Do you really understand SSH?

I do not.

@gaocegege
2022-11-18

About TensorChord Tea Hour

- 2022 TensorChord Tea Hour
- https://meeting.tencent.com/dm/SuauU1xzj2ZY

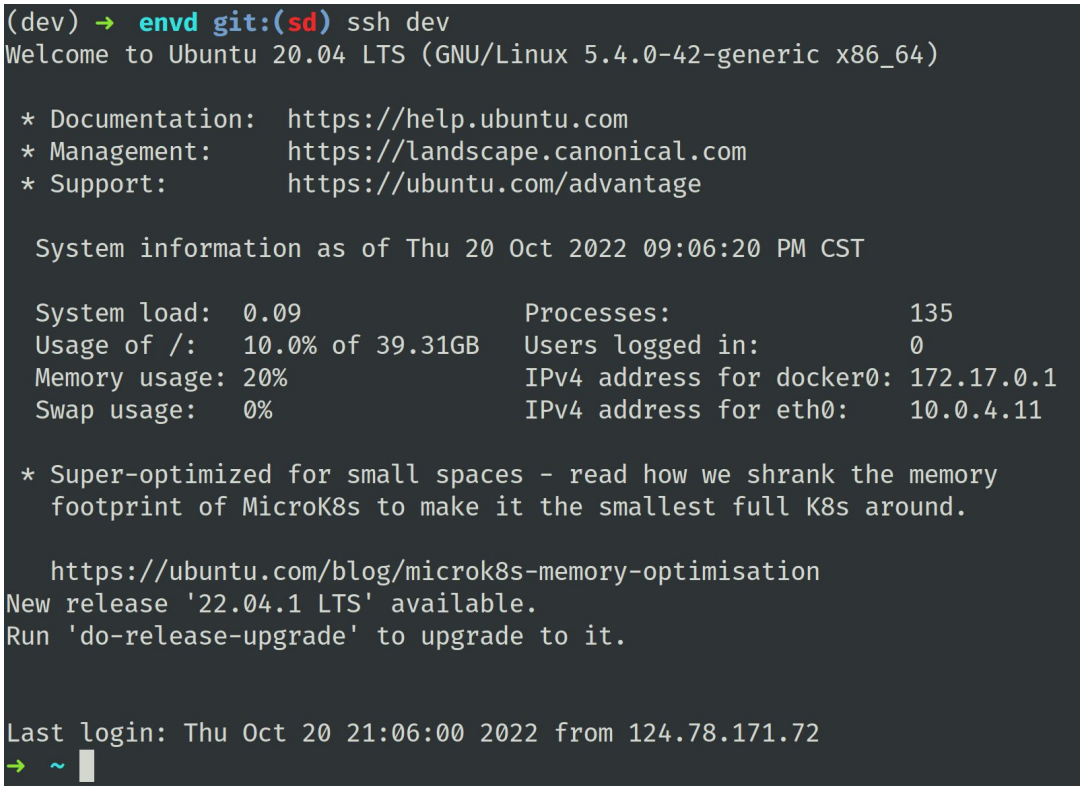
Ce Gao @TensorChord

- **kubeflow** co-chair, mainly focued on training-operator and katib
- **tensorchord/envd** maintainer

gaocegege.com @gaocegege

What is SSH?

