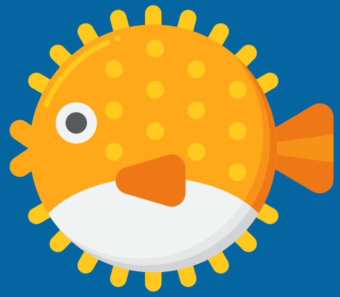
LuxCamp 2023 Edition
https://luxeria.ch

SSH - Secure Shell

Attacks and Best-Practices in 2023

04.08.2023, LuxCamp 2023 der LuXeria, Brüttisellen

Emanuel Duss <emanuel.duss@compass-security.com>



Intro

- Pentests & Security Assessments
- Linux Hardening Reviews
- Topics
  - SSH Introduction
  - Service Exposure
  - Information Disclosure
  - SSH Authentication (Host, User, CAs, 2FA, FIDO2)
  - SSH Port Forwarding
  - SSH Agent
  - SSH Session Multiplexing
  - Cryptographic Algorithms

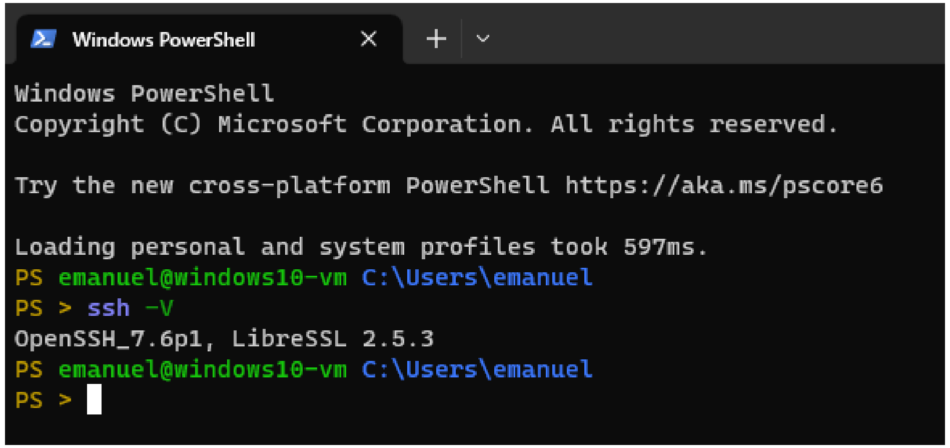
Condensed
1 Hour Version 🧐

SSH Introduction

Secure Shell

- Establish authenticated & encrypted network connection to remote systems
- Used for
  - Remote login / shell access
  - Data transfer
  - Port forwarding
  - Traffic tunneling, proxying

- History
  - Replacement for plaintext protocols (`rsh`, `rlogin`, `telnet`, `ftp`, …)
  - Server / Client Architecture
  - SSH version 1 in 1995
  - SSH version 2 in 2006, standardized in various RFCs
  - OpenSSH is one implementation 🐡
  - Available on all major Linux/Unix OS
  - Available for Windows since 2017



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

Loading personal and system profiles took 597ms.
PS emanuel@windows10-vm C:\Users\emanuel
PS > ssh -V
OpenSSH_7.6p1, LibreSSL 2.5.3
PS emanuel@windows10-vm C:\Users\emanuel
PS >
```

SSH Tools & Files

- Tools
  - Remote Operations: `ssh`, `scp`, `sftp`
  - Key Management: `ssh-add`, `ssh-keyscan`, `ssh-keygen`, `ssh-keysign`
  - Server side: `sshd`, `sftp-server`, `ssh-agent`
- Files
  - Server config: `/etc/ssh/sshd_config` and `/etc/ssh/sshd_config.d/`
  - Global client config: `/etc/ssh/ssh_config` and `/etc/ssh/ssh_config.d/`
  - Personal client config: `~/.ssh/config`
- Manpages
  - Everything is lovely documented!
  - `ssh(1)`, `ssh-add(1)`, `ssh-agent(1)`, `ssh-copy-id(1)`, `ssh-keygen(1)`, `ssh-keyscan(1)`, `ssh-keysign(8)`, `ssh-pkcs11-helper(8)`, `ssh_config(5)`, `sshd(8)`, `sshd_config(5)`

SSH Commands

- Establish remote shell session

