Libreswan

Teaching old code new tricks!

Libreswan is an IKE (Internet Key Exchange) daemon. Its origins can be traced back to the 90's.
But what is IKE and why should it run on NetBSD?

BSDCan 2020
Andrew Cagney
freenode.net #swan cagney
https://libreswan.org/

Part 1: What is and why use IKE?

A simple example using Libreswan

The problem #1

1.1.2. Endpoint-to-Endpoint Transport Mode

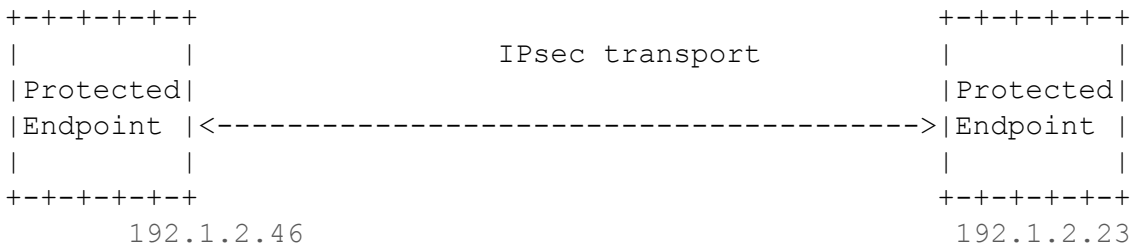


Figure 2: Endpoint to Endpoint

RFC 7296 (mostly)

The problem #2

1.1.1. Security Gateway to Security Gateway in Tunnel Mode

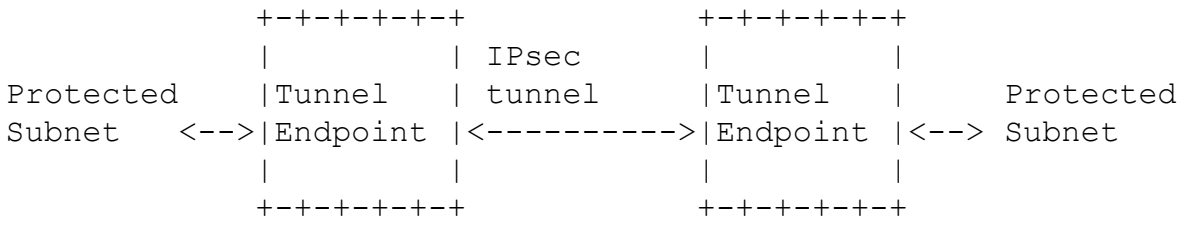
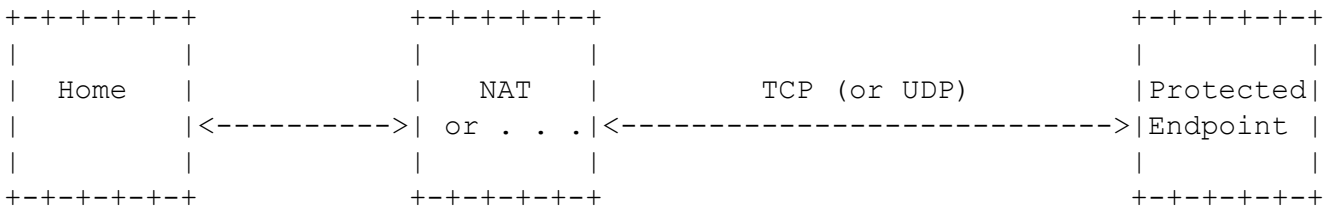


Figure 1: Security Gateway to Security Gateway Tunnel

RFC 7296

The problem #3



Using setkey ...

```
setkey -c <<EOF
add 192.1.2.46 192.1.2.23 esp 9876 -E 3des-cbc "hogehogehogehogehogehoge";
add 192.1.2.23 192.1.2.46 esp 10000 -E 3des-cbc
0xdeadbeefdeadbeefdeadbeefdeadbeefdeadbeefdeadbeef;
spdadd 192.1.2.46 192.1.2.23 any -P out ipsec esp/transport//use;
EOF
```