SSH



@TensorChord

SSH in envd

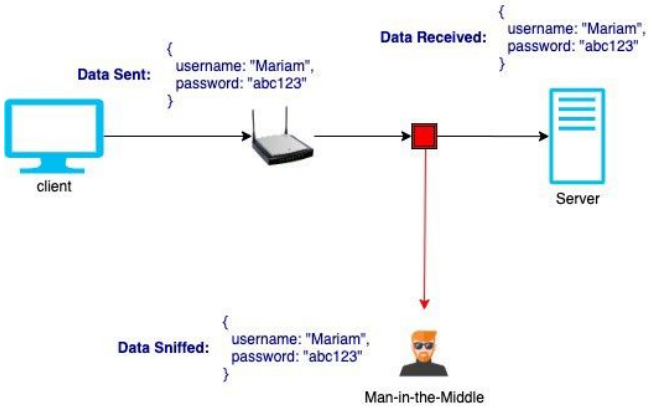
@TensorChord

SSH in envd

```
# entry generated by envd
Host fairydirt.envd
  ForwardAgent yes
  PubkeyAcceptedKeyTypes +ssh-rsa
  HostKeyAlgorithms +ssh-rsa
  HostName localhost
  Port 2222
  UserKnownHostsFile /dev/null
  IdentityFile "/home/gaocegege/.config/envd/id_rsa_envd"
  StrictHostKeyChecking no
  User a332139d39b89a241400013700e665a3/fairydirt
```

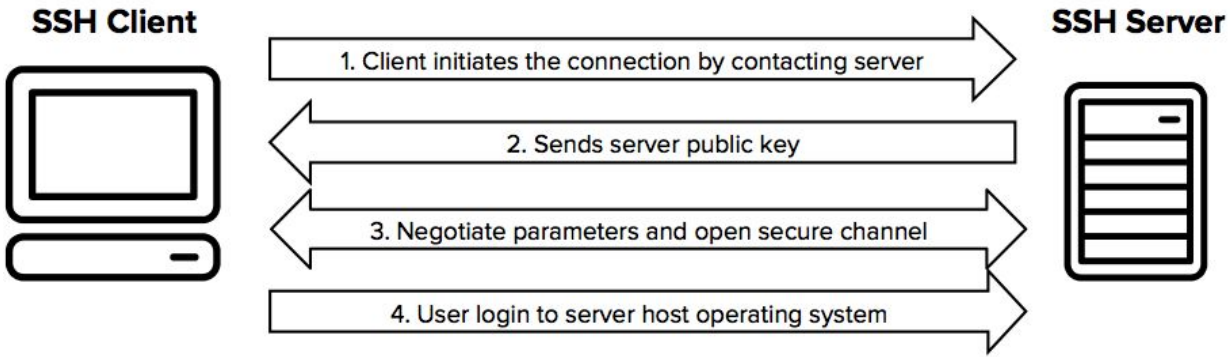
History

- 1995.06 SSH1, by Ylönen
- 1995.12 SSH Communications Security Corp. (SCS) with 20k users ssh.com
- 1997.02 SSH2.0 protocol first draft
- 1998 SSH2 product released
- 1999 OpenSSH, fork of SSH 1.2.12
- 2006 SSH2.0 RFC



SSH

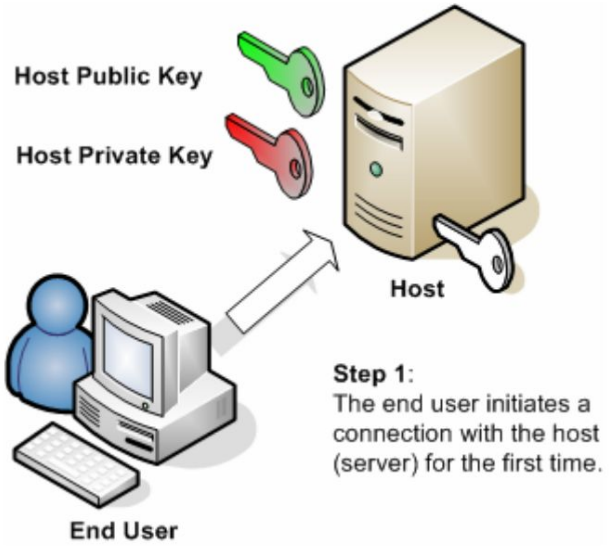
```
ssh {user}@{server} -p {port} -i {key}
```



@TensorChord

Host Key (RFC 4251)

- Each server host SHOULD have a host key.
- The server host key is used **during key exchange** to verify that the client is really talking to **the correct server**.

