Brief Introduction to SSH

Qinlin Chen, Yicheng Huang, Wei Lai and Shixuan Zhao

Nanjing University

Overview

1. SSH
   - Introduction
   - Session Procedures
2. OpenSSH
   - Introduction
   - User Authentication and Encryption Algorithms
   - Code Analysis

What is SSH?

- Secure Shell
- A remote administration protocol
- Using public-key cryptographic techniques
- Providing a mechanism for authenticating a remote user
- Using the client-server model
- Operating on TCP port 22 by default

Why do we need SSH?

- Two type of encryptions:
  - **Symmetrical encryption**
    A secret key is used for both encryption and decryption of a message.$\rightarrow$ How to store keys securely?
  - **Asymmetrical encryption**
    Public Key & Private Key
- Risk of being attacked by another(have no private key)
  $\rightarrow$ Authentication based on fingerprint
- More secure file transmission
  $\rightarrow$ SFTP & SCP

Before Encrytion Negotiation

- TCP handshake
- Matching encryption protocol and version

Session Encryption Negotiation

- Creation of symmetrical key
  - Using **Diffie-Hellman Key Exchange Algorithm**
  - Used henceforth to encrypt the entire communication session
- DH Algorithm
  1. Alice and Bob agree to use a prime modules p and base g
  2. Alice chooses a secret integer a, sends $A = g^a \mod p$ to Bob
  3. Bob chooses a secret integer b, sends $B = g^b \mod p$ to Alice
  4. Alice computes $s = B^a \mod p$
  5. Bob computes $s = A^b \mod p$
  6. Now Alice and Bob share a secret s because

$$A^b = (g^a)^b = (g^b)^a = B^a \mod p$$

User Authentication

- passwards: not recommended
- **SSH Key Pairs**
  1. First of all, client should add its public key into authorized keys of server.
  2. Client $\xrightarrow{\text{ID for key pair (login request)}}$ Server
  3. Server searchs for a public key matching ID and generate a random number R, encrypting it with the public key.
  4. Client $\xleftarrow{\text{pubKey}(R)}$ Server
  5. Client decryptes it with its private key, and combines it with the shared session key, calculating the MD5 hash of this value.
  6. Client $\xrightarrow{\text{Digest1}}$ Server
  7. Server does Step 5 and compares Digest1 with Digest2. If matching, then the client is authenticated.

Communication

- **SFTP**
  Secure file transfer program (FTP-like) that works over SSH1 and SSH2 protocol
- **SCP**
  Remote file copy program that acts like rcp
- **rsync**
  Intended to be more efficient than SCP
- ...

Introduction to OpenSSH

- A suite of secure networking utilities based on the Secure Shell protocol, which provides a secure channel over an unsecured network in a client–server architecture.
- Including the following command-line utilities and daemons
  - **scp** replacement for rcp
  - **sftp** replacement for ftp to copy files between computers
  - **ssh** replacement for rlogin, rsh and telnet
  - **ssh-add** and **ssh-agent** utilities to ease authentication by holding keys ready and avoid the need to enter passphrases every time they are used
  - **ssh-keygen** a tool to inspect and generate the RSA, DSA and Elliptic Curve keys that are used for authentication
  - **ssh-keyscan** scanning a list of hosts and collecting their public keys

User Authentication

Files to store private keys

~/.ssh/id_dsa
~/.ssh/id_ecdsa
~/.ssh/id_ed25519
~/.ssh/id_rsa

File to store client fingerprints that have been authorized

~/.ssh/authorized_keys

Files to store public keys

~/.ssh/id_dsa.pub
~/.ssh/id_ecdsa.pub
~/.ssh/id_ed25519.pub
~/.ssh/id_rsa.pub

File to store ID of remote hosts that have been authorized

~/.ssh/known_hosts

Encryption Algorithm

OpenSSH supports several algorithms like RSA, DSA and ECDSA. If not including -t option, ssh-keygen will generate RSA keys. Here we take RSA as an example. It involves 4 steps.