```
alice@beastie:~$ ssh puffy
Welcome to puffy.
alice@puffy:~$
```

- Execute command on remote system

```
alice@beastie:~$ ssh puffy id
Welcome to puffy.
uid=1001(bob) gid=1001(bob) groups=1001(bob),27(sudo)
```

- Copy files remotely

```
alice@beastie:~$ scp puffy:/etc/ssh/sshd_config .
alice@beastie:~$ scp .ssh/known_hosts puffy:.ssh/
alice@beastie:~$ scp puffy:notes aix:/tmp/
```

SSH Commands

- Local port forwarding

```
alice@beastie:~$ ssh -L 1234:localhost:8080 puffy
alice@beastie:~$ ssh -L 0.0.0.0:1234:10.5.23.52:8080 puffy
```

- Remote port forwarding

```
alice@beastie:~$ ssh -R 8080:10.5.23.42:8080 puffy
alice@beastie:~$ ssh -R 0.0.0.0:8080:localhost:8080 puffy
                                            # Requires 'gatewayports clientspecified'
```

- Create SOCKS proxy on the local host for tunneling traffic through remote host

```
alice@beastie:~$ ssh -D 1080 puffy
```

- Create SOCKS proxy on the remote host for tunneling traffic through local host

```
alice@beastie:~$ ssh -R 1080 puffy
```

- **This is not a talk about SSH tricks and ninja magic. This would fill several other talks 🤓!**

Useful Use-Case for Pentests

- Situation
  - You got a notebook of a customer for an internal pentest
  - The internal pentest is performed remotely using the VPN client on the notebook
  - The notebook has all the latest and fancy anti-malware / EDR software installed
- Poor analyst's problem
  - You can't use your `$TOOLS` from your Kali VM or on the customer's notebook 😢
- Solution: SSH to the rescue! 🐡 🎉
  - Connect the notebook to your testing network where your testing VM is
  - Use SSH from the notebook to create a SOCKS proxy on your attacker machine
  - You can then access the corporate network from your attacker machine

Useful Use-Case for Pentests

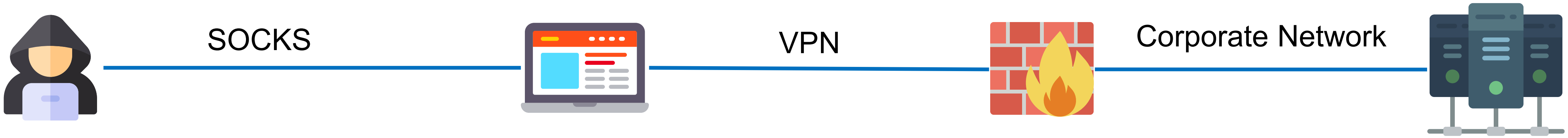
- Connect notebook to own network and execute:

```
PS domainuser@notebook C:\> ssh -R 1080 kali
```

- New SOCKS proxy on your attacker kali:

```
attacker@kali:~ $ sudo ss -ltpn sport = 1080 | cat
State  Recv-Q Send-Q Local Address:Port Peer Address:PortProcess
LISTEN 0      128          127.0.0.1:1080      0.0.0.0:*    users:(("sshd",pid=8169,fd=9))
LISTEN 0      128              [::1]:1080         [::]:*    users:(("sshd",pid=8169,fd=7))
```

- Created tunnel:



- You can now access the customer’s network (SOCKS limitations apply):

```
attacker@kali:~ $ proxychains crackmapexec smb -u alice -p s3cret -d example.net
dc.example.net
[...]
```

SSH Server Commands

- Show running SSH server configuration:

```
alice@beastie:~$ sudo sshd -T
port 22
addressfamily any
listenaddress [::]:22
[...]
```

Useful for hardening reviews!

- Test SSH server configuration:

```
alice@beastie:~$ /usr/sbin/sshd -t
/etc/ssh/sshd_config: line 18: Bad configuration option: ThisOptionDoesNotExist
/etc/ssh/sshd_config: terminating, 1 bad configuration options
```

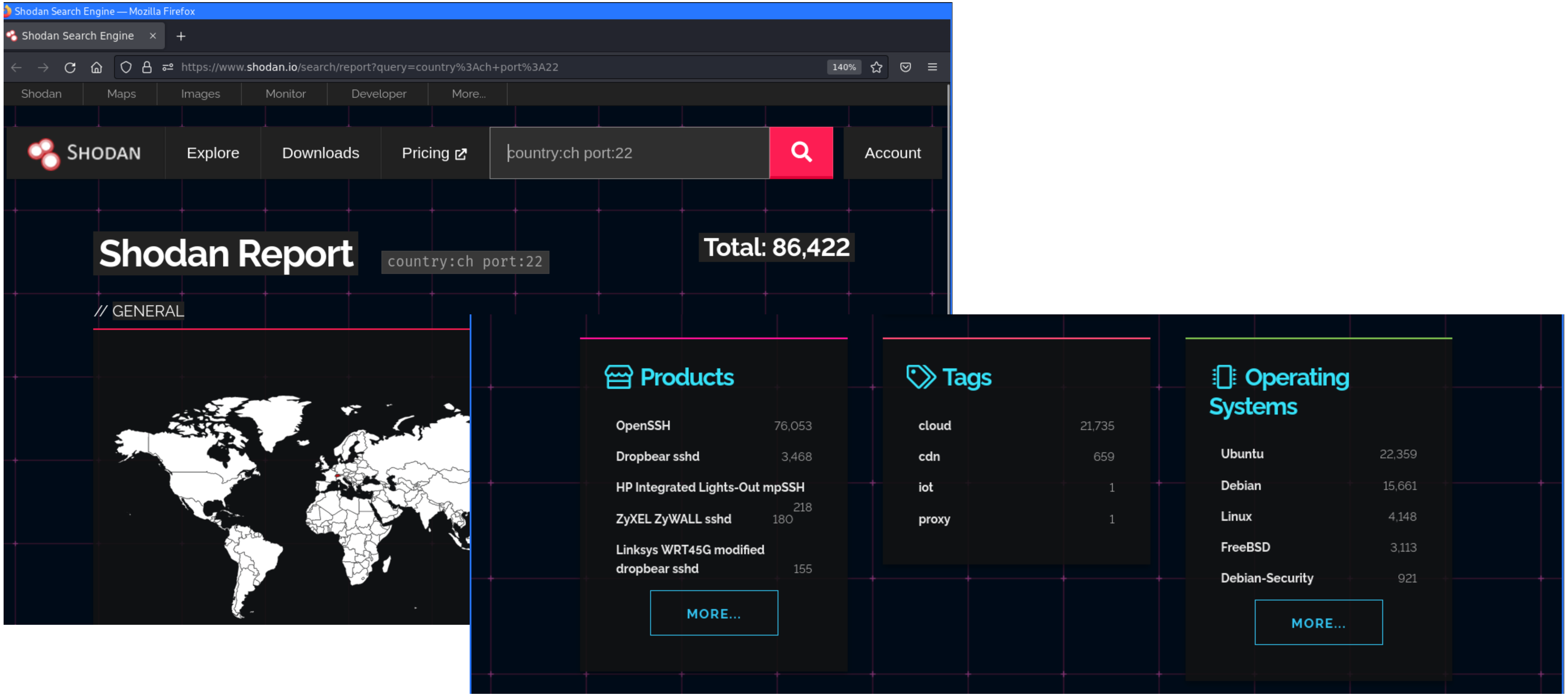
- Start SSH server in debug mode:

```
alice@beastie:~$ sudo /usr/sbin/sshd -d
debug1: sshd version OpenSSH_8.4, OpenSSL 1.1.1n  15 Mar 2022
debug1: Bind to port 22 on 0.0.0.0.
[...]
```

Service Exposure

Service Exposure

- The SSH server runs on port `22/tcp` by default.
- They can easily be found.



Shodan Report `country:ch port:22`

Total: 86,422

// GENERAL

Products

OpenSSH	76,053
Dropbear sshd	3,468
HP Integrated Lights-Out mpSSH	218
ZyXEL ZyWALL sshd	180
Linksys WRT45G modified dropbear sshd	155

MORE...

Tags

cloud	21,735
cdn	659
iot	1
proxy	1

Operating Systems

Ubuntu	22,359
Debian	15,661
Linux	4,148
FreeBSD	3,113
Debian-Security	921

MORE...

Service Exposure

- **Attack**
  - When exposed to the Internet, external attackers can easily find your SSH servers.
  - They can then perform further attacks on this system.
- **Recommendations**
  - Only expose your servers when necessary.
  - Only expose your servers to allowed IP addresses when possible.
- **Note**
  - It’s possible to “hide” the server by using a random high port or port knocking.
  - This is security by obscurity and should not be used for security reasons.
  - Can be used to prevent non-targeted attacks from script kiddies.
  - Can be done, but security should not rely on this.
  - Instead, the system should be correctly configured and managed.
  - This includes hardening, patching, network segregation, logging, monitoring, alerting, …
  - System events, file manipulation, firewall rules, user behavior, login sources, failed/successful logins, …

Information Disclosure

Information Disclosure

- The SSH version banner can be grabbed unauthenticated

```
$ ncat puffy.example.net 22
SSH-2.0-OpenSSH_8.4p1 Debian-5+deb11u1
```

- **Attack**
  - An attacker could gain information about the system and perform targeted attacks.
- **Recommendations**
  - Hide what’s possible.
  - But again: the security should not rely on hiding information!
  - Instead, patch your systems!
- The banner can’t be disabled via SSH server config.
- Debian can suppress some information:

```
DebianBanner no
```

- Result

```
$ ncat debian.example.net 22
SSH-2.0-OpenSSH_8.4p1
```

SSH Authentication

Host Authentication

- Users must authenticate the server.
- The host has one or more host keys (ECDSA, Ed25519, RSA).
- Alternatively, an SSH certificate authority (CA) can be used.
- On first connection, a host key fingerprint his shown:

```
alice@beastie:~$ ssh puffy
The authenticity of host ' puffy (203.0.113.23)' can't be established.
ED25519 key fingerprint is SHA256:aPDwXPsHTWTSebUW3jPkb4nH/lUGmvILmQsEkXKsY9c.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

- If accepted, host key is stored in users' known hosts file `~/.ssh/known_hosts`.
- These host keys are then trusted for future connections.
- TOFU: Trust On First Use principle
- Users generally don’t verify this fingerprint.

Host Authentication

- Host keys can be stored in DNS (SSHFP resource record)

```
puffy.example.net IN SSHFP 4 2 5b7629b7b5906567aaf57b[...]f96079c3
```

- SSH clients can use SSHFP records this to verify host key

```
VerifyHostKeyDNS ask # or 'yes' to automatically accept
```

- Example session

```
alice@beastie:~$ ssh puffy
[...]
Matching host key fingerprint found in DNS.
Are you sure you want to continue connecting (yes/no)?
```

- Not always trustworthy
  - Without DNSSEC, the DNS resolver can’t verify authenticity of SSHFP record.
  - Without DNSSEC or DNS over TLS (DoT), a client can’t trust DNS resolver.

Host Authentication

- **Attack**
  - When a user accepts an arbitrary host key, an attacker between client & server can sniff and manipulate the network traffic (like credentials) or let the user connect to an untrusted system.
- **Recommendation**
  - Users should NEVER have to verify host keys themselves, since they don’t do it properly.
  - A centrally managed known hosts file should be used (default is `/etc/ssh/ssh_known_hosts`)
  - Alternatively, an own SSH key CA could be used.