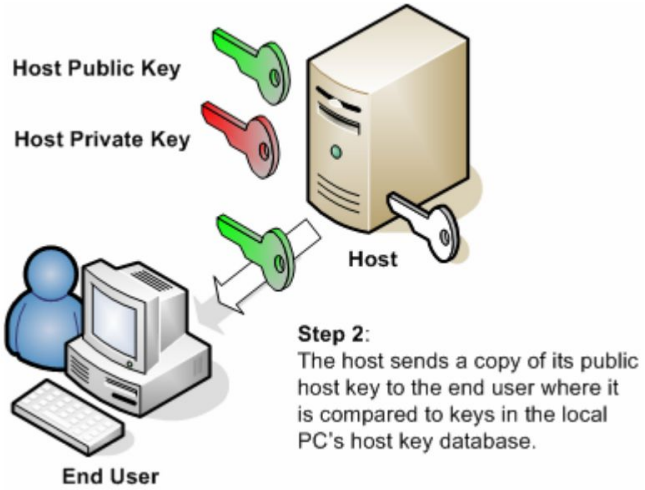



```
The authenticity of host 'xx.xx.xx.xx' can't be established.
ECDSA key fingerprint is SHA256:hRpUhnmlrEqZEChIuKicesJLfGAZR+n4PfAt3ywJIxw.
Are you sure you want to continue connecting (yes/no/[fingerprint])?
>
```

@TensorChord

Host Key (RFC 4251)

- Each server host SHOULD have a host key.
- The server host key is used **during key exchange** to verify that the client is really talking to **the correct server**.

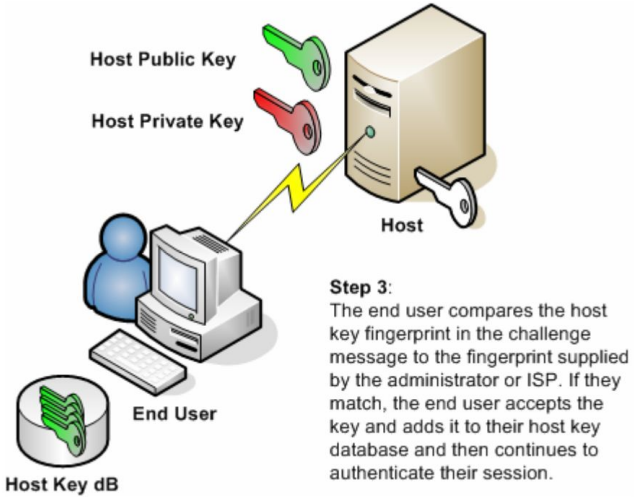



```
The authenticity of host 'xx.xx.xx.xx' can't be established.
ECDSA key fingerprint is SHA256:hRpUhnmlrEqZEChIuKicesJLfGAZR+n4PfAt3ywJIxw.
Are you sure you want to continue connecting (yes/no/[fingerprint])?
>
```

@TensorChord

Host Key (RFC 4251)

- Each server host SHOULD have a host key.
- The server host key is used **during key exchange** to verify that the client is really talking to **the correct server**.




```
The authenticity of host 'xx.xx.xx.xx' can't be established.
ECDSA key fingerprint is SHA256:hRpUhnmlrEqZEChIuKicesJLfGAZR+n4PfAt3ywJIxw.
Are you sure you want to continue connecting (yes/no/[fingerprint])?
>
```

@TensorChord

Host Key (RFC 4251)

How to verify?

- SMS your administrator
- An ISP or network administrator might distribute host key fingerprints on a secure web page that all customers or users have access to.
- The host key fingerprint can be sent by e-mail to end users so they have it readily available to compare to the fingerprint displayed in the challenge message.
- For enterprises that already use a system such as SMS to push files out to client systems, host keys could also be distributed through this system.
- Organizations using Kerberos could take advantage of Secure Shell's GSSAPI key exchange which doesn't require hosts keys and instead leverages Kerberos host verification. (RFC 4462)

Host Key (RFC 4251)

How to keep?

