ike Documentation

Release 0.0.3

Kimmo Parviainen-Jalanko

Mar 12, 2018

Contents

Contents:

CHAPTER 1

About ike

The goal of this project is to be a minimalistic IKEv2 (RFC 5996) implementation in Python.

1.1 Status

This project is in early stages. Use at own risk.

It will make your IP stack talk ESP to the remote peer.

What it can do:

- Act as an initiator
- Authenticate itself and peer using raw RSA keys.
- Install ESP SAs and SPD entries to use the key material via `setkey` command from ipsec-tools.

Limitations (hardcoded values):

- Cipher algorithm is Camellia in CBC mode with 256 bit keys.
- HMAC / Hash / PRF algorithm is SHA2/256.
- IKE group is Diffie-Hellman modp 14.
- Authentication (both own private and peer public) key file paths are hardcoded.
- 'setkey' syntax is of whatever the ipsec-tools on Debian 7.1 accept.
- Traffic selectors are myip:any:0-65535 <-> peerip:any:0-65535

1.2 Design principles

- Minimal amount of code.
- Support *MUST* features of draft-kivinen-ipsecme-ikev2-rfc5996bis-02 (RFC 5996 successor)

- Use strongest algorithms possible.

1.3 Documentation

You can read the Documentation at https://ike.readthedocs.org

1.3.1 What this project is *NOT* going to be

- ISAKMP (IKEv1) RFC 2409 compliant
- IPSec data plane / ESP protocol

1.4 License

- MIT License

1.5 References

- http://tools.ietf.org/html/draft-kivinen-ipsecme-ikev2-rfc5996bis-02
- http://tools.ietf.org/html/draft-kivinen-ipsecme-ikev2-minimal-01

CHAPTER 2

ike package

2.1 Subpackages

2.1.1 ike.util package

Submodules

ike.util.cipher module

class ike.util.cipher.**AES**(*key*, *iv=None*)
Bases: ike.util.cipher._Cipher

algorithm
alias of *AES*

class ike.util.cipher.**Camellia**(*key*, *iv=None*)
Bases: ike.util.cipher._Cipher

algorithm
alias of *Camellia*

ike.util.cipher.**pad**(*data*, *blocksize=16*)
Pads data to blocksize according to RFC 4303. Pad length field is included in output.

ike.util.conv module

ike.util.conv.**to_bytes**(*x*)

ike.util.dh module

class ike.util.dh.**DiffieHellman**(*group=14*, *n=64*)
Bases: object

derivate(*other_key*)

generate_private_key(*n*)

generate_public_key()

generator = 2

shared_secret

ike.util.dump module

ike.util.dump.**dump**(*src*)
Returns data in hex format in groups of 4 octets delimited by spaces for debugging purposes.

ike.util.external module

ike.util.external.**run_setkey**(*input*)
Runs a script through the 'setkey' command that is a user space insterface for PFKEY. :param input: setkey configuration file contents.

ike.util.prf module

ike.util.prf.**prf**(*key*, *data*, *hash_algorithm='sha256'*)

ike.util.prf.**prfplus**(*key*, *data*, *n*)

ike.util.pubkey module

ike.util.pubkey.**sign**(*data*, *filename*, *hash_alg='SHA-256'*)

ike.util.pubkey.**verify**(*data*, *signature*, *filename*)

Module contents

2.2 Submodules

2.3 ike.const module

class ike.const.**AuthenticationType**
Bases: enum.IntEnum

An enumeration.

DSS = 3

PSK = 2

RSA = 1

class ike.const.**ExchangeType**
Bases: enum.IntEnum

An enumeration.

CREATE_CHILD_SA = 36

IKE_AUTH = 35

IKE_SA_INIT = 34

INFORMATIONAL = 37

class ike.const.**MessageType**

Bases: enum.IntEnum

An enumeration.