User Authentication

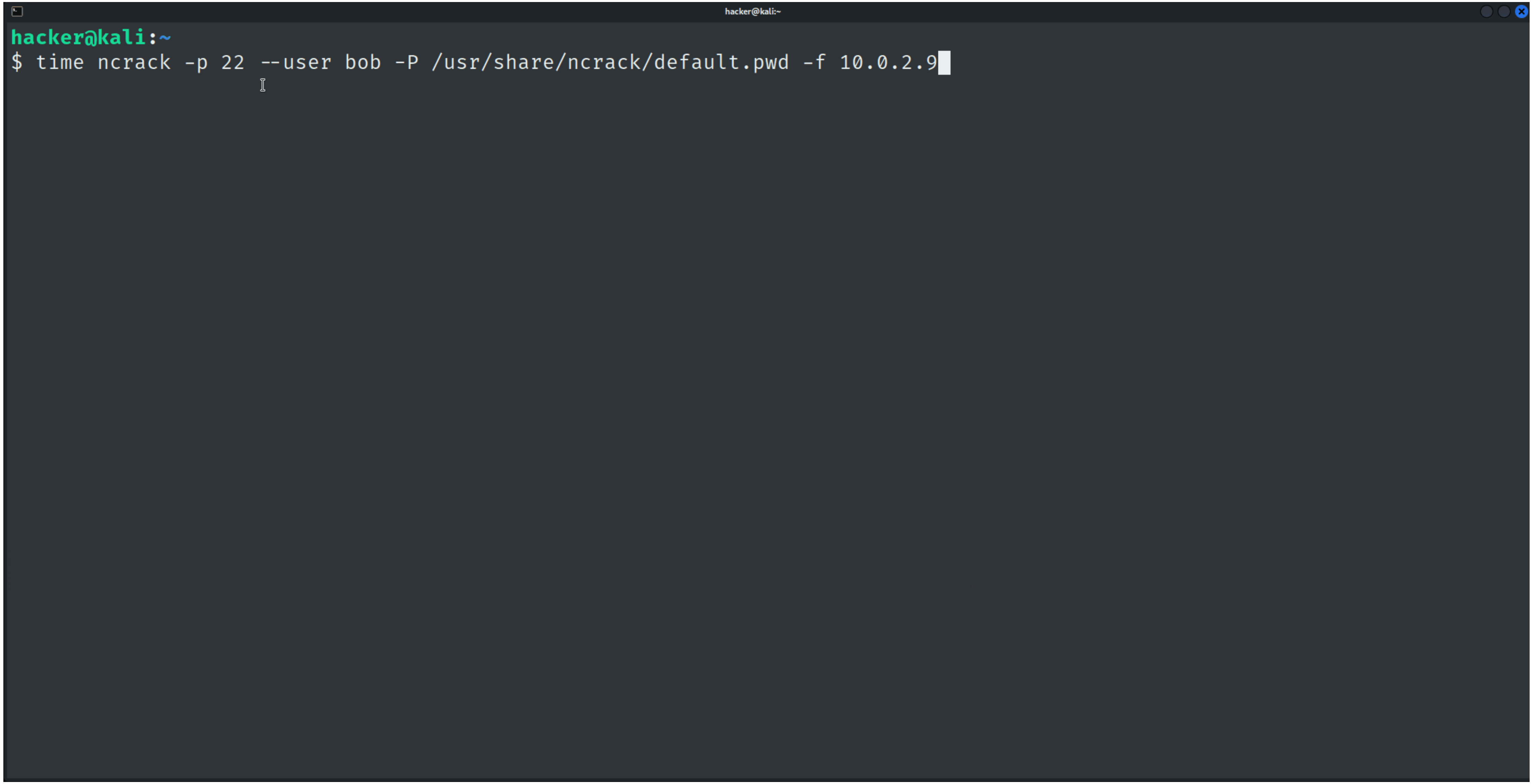
- Different types and combinations of user authentication methods
  - Host-based authentication
  - Password authentication
  - Public key authentication
  - GSSAPI (used for single sign-on like Kerberos or NTLM)
  - Keyboard interactive (via PAM, used e.g. for 2FA)
  - Combination using either public key & password or public key & 2FA
- Example server config (`sshd_config`)

```
AuthenticationMethods password # Password only
AuthenticationMethods publickey # Public key only
AuthenticationMethods keyboard-interactive # Authentication via PAM
AuthenticationMethods publickey,password publickey,keyboard-interactive # 2FA
AuthenticationMethods publickey,publickey # Two different public keys
```

