure to exclude it from IPsec traffic

During our initial rollout our SPs forced our previous VPN’s traffic through IPsec…

… Which still wasn’t working reliably enough…

… Effectively killing our whole internal network

Don’t just enable DPD… without testing

What’s DPD?

- DPD: Dead Peer Detection (RFC3706)
- Liveness checks on Phase 1 relationships
- If no response to R-U-THERE clears phase 1 and 2 relationships…

Sounds useful but test it in your environment first:

- racoon’s implementation is buggy!

“The trouble with DPDs”

In our case, enabling DPD results in 100s of SAs between two hosts:

- Every failed DPD check resulting in extra SAs

Combination of factors:

- Unreliable network
- Bugs in racoon

We ended up giving up on DPD

Don’t just disable DPD either

DPD can be legitimately useful

Example: What happens when rebooting a host?

Other nodes might not realise their SAs are no longer valid!

DPD: Rebooting hosts

bender's SAD:

```
5.6.7.8 -> 1.2.3.4 (spi: 0x01)

1.2.3.4 -> 5.6.7.8 (spi: 0x02)
```

flexo's SAD:

```
5.6.7.8 -> 1.2.3.4 (spi: 0x01)

1.2.3.4 -> 5.6.7.8 (spi: 0x02)
```

These are two happy hosts right now…

bender -> flexo (using spi `0x02`) traffic is received by flexo

flexo -> bender (using spi `0x01`) traffic is received by bender!

… But let's say we reboot bender!

DPD: Rebooting hosts

bender's SAD:

~~5.6.7.8 -> 1.2.3.4 (spi: 0x02)~~

~~1.2.3.4 -> 5.6.7.8 (spi: 0x01)~~

flexo's SAD:

```
5.6.7.8 -> 1.2.3.4 (spi: 0x02)
1.2.3.4 -> 5.6.7.8 (spi: 0x01)
```

bender's SADB is now empty

flexo->bender traffic (using spi 0x02) will be broken until:

- bender->flexo traffic (like ping) forces establishment of new SAs
- The SAs on flexo's side expire

DPD: Possible workarounds (on bender's side)

Easy to detect in racoon logs:

`"1.2.3.4 can't start the quick mode, there is no ISAKMP-SA"`

Obvious but risky fix:

- Patch racoon. Delete any existing SAs with that host.

Terrible but trivial to implement workaround:

- Detect it happened and force a ping to negotiate new SAs ¯_(ツ)_/¯

“The case of the sad halfling”

bender’s SAD:

```
5.6.7.8 -> 1.2.3.4 (spi: 0x02)

1.2.3.4 -> 5.6.7.8 (spi: 0x01)
```

flexo’s SAD:

```
5.6.7.8 -> 1.2.3.4 (spi: 0x02)

1.2.3.4 -> 5.6.7.8 (spi: 0x01)
```

These are two happy hosts right now...

… But let’s say one SA “disappears” during a brief netsplit:

```
bender$ echo “deleteall 5.6.7.8 1.2.3.4 esp ; ” | setkey -c

# The 5.6.7.8 1.2.3.4 association gets removed from bender
```

“The case of the sad halfling”

bender’s SAD:

~~5.6.7.8 -> 1.2.3.4 (spi: 0x02)~~

1.2.3.4 -> 5.6.7.8 (spi: 0x01)

flexo’s SAD:

5.6.7.8 -> 1.2.3.4 (spi: 0x02)

1.2.3.4 -> 5.6.7.8 (spi: 0x01)

Any communication attempt will fail!

bender -> flexo (using spi `0x01`) traffic is received by flexo

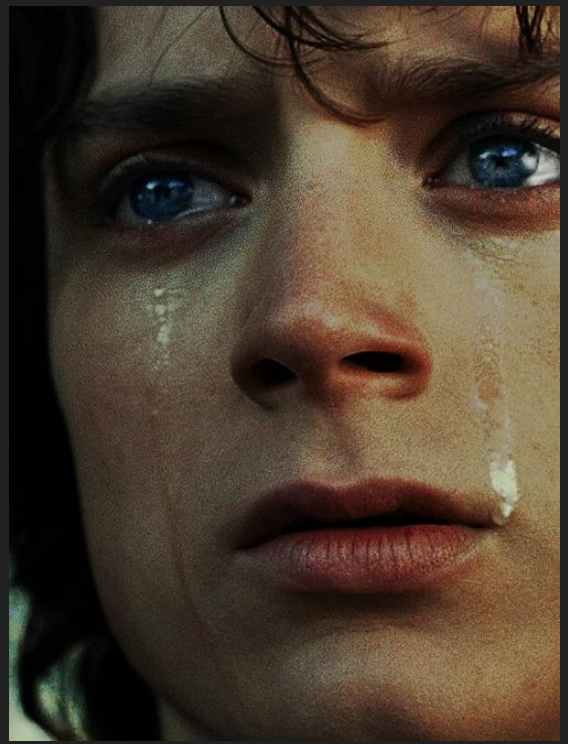
flexo -> bender (using spi `0x02`) traffic is ignored by bender!

“The case of the sad halfling”

Our solution: Implement our own phase 2 liveness check

- Check a known port for every host we have a mature SA with:
- Clear the SAs if ${max_tries} timeouts

Bonus points: Also check a port that won’t use IPsec to compare



Do instrument all the things!