ADDITIONAL_IP4_ADDRESS = 16397

ADDITIONAL_IP6_ADDRESS = 16398

ADDITIONAL_TS_POSSIBLE = 16386

ANOTHER_AUTH_FOLLOWS = 16405

AUTHENTICATION_FAILED = 24

AUTHORIZATION_FAILED = 46

AUTH_LIFETIME = 16403

CHILDLESS_IKEV2_SUPPORTED = 16418

CHILD_SA_NOT_FOUND = 44

COOKIE = 16390

COOKIE2 = 16401

EAP_ONLY_AUTHENTICATION = 16417

ERX_SUPPORTED = 16427

ESP_TFC_PADDING_NOT_SUPPORTED = 16394

FAILED_CP_REQUIRED = 37

HTTP_CERT_LOOKUP_SUPPORTED = 16392

IFOM_CAPABILITY = 16428

IKEV2_MESSAGE_ID_SYNC = 16422

IKEV2_MESSAGE_ID_SYNC_SUPPORTED = 16420

INITIAL_CONTACT = 16384

INTERNAL_ADDRESS_FAILURE = 36

INVALID_GROUP_ID = 45

INVALID_IKE_SPI = 4

INVALID_KE_PAYLOAD = 17

INVALID_MAJOR_VERSION = 5

INVALID_MESSAGE_ID = 9

INVALID_SELECTORS = 39

INVALID_SPI = 11

INVALID_SYNTAX = 7

```
IPCOMP_SUPPORTED = 16387
IPSEC_REPLAY_COUNTER_SYNC = 16423
IPSEC_REPLAY_COUNTER_SYNC_SUPPORTED = 16421
LINK_ID = 16414
MOBIKE_SUPPORTED = 16396
MULTIPLE_AUTH_SUPPORTED = 16404
NAT_DETECTION_DESTINATION_IP = 16389
NAT_DETECTION_SOURCE_IP = 16388
NON_FIRST_FRAGMENTS_ALSO = 16395
NO_ADDITIONAL_ADDRESSES = 16399
NO_ADDITIONAL_SAS = 35
NO_NATS_ALLOWED = 16402
NO_PROPOSAL_CHOSEN = 14
PSK_CONFIRM = 16426
PSK_PERSIST = 16425
QUICK_CRASH_DETECTION = 16419
REDIRECT = 16407
REDIRECTED_FROM = 16408
REDIRECT_SUPPORTED = 16406
REKEY_SA = 16393
ROHC_SUPPORTED = 16416
Reserved = 0
SECURE_PASSWORD_METHODS = 16424
SENDER_REQUEST_ID = 16429
SET_WINDOW_SIZE = 16385
SINGLE_PAIR_REQUIRED = 34
TEMPORARY_FAILURE = 43
TICKET_ACK = 16411
TICKET_LT_OPAQUE = 16409
TICKET_NACK = 16412
TICKET_OPAQUE = 16413
TICKET_REQUEST = 16410
TS_UNACCEPTABLE = 38
UNACCEPTABLE_ADDRESSES = 40
UNEXPECTED_NAT_DETECTED = 41
UNSUPPORTED_CRITICAL_PAYLOAD = 1
```

UPDATE_SA_ADDRESSES = 16400

USE_ASSIGNED_HoA = 42

USE_TRANSPORT_MODE = 16391

USE_WESP_MODE = 16415

class ike.const.**ProtocolID**
Bases: enum.IntEnum

An enumeration.

AH = 2

ESP = 3

IKE = 1

2.4 ike.initiator module

IKE v2 (RFC 5996) initiator implementation

Usage: initiator.py <remote_peer>

To clean up afterwards,

```
setkey -FP && setkey -F
```

class ike.initiator.**IKEInitiator**
Bases: asyncio.protocols.DatagramProtocol

Implements an IKE initiator that attempt to negotiate a single child SA to remote peer.

connectionRefused()

connection_made(*transport*)

datagram_received(*data*, *address*)

ike.initiator.**main**(*peer*)

2.5 ike.payloads module

IKEv2 Payloads as specified in RFC 5996 sections 3.2 - 3.16

class ike.payloads.**AUTH**(*signed_octets=None*, *data=None*, *next_payload=<no_next_payload: 0>*, *critical=False*)
Bases: ike.payloads._IkePayload

Authentication Payload

class ike.payloads.**IDi**(*data=None*, *next_payload=<no_next_payload: 0>*, *critical=False*)
Bases: ike.payloads._IkePayload

Identification Payload for initiator

class ike.payloads.**IDr**(*data=None*, *next_payload=<no_next_payload: 0>*, *critical=False*)
Bases: ike.payloads._IkePayload

Identification Payload for responder

class ike.payloads.**KE**(*data=None*, *next_payload=<no_next_payload: 0>*, *critical=False*, *group=14*, *diffie_hellman=None*)

Bases: ike.payloads._IkePayload

Key Exchange Payload

parse(*data*)

class ike.payloads.**Nonce**(*data=None*, *next_payload=<no_next_payload: 0>*, *critical=False*, *nonce=None*)

Bases: ike.payloads._IkePayload

Nonce Payload

parse(*data*)

class ike.payloads.**Notify**(*notify_type=None*, *data=None*, *next_payload=<no_next_payload: 0>*, *critical=False*)

Bases: ike.payloads._IkePayload

Notify Payload

parse(*data*)

class ike.payloads.**SA**(*data=None*, *proposals=None*, *next_payload=<no_next_payload: 0>*, *critical=False*)