All very manual and error prone

NIST requirements and recommendations for the configuration of IPsec VPNs are:
[...] child SAs should be re-keyed after at most 8 hours.

http://www.netbsd.org/docs/network/ipsec/#trans_tunnel
https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-77.pdf

IKEv2:
Internet Key Exchange version 2

Handles setting up and maintaining IPsec connections:

- Cryptosuite selection - AES_GCM? ….
- Establishing one time secrets between ends
- …

Doesn’t suffer from the travesties of IKEv1

Building Libreswan on NetBSD ...

Defaults set by mk/defaults/netbsd.mk:

```
/usr/local/sbin/ipsec
/usr/local/libexec/ipsec/
/usr/local/etc/ipsec.{conf,secrets,d/}
/var/log/pluto.log
/var/run/pluto/pluto.{pid,ctl}
/etc/rc.d/pluto
```

Defaults in mk/config.mk

Put local overrides in Makefile.inc.local or pass to gmake

By default don’t scribble on /usr/pkg/

… Building Libreswan on NetBSD

(assuming pkgsrc was enabled during install)

pkgin install mozilla-rootcerts && mozilla-rootcerts install
Ref: https://www.cambus.net/installing-ca-certificates-on-netbsd/

pkgin install git

$ git clone https://github.com/libreswan/libreswan

pkgin install gmake nss unbound bison ldns xmlto

$ gmake

gmake install

Starting Libreswan

```
# ipsec start
Redirecting to: /etc/rc.d/pluto onestart
Starting pluto.

# ipsec status
...
```

Configuring Libreswan

```
# cat /usr/local/etc/ipsec.conf
config setup
     #logfile=/var/log/pluto.log
     #logappend=no
     #plutodebug=base,crypt
     dumpdir=/tmp
conn transport
     leftid=@west
     rightid=@east
     authby=secret
     left=192.1.2.46
     right=192.1.2.23
     type=transport
     esp=aes_128-sha1
```

```
Do not enable “crypt” debugging at home,
It exposes keying material (but is good for kernel
and pfkey debugging)
```

```
# cat /usr/local/etc/ipsec.secrets
@west @east : PSK "ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890"
: real configurations use certificates
```

```
: libreswan determines left - right from interface
$ ifconfig wm2
        inet 192.1.2.46/24 broadcast 192.1.2.255 flags 0x0
```

Establishing the connection

```
# ipsec auto --add transport
002 added connection description "transport"

# sudo ipsec auto --up transport
181 "transport" #1: initiating IKEv2 IKE SA
181 "transport" #1: STATE_PARENT_I1: sent v2I1, expected v2R1
182 "transport" #1: STATE_PARENT_I2: sent v2I2, expected v2R2 {auth=IKEv2 cipher=AES_GCM_16_256 integ=n/a
          prf=HMAC_SHA2_512 group=MODP2048}
002 "transport" #2: IKEv2 mode peer ID is ID_FQDN: '@east'
003 "transport" #1: authenticated using authby=secret
002 "transport" #2: negotiated connection [192.1.2.46-192.1.2.46:0-65535 0] ->
          [192.1.2.23-192.1.2.23:0-65535 0]
004 "transport" #2: STATE_V2_ESTABLISHED_CHILD_SA: IPsec SA established transport mode
          {ESP=>0x628e8d1b <0xa8e67137 xfrm=AES_CBC_128-HMAC_SHA1_96 NATOA=none NATD=none DPD=passive}
```

... and confirm

```
# ping 192.1.2.23 > /dev/null &

# tcpdump -i wm2
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on wm2, link-type EN10MB (Ethernet), capture size 262144 bytes
18:45:36.670632 IP 192.1.2.46 > 192.1.2.23: ESP(spi=0x628e8d1b,seq=0xc), length 116
18:45:37.322999 IP 192.1.2.23 > 192.1.2.46: ESP(spi=0xa8e67137,seq=0xc), length 116
```

Part 2: Why libreswan and IKEv2

Why IKEv2?