User Authentication

- **Attack**
  - When password authentication is enabled, attackers can try to online brute-force passwords.
- **Recommendation**
  - Generally, if the password is strong (long and random), password authentication is OK.
  - However, users tend to choose weak passwords.
  - Furthermore, passwords may leak through data breaches, phishing attacks, password reuse, …
  - Therefore, enforce public key authentication or 2FA.
  - When using passwords, a brute-force protection should be implemented.

Exercise Solution



```
hacker@kali:~
$ time ncrack -p 22 --user bob -P /usr/share/ncrack/default.pwd -f 10.0.2.9
```

Password Sniffing as Root

- **Attack**
  - An attacker with root access on the server can read the password when a user authenticates.
  - If this password is valid on another system (when the same password is configured or when LDAP is used), an attacker can use the password and use it for lateral movement.
  - Common scenarios
    - Server owner who is admin on only one system.
    - External partner who has admin access on only some systems.
- **Recommendation**
  - Enforce public key authentication or 2FA (when the 2FA secret is different on every server).

Password Sniffing as Root

- Attacker

```
root@tux:~$ sudo strace -p "$(pgrep -f /usr/sbin/sshd)" -f -e trace=write
strace: Process 19531 attached
strace: Process 19594 attached
[...]
[pid 19595] write(5, "\0\0\0\4alice", 8) = 8
[...]
[pid 19595] write(5, "\0\0\0\10P@ssw0rd", 12) = 12
[...]
^C
root@tux:~$ ssh alice@puffy
alice@puffy’s password: ********
Welcome to puffy.
```

- User

```
alice@beastie:~$ ssh alice@tux
alice@tux’s password: ********
```

Session Sniffing as Root

- The `root` user can by design do everything on a system.
- Tools like `sshspy` can show the terminal of logged in users in real-time
  - It's a small `bash` script which uses `strace` to get all the information.
  - https://github.com/InfosecMatter/Scripts/blob/master/sshspy.sh

