Kerberos Authentication Service

- Kerberos is trusted authority with whom everyone shares keys.
- When a client on a network wants to talk to a server, he issues a request for a ticket to Kerberos’ Ticket Granting Server (TGS).
- Client uses this ticket always when he talks to the server, sometimes he also sends authenticators.
- Clients and servers do not trust each other.

A ticket is used to pass securely to the server the identity of the client.
- It is good for a single client and single server for some period of time.
- It contains client’s name and network address, server’s name, timestamp and a session key, all encrypted with a key server shares with Kerberos.
- An authenticator is generated whenever a client requests some service from the server.
- It is good for one request.
- It contains client’s name, a timestamp and an optional additional key, all encrypted with session key.

Getting Initial Ticket

- To get initial ticket:
  - Alice sends a message with her name and a name of a Ticket Granting Server (TGS) to Kerberos.
  - Kerberos generates a session key $K_{A,TGS}$ to be used between her and TGS and also generates Ticket Granting Ticket (TGT).
  - Kerberos encrypts $K_{A,TGS}$ with Alice’s secret key and sends that and TGT to Alice.
  - Alice retrieves $K_{A,TGS}$ and saves it and TGT.

- Alice sends a request with her name and server’s name to TGS, encrypted with $K_{A,TGS}$, accompanied with TGT and authenticator.
- TGS decrypts TGT with his secret key and retrieves $K_{A,TGS}$
- TGS uses $K_{A,TGS}$ to decrypt authenticator and compare Alice’s information in authenticator with information in TGT, and compare timestamps.
- If everything matches he generates a session key $K_{A,S}$ to be used between her and server and a valid ticket $T_{A,S}$
- TGS encrypts $K_{A,S}$ with $K_{A,TGS}$ and sends this and $T_{A,S}$ to Alice.
Getting Ticket for Server S

### Kerberos Authentication Service

- To request service:
  - Alice sends a valid ticket $T_{A,S}$ and authenticator
  - Server decrypts $T_{A,S}$ with his secret key and retrieves $K_{A,S}$
  - Server uses $K_{A,S}$ to decrypt authenticator and compare Alice’s information in authenticator with information in $T_{A,S}$ and compare timestamps
  - If everything matches he grants the request
  - For applications that require mutual authentication server will send to Alice a timestamp encrypted with $K_{A,S}$

### SSH Transport Protocol

1. Client contacts the server, performs TCP 3-way handshake
2. Both sides send the protocol and software version numbers
3. Key exchange
3-way handshakes

1. TCP SYN
2. TCP SYN ACK
3. TCP ACK

Negotiate protocol

SSH protocol and version

Negotiate Key Exchange

SSH_KEXINIT

Random value R
Key exchange algorithms
Server host key algorithm
Encryption algorithm client-to-server
Encryption algorithm server-to-client
MAC algorithm client-to-server
MAC algorithm server-to-client
Compression algorithm client-to-server
Compression algorithm server-to-client

Negotiate Key Exchange

SSH_KEXINIT

Random value R
Key exchange algorithms
Server host key algorithm
Encryption algorithm client-to-server
Encryption algorithm server-to-client
MAC algorithm client-to-server
MAC algorithm server-to-client
Compression algorithm client-to-server
Compression algorithm server-to-client

Negotiate Key Exchange

SSH_KEXINIT

Random value R
Key exchange algorithms
Server host key algorithm
Encryption algorithm client-to-server
Encryption algorithm server-to-client
MAC algorithm client-to-server
MAC algorithm server-to-client
Compression algorithm client-to-server
Compression algorithm server-to-client

Negotiate Key Exchange

SSH_KEXINIT

Random value R
Key exchange algorithms
Server host key algorithm
Encryption algorithm client-to-server
Encryption algorithm server-to-client
MAC algorithm client-to-server
MAC algorithm server-to-client
Compression algorithm client-to-server
Compression algorithm server-to-client

Negotiate Key Exchange

SSH_KEXINIT

Random value R
Key exchange algorithms
Server host key algorithm
Encryption algorithm client-to-server
Encryption algorithm server-to-client
MAC algorithm client-to-server
MAC algorithm server-to-client
Compression algorithm client-to-server
Compression algorithm server-to-client
Negotiate Key Exchange

SSH_KEXINIT

Client

Server

SSH_KEXINIT

Random value R

Key exchange algorithms

Encryption algorithm client-to-server

Encryption algorithm server-to-client

MAC algorithm client-to-server

MAC algorithm server-to-client

Compression algorithm client-to-server

Compression algorithm server-to-client

Key Exchange (Diffie-Hellman)

Client

e^g^x \mod p

Server

Compute

K = (e^g^x)^s \mod p

H = hash(V_c || l || l_s || K_s || e || f || K)