- IKEv2 is a far simpler protocol
- UDP tunneling (NAT)
- Authenticated Encryption with Associated Data (AEAD in IKE)
- Fragmentation (RFC 7383)
- Post quantum Keys (draft-ietf-ipsecme-qr-ikev2)
- Latest Algorithms for IKE (chacha poly …)

Needs work on NetBSD:

- Opportunist Encryption (packet triggers encryption)
- TCP tunneling (RFC 8229) - kernel support
- Mobile IKE (RFC 4555)

https://libreswan.org/wiki/RFC_List

Why Libreswan on NetBSD

	Racoon	Racoon 2	Strongswan	OpenIKED	Libreswan
IKEv1	Yes	Yes	FreeBSD	No	Yes
IKEv2	No	Yes	FreeBSD	OpenBSD	Yes
Crypto Library	openssl	openssl	openssl	libressl	NSS
FIPS Tested	No	No	Yes	No	Yes
FIPS Boundary	No	No	No	No	Yes (NSS)
Test framework			KVM		KVM / namespaces
Tracking RFCs	No		Yes	Yes	Yes

Errata: The version in the Video has OpenIKED using openssl

Part 3: More on IKEv2

Basic IKEv2 involves two exchanges

IKE_SA_INIT exchange:

- Establishes the IKE SA (security association)
- A secure channel between two machines
- No trust

IKE_AUTH exchange:

- Establishes trust
- Allows both ends to prove their identity
- Also establishes the CHILD SA (i.e., IPsec tunnel)

IKE_SA_INIT: Initiator -> ...

Initiator generates:

- Key Material (KE) for a Diffie-Hellman group

and then sends the IKE_SA_INIT request containing:

- IKE initiator SPI (security parameter index)
- Nonce (something random)
- Sequence ID 0
- Proposed cryptographic algorithms for the IKE SA:
  Encryption (ENCR) …, integrity (INTEG), …, pseudorandom function (PRF), … Diffie-Hellman group (DH) ...
- Key Material (KE) for one of the Diffie-Hellman groups

IKE_SA_INIT … -> Responder -> ...

Responder:

- Selects a cryptosuite (ENCR+INTEG+PRF+D-H)
- Computes corresponding keying material

and then responds with:

- IKE initiator and responder SPIs
- Nonce (something random)
- Sequence ID 0
- Selected crypto suite (ENCR+INTEG+PRF+D-H)
- Key Material (KE) for selected D-H group

IKE_SA_INIT: … -> Initiator

Initiator:

- Checks the accepted cryptosuite is OK (ENCR+INTEG+PRF+D-H)
- Computes corresponding keying material

Both ends can then complete the D-H computation:

- Have the shared secret (g^ir)
- Combines the g^ir with the nonces using the PRF (SKEYSEED)
- Expand the SKEYSEED to obtain keying material:
  - Initiator and responder keys
  - Material for proving identity
  - CHILD SA keys

IKE_AUTH: Initiator -> ...

The initiator sends an IKE_AUTH request containing:

- IKE initiator and responder SPIs
- Sequence ID 1
- Encrypt the following using ENCR:
  - Signed material to prove identity
  - Proposed Algorithms for the CHILD SA
  - Details of what CHILD SA to establish (ESP, AH) (IP address, …)

All secured using INTEG

IKE_AUTH: … -> Responder -> ...

The responder:

- Checks the message’s integrity
- Decrypts the encrypted payload
- Verifies the initiator’s identity
- Selects and creates the CHILD SA

Then responds with its IKE_AUTH:

- Includes proof of Responder’s identity
- Details of created CHILD SA

IKE_AUTH: … -> Initiator

Finally the initiator:

- Checks the message’s integrity
- Decrypts the content
- Verifies the responders identity
- Creates its end of the CHILD SA

If the initiator didn’t like the responder’s proof-of-identity it instead deletes the SA.