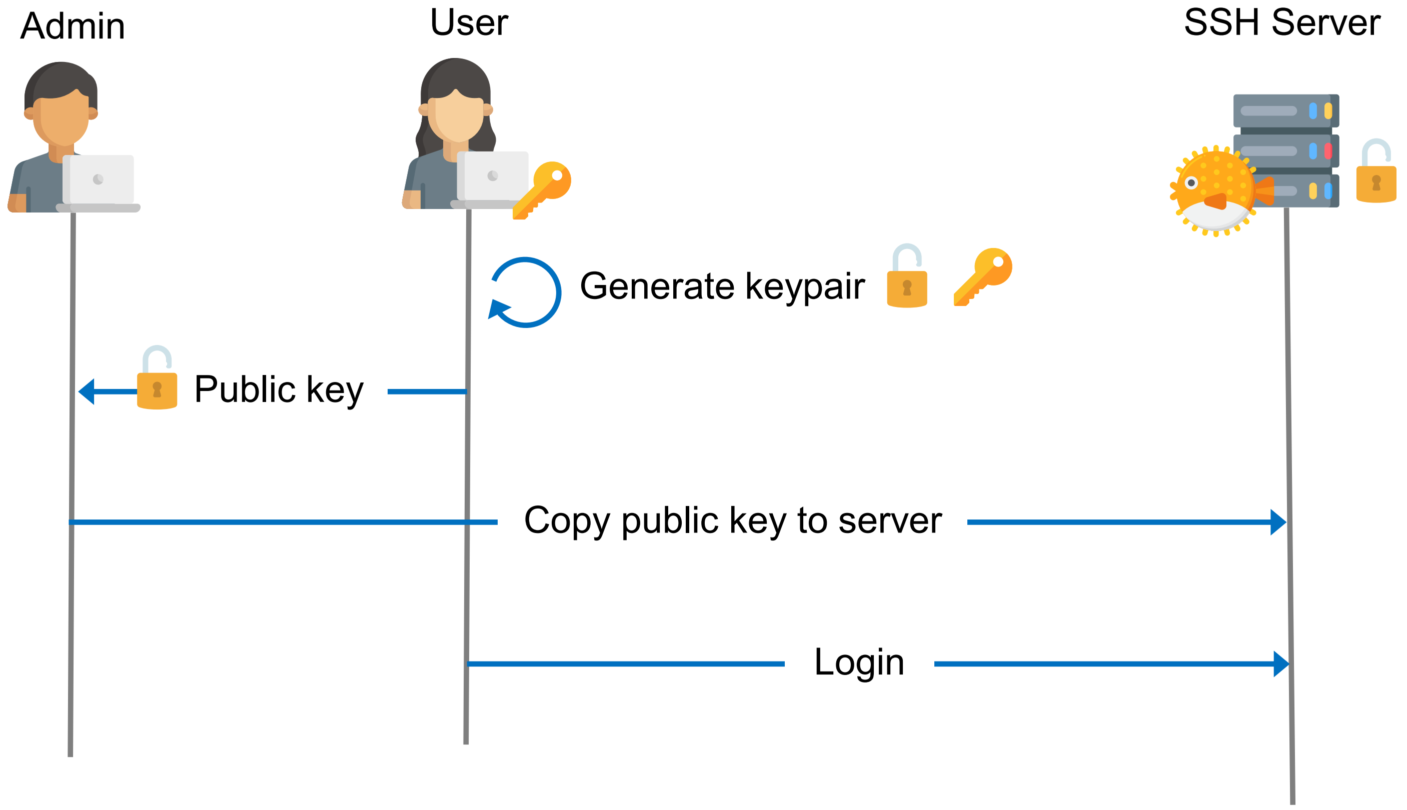
- **Attack**
  - An attacker with root access on the server can see/read everything other users do on the system.
  - This also includes typed passwords.
- **Recommendation**
  - Keep in mind who is `root` on the system and act accordingly.
  - Don't store data / process information / type passwords on untrusted systems.

SSHSpy Demo

```
root@server:~
# 
```

Public Key Authentication

- Passwordless authentication
- User generates keypair
- Private key on client
- Public key on target server
- Login using key



Public Key Authentication

- Key pairs are stored in users's `~/.ssh` directory.
- Different algorithms are available (DSA, ECDSA, Ed25519, RSA).
- Private keys can be encrypted using a passphrase.
- Private keys can be stored on secure devices
  - Smart Cards (PKCS11)
  - Hardware Keys / FIDO2 keys (e.g. Yubikey, Nitrokey, …)
  - TPM
- Public keys are stored in the authorized keys file on the target server
- By default, two authorized keys files are used (`sshd_config`):

```
AuthorizedKeysFile
      Specifies the file that contains the public keys used for user authentication
      [...] The default is ".ssh/authorized_keys .ssh/authorized_keys2".
```

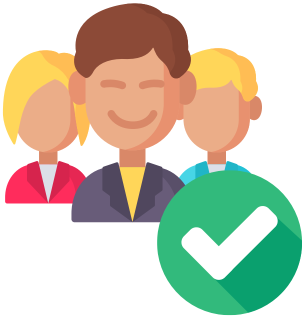
- Instead of distributing lots of keys, an SSH key CA could be used.