- ~/.ssh/known_hosts is the local database
- Another way: The host name-to-key association is certified by a trusted certification authority (CA). The client only knows the CA root key, and can verify the validity of all host keys certified by accepted CAs.

Authentication (RFC 4252)

```
func main() {
    // ssh config
    hostKeyCallback, err := knownhosts.New("/home/debian11/.ssh/known_hosts")
    if err ≠ nil {
        log.Fatal(err)
    }
    config := &ssh.ClientConfig{
        User: "ubuntu",
        Auth: []ssh.AuthMethod{
            ssh.PublicKeys(signer),
        },
        HostKeyCallback: hostKeyCallback,
    }
    // connect ot ssh server
    conn, err := ssh.Dial("tcp", "192.168.205.217:22", config)
    if err ≠ nil {
        log.Fatal(err)
    }
    defer conn.Close()
}
```

Authentication (RFC 4252)

```
ssh {user}@{server} p {port} -vvv
```

```
Client
byte      SSH_MSG_USERAUTH_REQUEST (50)
string    user name
string    service name (ssh-connection)
string    "publickey"
boolean   TRUE (partial success)
string    public key algorithm name
string    public key to be used for authentication
string    signature
```

Authentication (RFC 4252)

```
ssh {user}@{server} p {port} -vvv
```

Server

byte	SSH_MSG_USERAUTH_FAILURE
name-list	authentications that can continue (publickey)
boolean	partial success

```
debug1: Offering public key: gaocegege@hotmail.com ED25519 SHA256:Kegz5ir57Z9s23GKEsqJM1LIKPJpWZ
yPGFMYeGeNxgM agent
debug3: send packet: type 50
debug2: we sent a publickey packet, wait for reply
debug3: receive packet: type 51 (SSH_MSG_USERAUTH_FAILURE)
debug1: Authentications that can continue: publickey
```

@TensorChord

Authentication (RFC 4252)

```
ssh {user}@{server} p {port} -vvv
```

Client

```
byte      SSH_MSG_USERAUTH_REQUEST (50)
string    user name
string    service name (ssh-connection)
string    "publickey"
boolean   TRUE (partial success)
string    public key algorithm name
string    public key to be used for authentication
string    signature
```

```
debug1: Trying private key: /home/gaocegege/.ssh/azure_rsa
debug3: sign_and_send_pubkey: RSA SHA256:wk3Ux0b2T4UFZ3iPjfzdjuPT82AM0t+4s0wSt5Oxwv0
debug3: sign_and_send_pubkey: signing using rsa-sha2-512 SHA256:wk3Ux0b2T4UFZ3iPjfzdjuPT82AM0t+4
s0wSt5Oxwv0
debug3: send packet: type 50
debug2: we sent a publickey packet, wait for reply
debug3: receive packet: type 52 (SSH_MSG_USERAUTH_SUCCESS)
debug1: Authentication succeeded (publickey).
```

@TensorChord

Authentication (RFC 4252)

```
// manually wrap the serialized signature in a string
s := Marshal(sign)
sig := make([]byte, stringLength(len(s)))
marshalString(sig, s)
msg := publickeyAuthMsg{
       User:     user,
       Service:  serviceSSH,
       Method:   cb.method(),
       HasSig:   true,
       Algoname: algo,
       PubKey:   pubKey,
       Sig:      sig,
}
p := Marshal(&msg)
if err := c.writePacket(p); err ≠ nil {
       return authFailure, nil, err
}
var success authResult
success, methods, err = handleAuthResponse(c)
if err ≠ nil {
       return authFailure, nil, err
}

// If authentication succeeds or the list of available methods does not
// contain the "publickey" method, do not attempt to authenticate with any
// other keys.  According to RFC 4252 Section 7, the latter can occur when
// additional authentication methods are required.
if success == authSuccess || !containsMethod(methods, cb.method()) {
       return success, methods, err
}
```