Bases: ike.payloads._IkePayload

Security Association Payload

parse(*data*)

class ike.payloads.**SK**(*data=None*, *next_payload=<no_next_payload: 0>*, *critical=False*, *iv=None*, *ciphertext=None*)

Bases: ike.payloads._IkePayload

Encrypted Payload

mac(*hmac*)

class ike.payloads.**TSi**(*addr=None*, *data=None*, *next_payload=<no_next_payload: 0>*, *critical=False*)

Bases: ike.payloads._TS

Traffic Selector Payload for initiator

class ike.payloads.**TSr**(*addr=None*, *data=None*, *next_payload=<no_next_payload: 0>*, *critical=False*)

Bases: ike.payloads._TS

Traffic Selector Payload for responder

class ike.payloads.**Type**

Bases: enum.IntEnum

Payload types from IANA

AUTH = 39

CERT = 37

CERTREQ = 38

CP = 47

Delete = 42

EAP = 48

GSA = 51

GSPM = 49

IDg = 50

IDi = 35

IDr = 36

KD = 52

KE = 34

Ni = 40

Nonce = 40

Notify = 41

Nr = 40

SA = 33

SK = 46

TSi = 44

TSr = 45

no_next_payload = 0

ike.payloads.**get_by_type**(*payload_type*)
Returns an IkePayload (sub)class based on the RFC5996 payload_type :param payload_type: int() Ike Payload type

2.6 ike.proposal module

Implements Proposal and Transform substructures for Security association (SA) payloads.

Conforms to RFC5996 section 3.3

class ike.proposal.**Proposal**(*data=None*, *num=1*, *protocol=<ProtocolID.IKE: 1>*, *spi=None*, *spi_len=0*, *last=False*, *transforms=None*)
Bases: object

data

parse(*data*)

class ike.proposal.**Transform**(*name*, *keysize=None*, *last=False*)
Bases: object

data

2.7 ike.protocol module

High level interface to IKEv2 protocol

class ike.protocol.**IKE**(*address*, *peer*, *dh_group=14*, *nonce_len=32*)
Bases: object

A single IKE negotiation / SA.

Currently implements only Initiator side of the negotiation.

auth_recv()
Handle peer's IKE_AUTH response.

auth_send()
Generates the second (IKE_AUTH) packet for Initiator

Returns bytes() containing a valid IKE_INIT packet

authenticate_peer(*auth_data*, *peer_id*, *message*)
Verifies the peers authentication.

decrypt(*data*)
Decrypts an encrypted (SK, 46) IKE payload using self.SK_er

Parameters **data** – Encrypted IKE payload including headers (payloads.SK())

Returns next_payload, data_containing_payloads

Raises ***IkeError*** – If packet is corrupted.

encrypt_and_hmac(*packet*)
Encrypts and signs a Packet() using self.SK_ei and self.SK_ai

Parameters **packet** – Unecrypted Packet() with one or more payloads.

Returns Encrypted and signed Packet() with a single payloads.SK

init_recv()
Parses the IKE_INIT response packet received from Responder.

Assigns the correct values of rSPI and Nr Calculates Diffie-Hellman exchange and assigns all keys to self.

init_send()
Generates the first (IKE_INIT) packet for Initiator

Returns bytes() containing a valid IKE_INIT packet

install_ipsec_sas()

parse_packet(*data*)
Parses a received packet in to Packet() with corresponding payloads. Will decrypt encrypted packets when needed.

Parameters **data** – bytes() IKE packet from wire.

Returns Packet() instance

Raises ***IkeError*** – on malformed packet

verify_hmac(*data*)
Verifies the HMAC signature of an encrypted (SK, 46) payload using self.SK_ar

Parameters **data** – bytes(payloads.SK())

Raises ***IkeError*** – if calculated signature does not match the one in the payload

exception ike.protocol.**IkeError**
Bases: Exception

class ike.protocol.**Packet**(*data=None*, *exchange_type=None*, *message_id=0*, *iSPI=0*, *rSPI=0*)
Bases: object

An IKE packet.

To generate packets:

1. instantiate an Packet()
2. add payloads by Packet.add_payload(<payloads.IkePayload instance>)
3. send bytes(Packet) to other peer.

Received packets should be generated by IKE.parse_packet().

add_payload(*payload*)
Adds a payload to packet, updating last payload's next_payload field

class ike.protocol.**State**
Bases: enum.IntEnum

An enumeration.

AUTH = 2

INIT = 1

STARTING = 0

2.8 Module contents

CHAPTER 3

ike

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

i

Index

P

Q

R

S

T

U

V