S = signature on H with private key

V_c = version string for client

K_s = random number client sent in KEXINIT message

K_p = server’s public (host) key

Host keys

- Each host has a host key – private/public key pair used for authentication

- Two different trust models exist:
  - First time client contacts a new host it stores the public key and associates it with the server name. Next time client checks if public key matches the stored value. If you have specified strict checking mismatches will be rejected, otherwise not.
  - Host’s name-to-key mapping is certified by a certification authority, client only knows CA’s public key

Example

ssh -v copland
debag 1: Connecting to copland [128.175.13.92] port 22.
debag 1: Connection established.
debag 1: Remote protocol version 1.99, remote software version OpenSSH_3.7.1p2
debag 1: match: OpenSSH_3.7.1p2 pat OpenSSH*
debag 1: Enabling compatibility mode for protocol 2.0

debag 1: Local version string SSH-2.2 OpenSSH_3.7.1p1

debag 1: SSH2_MSG_KEXINIT received

debag 1: key: server->client blowfish-cbc hmac-md5 none

debag 1: key: client->server blowfish-cbc hmac-md5 none

debag 1: SSH2_MSG_KEX_DH_GEX_REQUEST sent

debag 1: expecting SSH2_MSG_KEX_DH_GEX_GROUP

debag 1: expecting SSH2_MSG_KEX_DH_GEX_INIT sent

debag 1: expecting SSH2_MSG_KEX_DH_GEX_REPLY

Warning: Permanently added 'copland.128.175.13.92'(RSA) to the list of known hosts.
User Authentication Protocol

- Runs on top of transport layer
- Authenticates the client’s user to the server
- Uses value H from key exchange as session identifier

User Authentication

Client

List of authentications that can continue
Client chooses one of the offered methods

Server

publickey, password, hostbased

User Authentication

Client

Authentication request:
Username: alice
Service name
Authentication method name
Authentication data

Server

Server verifies that the username exists and authentication data is correct

User Authentication

Client

Authentication request:
Username: alice
Service name
Authentication method name: publickey
Public key algorithm name
Alice’s public key
Signature on session identifier and X with Alice’s private key

Server

User Authentication

Client

Authentication request:
Username: alice
Service name
Authentication method name: password
Password in plaintext

Server

Authentication success or Authentication failure
List of authentications that can continue
Host Based Authentication

- Authentication request: User name: alice
- Service name: Authentication method name: host-based
- Public key algorithm for host key
- Public host key and certificates
- Client host name: strauss
- Signature on session identifier and X with
- Server

Server sometimes does DNS lookup to verify that source address in packets and client host name match

IPSec

- Protocol for providing security at IP level
  - Access control
  - Authentication
  - Integrity
  - Replay control
  - Data confidentiality
- Consists of two protocols:
  - Authentication Header (AH) protocol
  - Encapsulating Security Payload (ESP) protocol

IPSec Transport Mode

- IP header IPSec data
- Host A
- A to B AH
- A to B ESP
- B to A AH
- B to A AH

IPSec Tunnel Mode

- IP header IPSec data
- Encapsulated packet
- Host A
- A to B AH
- A to B ESP
- B to A AH
- B to A AH
- Host B
- A to B AH
- A to B ESP
- B to A AH
- B to A AH
- GW C
- GW D
- Host A
- A to B AH
- A to B ESP
- B to A AH
- B to A AH
- Host B
- A to B AH
- A to B ESP
- B to A AH
- B to A AH

Connection Protocol

- A channel can be open for terminal sessions, forwarded connections, etc.
- All channels are multiplexed onto a single connection
- Channels can be open from either side
  - Send channel request describing the type of channel
  - Request is granted if such channel is available
Authentication Header

<table>
<thead>
<tr>
<th>Next header</th>
<th>Payload length</th>
<th>Security parameters index</th>
<th>Sequence number</th>
<th>Authentication data</th>
</tr>
</thead>
</table>

Prevents replays

Integrity check vector (ICV): Keyed MAC of:
- Some immutable IP header fields
- AH header
- The rest of the original packet (besides IP header)

Encapsulating Security Payload

IP header

Prevents replays

Security parameters index

Sequence number

The rest of the original packet

Next header

Encrypted with symmetric algorithm

Authentication data

Pretty Good Privacy (PGP)

- Cryptographic program that provides
  - Privacy (encrypts messages)
  - Authentication
  - Message integrity
  - Also enables file encryption
  - Used in E-mail service

PGP Web of Trust

- Every user can sign other user’s certificate, it is upon the recipient to follow certificate chain and decide whether to trust the certificate
- Signature bestows reputation – I signed your key because you are trustworthy
  - If you sign Alice’s key, this will mean that I co-sign it too
  - Once you sign someone’s key (and add it to your list) you indicate level of trust: complete, marginal or none
- Now when you receive key signed by Alice it will be regarded as valid, marginally valid or invalid based on trust
- One valid or two marginally valid signatures are needed to establish a key as valid
S/MIME

- Similar to PGP but uses different encryption algorithms and certificate formats
- Also used for E-mail messages
- Does not have “web of trust” concept