Client	
byte	SSH_MSG_USERAUTH_REQUEST (50)
string	user name
string	service name (ssh-connection)
string	"publickey"
boolean	TRUE (partial success)
string	public key algorithm name
string	public key to be used for authentication
string	signature

@TensorChord

Connection (RFC 4254)

```
// NewSession opens a new Session for this client. (A session is a remote
// execution of a program.)
func (c *Client) NewSession() (*Session, error) {
	ch, in, err := c.OpenChannel("session", nil)
	if err ≠ nil {
		return nil, err
	}
	return newSession(ch, in)
}
```

Connection (RFC 4254)

Two types of messages are sent (OpenSSH):

- Global request (no-more-sessions@openssh.com) for security
  - several kinds of requests that affect the state of the remote end globally, independent of any channels.
- **Channel**

```
debug1: channel 0: new [client-session]
debug3: ssh_session2_open: channel_new: 0
debug2: channel 0: send open
debug3: send packet: type 90 (SSH_MSG_CHANNEL_OPEN)
debug1: Requesting no-more-sessions@openssh.com
debug3: send packet: type 80 (SSH_MSG_GLOBAL_REQUEST)
```

Connection (RFC 4254) - Open session channel

Client

```
byte      SSH_MSG_CHANNEL_OPEN (90)
string    channel type in US-ASCII only
uint32    sender channel
uint32    initial window size
uint32    maximum packet size
....      channel type specific data follows
```

Channel type	Reference
session	[SSH-CONNECT, Section 6.1]
x11	[SSH-CONNECT, Section 6.3.2]
forwarded-tcpip	[SSH-CONNECT, Section 7.2]
direct-tcpip	[SSH-CONNECT, Section 7.2]

```
byte      SSH_MSG_CHANNEL_OPEN (90)
string    "session"
uint32    sender channel
uint32    initial window size
uint32    maximum packet size
```

Server

```
byte      SSH_MSG_CHANNEL_OPEN_CONFIRMATION (91)
uint32    recipient channel
uint32    sender channel
uint32    initial window size
uint32    maximum packet size
....      channel type specific data follows
```

@TensorChord

Connection (RFC 4254)

```
func (m *mux) openChannel(chanType string, extra []byte) (*channel, error) {
	ch := m.newChannel(chanType, channelOutbound, extra)

	ch.maxIncomingPayload = channelMaxPacket

	open := channelOpenMsg{
		ChanType:         chanType,
		PeersWindow:      ch.myWindow,
		MaxPacketSize:    ch.maxIncomingPayload,
		TypeSpecificData: extra,
		PeersID:          ch.localId,
	}
	if err := m.sendMessage(open); err != nil {
		return nil, err
	}

	switch msg := (<-ch.msg).(type) {
	case *channelOpenConfirmMsg:
		return ch, nil
	case *channelOpenFailureMsg:
		return nil, &OpenChannelError{msg.Reason, msg.Message}
	default:
		return nil, fmt.Errorf("ssh: unexpected packet in response to channel open: %T", msg)
	}
}
```

@TensorChord

Connection (RFC 4254) - Open pseudo-terminal

```
byte      SSH_MSG_CHANNEL_REQUEST (98)
uint32    recipient channel
string    "pty-req"
boolean   want_reply
string    TERM environment variable value (e.g.,
vt100)
uint32    terminal width, characters (e.g., 80)
uint32    terminal height, rows (e.g., 24)
uint32    terminal width, pixels (e.g., 640)
uint32    terminal height, pixels (e.g., 480)
string    encoded terminal modes
```

Request type	Reference
pty-req	[SSH-CONNECT, Section 6.2]
x11-req	[SSH-CONNECT, Section 6.3.1]
env	[SSH-CONNECT, Section 6.4]
shell	[SSH-CONNECT, Section 6.5]
exec	[SSH-CONNECT, Section 6.5]
subsystem	[SSH-CONNECT, Section 6.5]
window-change	[SSH-CONNECT, Section 6.7]
xon-xoff	[SSH-CONNECT, Section 6.8]
signal	[SSH-CONNECT, Section 6.9]
exit-status	[SSH-CONNECT, Section 6.10]
exit-signal	[SSH-CONNECT, Section 6.10]