The proposals in more detail

The list of proposals contains a list of possible algorithms.
For instance, a single proposal containing all the original IKEv1 algorithms would be:

```
  (ENCR) DES+3DES - (PRF) SHA1+MD5+Tiger - (INTEG) SHA1_96+MD5+Tiger - (D-H) MODP1+MODP2+MODP3+MODP4
```

Giving 72 combinations. Libreswan on NetBSD currently sends 4 proposals:

```
$ ipsec algparse ike
algparse -v2 'ike'
      AES_GCM_16_256-HMAC_SHA2_512+HMAC_SHA2_256-MODP2048+MODP3072+MODP4096+MODP8192+DH19+DH20+DH21+DH31
      AES_GCM_16_128-HMAC_SHA2_512+HMAC_SHA2_256-MODP2048+MODP3072+MODP4096+MODP8192+DH19+DH20+DH21+DH31
      AES_CBC_256-HMAC_SHA2_512+HMAC_SHA2_256-MODP2048+MODP3072+MODP4096+MODP8192+DH19+DH20+DH21+DH31
      AES_CBC_128-HMAC_SHA2_512+HMAC_SHA2_256-MODP2048+MODP3072+MODP4096+MODP8192+DH19+DH20+DH21+DH31
```

Giving 96 combinations.

The responder needs to match this against its local list, select one combination, and send it back.

The initiator then checks that the accepted proposal was sent.

Let’s say the initiator sends like ….

```
AES_CBC_{128,192,256}+CAMELLIA_CBC_{128,192,256}+AES_CTR_{128,1
92,256}+CAMELLIA_CTR_{128,192,256}+3DES-HMAC_SHA2_256_128+HMAC_
SHA2_384_192+HMAC_SHA2_512_256HMAC_SHA1_96+AES_XCBC_96+AES_CMAC
_96-HMAC_SHA2_256+HMAC_SHA2_384+HMAC_SHA2_512+AES128_XCBC+AES12
8_CMAC+HMAC_SHA1-ECP_256+ECP_384+ECP_521+BRAINPOOL_P256R1+BRAIN
POOL_P384R1+BRAINPOOL_P512R1+CURVE25519+CURVE448+OAKLEY_GROUP__
1040+MODP3072+MODP4096+MODP6144+MODP8192+MODP2048
...
```

With debug logging enabled, Libreswan was a tad slow.... Oops!
Fixed in 2016.

Interop testing passes, but then ….

... this arrives:

```
As soon as the [...] client starts negotiating
IKEv2 with Libreswan, [...] blows up and
reloads.
```

The detail was in the response ...

The accepted proposal response contains:

- The index of the proposal that was accepted: 1, 2, or …
- The cryptosuite (ENCR+INTEG+PRF+H-H) chosen from that proposal

The index was out-of-bounds.

Libreswan (and presumably the other IKE daemons we tried) didn’t trust the index and, presumably, ignored it.

Part 4: From old to New

Libreswan’s Long History

1995 FreeS/WAN started by John Gilmore (Sun, Cygnus, EFF) to encrypt the internet

2003/4FreeS/WAN winds down, Openswan (and Strongswan) fork

2005? Openswan ported to the BSDs

2012 Paul Wouters et.al., fork Openswan creating Libreswan

2018 Libreswan kick’s the old BSD code bases tyres, only one wheel falls off

2019 Libreswan announces KLIPS is being removed

2020 Libreswan drops KLIPS support (leaving XFRM as only Linux kernel backend)

2020 Libreswan revives NetBSD - ensures that there are multiple kernel backend

2020 Ravi Teja (GSOC) starts work on adding BSDs to KVM test infrastructure

https://en.wikipedia.org/wiki/FreeS/WAN
https://freeswan.org/history.html
https://libreswan.org/wiki/History

Why does history matter?
Because the workload changed!