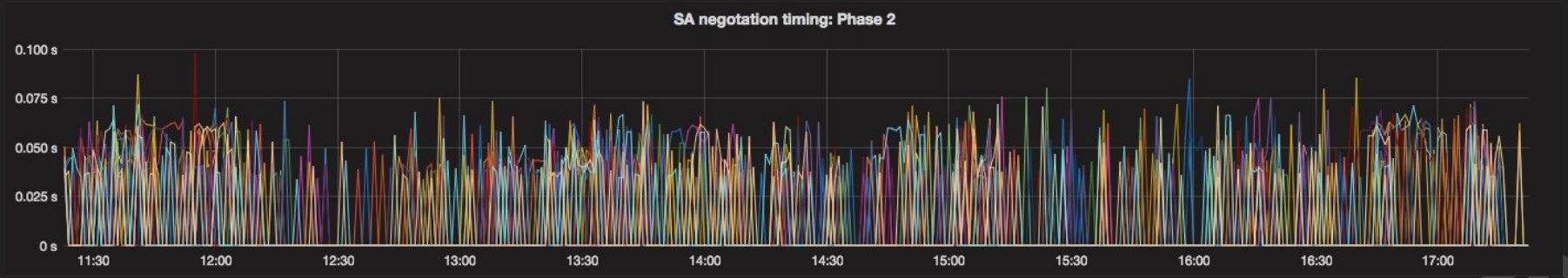
You can never have enough data about your systems!

You want your kernel compiled with `CONFIG_XFRM_STATISTICS`

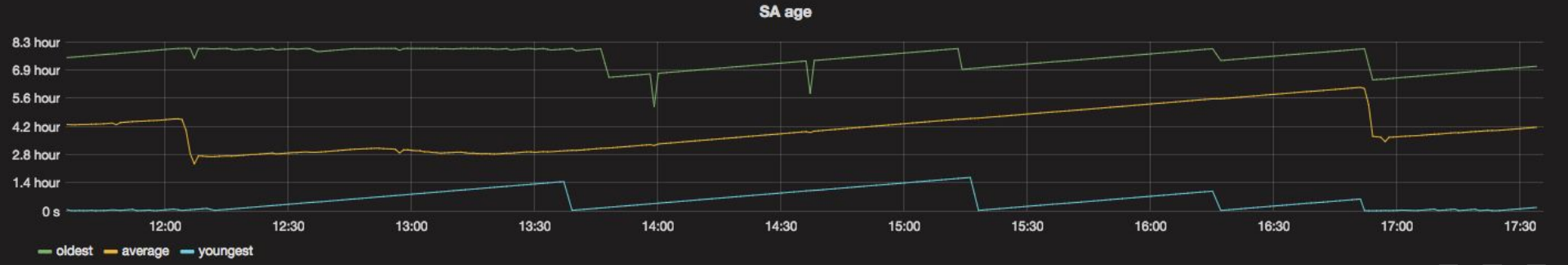
```
$ cat /proc/net/xfrm_stat
XfrmInError                     0
XfrmInBufferError               0
...
```

Build racoon to emit timing info on logs (build with `--enable-stats`)

Instrumenting the racoon logs



Instrumenting the SADB



Instrumenting kernel xfrm stats: XfrmInNoStates

Don’t trust the defaults!

Beware of IPsec overhead and MTU/MSS:

A 1450 bytes IP packet becomes:

- 1516 bytes in Transport mode
- 1532 bytes in Tunnel mode

(Docker containers will also have extra overhead)

Path MTU Discovery should help, but test it first!

Do use certs for auth, or don't use a weak PSK

PSK is great for getting started if you don't have PKI in place (we didn't)

But please:

- Use a strong PSK (if you must use PSK)
- Enable PFS (Perfect Forward Secrecy)
- Do not use aggressive mode for phase 1

Not following all that makes the NSA happy!

You don’t have the same tools available

tcpdump will just show ESP traffic, not its content:

```
15:47:51.511135 IP 1.2.3.4 > 5.6.7.8: ESP(spi=0x00fb0c52,seq=0x1afa), length 84

15:47:51.511295 IP 5.6.7.8 > 1.2.3.4: ESP(spi=0x095e5523,seq=0x173a), length 84
```

Traffic can be decrypted with wireshark/tshark if you dump the keys first

You don’t have the same tools available

Can use tcpdump with netfilter logging framework:

```
$ iptables -t mangle -I PREROUTING -m policy --pol ipsec --dir in -j NFLOG --nflog-group 5

$ iptables -t mangle -I POSTROUTING -m policy --pol ipsec --dir out -j NFLOG --nflog-group 5

$ tcpdump -i nflog:5
```

Doesn’t allow most filters

Might need to increase the buffer size

You don’t have the same tools available

Traceroute will attempt to use udp by default:

```
$ traceroute that.other.host
traceroute to that.other.host (5.6.7.8), 30 hops max, 60 byte packets
 1  * * *
 2  * * *
 3  * * *
 4  that.other.host (5.6.7.8)  0.351 ms  0.295 ms  0.297 ms
```

You can force it to use ICMP with `traceroute -I`

Don’t flush/restart on changes

Never restart racoon!

A racoon restart will flush all phase 1 and 2 SAs:

- Negotiating ~1000 SAs at once is no fun

To flush an individual SA: `ip xfrm state delete`

To reload config changes: `killall -HUP racoon`

Don’t flush/restart on changes

When adding/removing hosts, do not flush both SPD and SAD!

Instead, consider just flushing your SPD and letting unwanted SAs to expire

SAs not reflected in the SPD will never get used

Can flush that SA individually if feeling paranoid

Can just include `spdflush;` in your `ipsec-tools.conf` and reload with:

```
setkey -f /etc/ipsec-tools.conf
```

Thanks!

Questions?

Hated it? Want to say hi?

fran@hostedgraphite.com

@hostedgraphite