Connection (RFC 4254) - Open pseudo-terminal

```
// configure terminal mode
 modes := ssh.TerminalModes{
     ssh.ECHO:          0,     // supress echo

 }
 // run terminal session
 if err := session.RequestPty("xterm", 50, 80, modes); err ≠ nil {
     log.Fatal(err)
 }
 // start remote shell
 if err := session.Shell(); err ≠ nil {
     log.Fatal(err)
 }
```

Connection (RFC 4254) - Open pseudo-terminal

```
// RequestPty requests the association of a pty with the session on the remote host.
func (s *Session) RequestPty(term string, h, w int, termmodes TerminalModes) error {
	var tm []byte
	for k, v := range termmodes {
		kv := struct {
			Key byte
			Val uint32
		}{k, v}

		tm = append(tm, Marshal(&kv)...)
	}
	tm = append(tm, tty_OP_END)
	req := ptyRequestMsg{
		Term:     term,
		Columns:  uint32(w),
		Rows:     uint32(h),
		Width:    uint32(w * 8),
		Height:   uint32(h * 8),
		Modelist: string(tm),
	}
	ok, err := s.ch.SendRequest("pty-req", true, Marshal(&req))
	if err == nil && !ok {
		err = errors.New("ssh: pty-req failed")
	}
	return err
}
```

```
byte      SSH_MSG_CHANNEL_REQUEST (98)
uint32    recipient channel
string    "pty-req"
boolean   want_reply
string    TERM environment variable value
uint32    terminal width, characters (e.g., 80)
uint32    terminal height, rows (e.g., 24)
uint32    terminal width, pixels (e.g., 640)
uint32    terminal height, pixels (e.g., 480)
string    encoded terminal modes
```

Connection (RFC 4254) - Open pseudo-terminal

```
// Shell starts a login shell on the remote host. A Session only
// accepts one call to Run, Start, Shell, Output, or CombinedOutput.
func (s *Session) Shell() error {
	if s.started {
		return errors.New("ssh: session already started")
	}

	ok, err := s.ch.SendRequest("shell", true, nil)
	if err == nil && !ok {
		return errors.New("ssh: could not start shell")
	}
	if err != nil {
		return err
	}
	return s.start()
}
```

```
byte      SSH_MSG_CHANNEL_REQUEST
uint32    recipient channel
string    "shell"
boolean   want reply
```

@TensorChord

Connection (RFC 4254)

no-more-sessions@openssh.com

Most SSH connections will only ever request a single session, but a attacker may abuse a running ssh client to s**urreptitiously open additional sessions under their control**.

OpenSSH provides a global request "no-more-sessions@openssh.com" to mitigate this attack.

Fun with SSH

```
$ sshfs [username]@[hostname]:[hostpath] [mountpoint]
$ fusermount -u [mountpoint]

$ scp [username]@[hostname]:[source file] [username]@[hostname]:[destination
directory]
# RFC 913
```

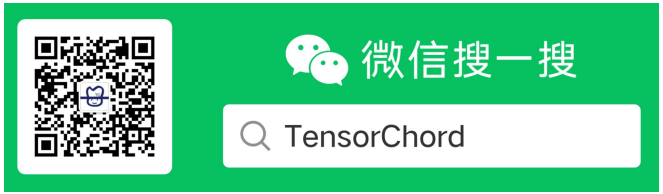
@TensorChord

RFCs

- https://containerssh.io/development/containerssh/ssh/#rfcs

Thanks 🎉

- Follow @TensorChord on Twitter
- Join our Discord community!



We are hiring 👀 !

- buildkit / nerdctl / buildah / podman / kubernetes / …
- kubeflow / mlflow / flyte / weights&biases / …

Send a mail to cegao@tensorchord.ai!