1. Key generation
2. Key distribution
3. Encryption
4. Decryption

RSA

- Key generation
  1. Choose two distinct prime number p and q.
  2. Compute $n = pq$.
  3. Compute $\lambda(n) = \mathrm{lcm}(p-1, q-1)$, where λ is Carmichael's totient function.
  4. Choose an integer e such that $1 < e < \lambda(n)$ and $\gcd(e, \lambda(n)) = 1$
  5. Determine d as $d \equiv e^{-1} \mod \lambda(n)$

  Now, we have public keys e, n and private key d.
- Key distribution

  To enable Bob to send his encrypted messages, Alice transmits her public key (n, e) to Bob via a reliable but not necessarily secret route. Alice's private key (d) is never distributed.

RSA

- Encryption

  After Bob obtains Alice's public key, he can send a message M to Alice. He firstly turns M into an integer m so that $0 \leq m < n$ with padding scheme. He then computes the ciphertext c using public key e corresponding to

$$c \equiv m^e \mod n$$

- Decryption

  Alice can recover $m$from$c$ by using her private key exponent d by computing

$$c^d \equiv (m^e)^d \equiv m \mod n$$

By reversing the padding scheme, she can recover M from m.

Methods of user authetication in OpenSSH

1. The client uses ssh-keygen to generate a pair of keys.

```
ssh-keygen [-q] [-b bits] [-t type]
[-f output_keyfile] [-P passphrase]
-t {rsa|ecdsa|dsa}
```

2. Then it use ssh-copy-id to send the public key to the directory of remote server.

```
ssh-copy-id [-i [identity_file]] [-p port]
[-o ssh_option] [user@hostname]
```

Behavior of ssh-keygen

Since options are complex, we only analyze the default mode. The ssh-keygen illustrates the public-key allocation and encryption strategies.

Behavior of ssh-keygen

Firstly, the ssh-keygen calls

sshkey_generate(int type, u_int bits, struct sshkey **keyp);

If OpenSSL (Open Secure Sockets Laye) is used, then we call

rsa_generate_private_key(bits, &k->rsa);

So we have the method oriented for RSA-key's generation. By calling RSA_new(), we get a new RSA key, which is stored in variable private. The RSA_new() is implemented in library <openssl/rsa.h>.

Now, a private key is successfully generated.

Behavior of ssh-keygen

Secondly, the ssh-keygen calls

int sshkey_from_private(const struct sshkey *k, struct sshkey **pkp);

to generate the public key matched with the private one, which is stored in variable public. Then, the key pairs will be wrapped with passphrase and comment and stored in files.

int sshkey_save_private(struct sshkey *key, const char *filename, const char *passphrase, const char *comment, int force_new_format, const char *new_format_cipher, int new_format_rounds)

Behavior of ssh-keygen

Encryption algorithms are called by many methods. One example is a method using for reading a private key from buffer.

```
static struct sshkey *
do_convert_private_ssh2_from_blob(u_char *blob, u_int
blen);
```

Behavior of ssh-ketgen

```
buffer_get_bignum_bits(b, rsa_d);
buffer_get_bignum_bits(b, rsa_n);
buffer_get_bignum_bits(b, rsa_iqmp);
buffer_get_bignum_bits(b, rsa_q);
buffer_get_bignum_bits(b, rsa_p);
if (!RSA_set0_key(key->rsa, rsa_n, rsa_e, rsa_d))
    fatal("%s: RSA_set0_key failed", __func__);
rsa_n = rsa_e = rsa_d = NULL; /* transferred */
if (!RSA_set0_factors(key->rsa, rsa_p, rsa_q))
    fatal("%s: RSA_set0_factors failed", __func__);
rsa_p = rsa_q = NULL; /* transferred */
if ((r = ssh_rsa_complete_crt_parameters(key, rsa_iqmp)) != 0)
    fatal("generate RSA parameters failed: %s", ssh_err(r));
```

It's easy to see that variables and factors like *d*, *n*, *p*, *q* of RSA are parsed from buffer. library function RSA_set0_key and RSA_set0_factors are called to set key->rsa.

The End