When FreeS/WAN started:

- VPNs were between home computers
- Once up, they stayed up

But then in 2007 the iPhone happened:

- VPNs are between mobile phones and servers
- Never stay up
- Constant connections

Shared history can mean shared bugs

CVE-2019-10155:
IKEv1 Informational exchange integrity check failure

The impact of this vulnerability is low, as it cannot be exploited.
(for libreswan; for strongswan and openswan see below)

Vulnerable versions:

- libreswan < 3.29
- strongswan < 5.0
- openswan - all versions (as of writing: 2.6.51.3)

Not vulnerable: libreswan 3.29 and later, strongswan 5.0 and later, **freeswan**

Performance can’t beat physics

(all numbers are relative, use plutodebug=cpu-usage)

- 3ms - initiator computes DH secret
- 3ms - responder computes DH secret
- 8ms - initiator computes DH shared secret
- 8ms - responder computes DH shared secret

Ignoring RSA (30ms+?) a responder requires >11ms per IKE SA:

Libreswan’s Performance

Focus switched from staying up to how many IKE connections can be established per second:

Libreswan:

- Hash tables
- Private key cache
- Helper threads

FreeS/WAN:

- Linked lists

Reduced 100’s of ms per exchange to 30-60ms (with RSA); work ongoing.
Hitting scaling problems with Linux’s XFRM

Testing: extract unit tests

Then:

The code base already contained lots of embedded (but long forgotten) test code:

```
#ifdef TTOADDR_MAIN
Int main(int argc, char *argv[])
{
     exit(regress());
}

{"1.2.3.0",    0,   0,   0,   "1.2.3.0"},
{"1:2::3:4",   0,   0,   0,   "1:2::3:4"},
```

Now:

- Test code extracted
- Extended
- Added to test framework

For instance, algparse uses pluto's code for parsing proposals.

Testing: interop. testing

Then:

Code base used UML (user mode linux) to test interops:

- Mostly IKE interops.
- No FIPS
- Only one kernel
- No routers ...

Now:

UML replaced by KVM:

- Tests migrated
- Minimal host requirements (building, key generation, on KVM)
- In theory, portable (good at finding KVM bugs ...)
- https://testing.libreswan.org/ 4 cores: 3 threads booting VMs; 6 test parallel tests
- Ravi Teja (GSOC) adding BSD guests

Namespaces

- Uses identical test files
- Much faster
- Limited interop. testing

Testing: FIPS

FIPS is interesting as it forces some discipline onto the code base:

- Need to structure code so it can test the cryptographic interfaces
- Free test vectors for PRFs, DH, ...

Then: Codebase was littered with switches:

```
case ESP_3DES:
     needed_len = DES_CBC_BLOCK_SIZE * 3;
     break;
case ESP_AES:
     needed_len = AES_CBC_BLOCK_SIZE;
     break;
```

Now: table driven:

- Easier to add algorithms

```
const struct encrypt_desc
    ike_alg_encrypt_aes_cbc = {
     .enc_blocksize = AES_CBC_BLOCK_SIZE,
     .pad_to_blocksize = TRUE,
     .wire_iv_size =     AES_CBC_BLOCK_SIZE,
     .keydeflen =   AES_KEY_DEF_LEN,
     .key_bit_lengths = { 256, 192, 128, },
     .encrypt_ops = &ike_alg_encrypt_nss_cbc_ops,
};
```

NetBSD backend - PFKEY

- Based on RFC 2367
  PF_KEY Key Management API, Version 2
- Wrapped in the KAME derived library libipsec

Libreswan is using NetBSD's libipsec:

- Less embedded than FreeBSD's
- Less maintained than FreeBSD's
  - found out-of-bounds bug fixed in FreeBSD
  - Required -Werror fixes
- #defined printf() to use Libreswan's debug library (plutodebug=crypt)

Debugging NetBSD’s PFKEY

From command line:

```
# setkey -v …
# setkey -D ; setkey -P -D
```

From pluto:

- plutodebug=all,crypt
- Compare output

The End