Public Key Authentication

- **Attacks**
  - An attacker who can perform privilege escalation on a host where private keys are stored can use them.
  - An attacker can try to offline brute-force the private key passphrase.
  - The authorized keys files can be used as a “backdoor”.
- **Recommendation**
  - Keys should not be stored on systems where other user’s have access to (e.g. jump hosts, source code repositories, scripts, …).
  - Keys should be protected using a strong passphrase or placed on a secure device.
  - The authorized keys files should be centrally managed and monitored.
  - Explicitly define the authorized keys file.
- Example server config (`sshd_config`)

```
AuthorizedKeysFile .ssh/authorized_keys
```

Allowed Users & Groups



- By default, all users are allowed to login if they have a login method configured.
  - Users with a password / SSH keys configured
- **Attacks**
  - RCE in a web application → change password via «`echo alice:P@ssw0rd | chpasswd`» → shell
  - Arbitrary file write in a web application → write one's SSH keys to `~/.ssh/authorized_keys` → shell
- **Recommendation**
  - Shared accounts should generally not be used → disable root login via SSH.
  - Only authorized users/groups should be able to establish an SSH connection.
  - Restrict SSH access to explicitly allowed users or groups.
- Example server config (`sshd_config`)

```
AllowUsers alice bob
AllowGroups sysadmins ssh-users
PermitRootLogin no # Implicit, but enforce even if root is in allowed group
```

Public Key Information Leakage

- Public keys are public (as the name says).
- Without the private key, you can’t use them to authenticate.
- Keys might be exposed where you don’t expect.
- E.g. all GitHub user keys are public:

```
$ curl https://github.com/emanuelduss.keys
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIHUpSBIZZ8EJy6hGGF0x9uypjJhLPuZNRFeYEIZtyKT4
```

- Also, you send your public key(s) to every server you try to authenticate.
  - By default: `id_rsa`, `id_ecdsa`, `id_ecdsa_sk`, `id_ed25519`, `id_ed25519_sk`, `id_dsa in ~/.ssh/` & all keys loaded into the SSH agent.
- The user’s public key is sent encrypted over the network after the host authentication.
  - A Machine-in-the-Middle attacker can’t read the public key.

Public Key Information Leakage

- If you login to arbitrary servers, you do expose your public key.
- PoC by Filippo Valsorda: https://words.filippo.io/ssh-whoami-filippo-io/
- It checks if your sent public key is on GitHub and shows your username:

```
$ ssh whoami.filippo.io
    +---------------------------------------------------------------------+
    |             _o/ Hello Emanuel Duss!
    |                                                                     |
    |  Did you know that ssh sends all your public keys to any server     |
    |  it tries to authenticate to?                                       |
    |                                                                     |
    |  We matched them to the keys of your GitHub account,                |
    |  @emanuelduss, which are available via the GraphQL API
    |  and at https://github.com/emanuelduss.keys
    |                                                                     |
    |  -- Filippo (https://filippo.io)                                    |
    |                                                                     |
    |  P.S. The source of this server is at                               |
    |  https://github.com/FiloSottile/whoami.filippo.io                   |
    +---------------------------------------------------------------------+
```

Public Key Information Leakage

- It’s possible to verify if a public key can be used to login or not, even without the private key:

```
alice@beastie:~$ ssh -v -i key.pub root@puffy
[...]
debug1: Offering public key: key.pub ED25519
SHA256:L619XZboqfh8ui85GqTBRPCpwkrxECR3WOoIagTWeno explicit
debug1: Authentications that can continue: publickey
debug1: No more authentication methods to try.
[...]

alice@beastie:~$ ssh -v -i key.pub alice@puffy
[...]
debug1: Offering public key: key.pub ED25519
SHA256:L619XZboqfh8ui85GqTBRPCpwkrxECR3WOoIagTWeno explicit
debug1: Server accepts key: key.pub ED25519
SHA256:L619XZboqfh8ui85GqTBRPCpwkrxECR3WOoIagTWeno explicit
[...]
```

Public Key Information Leakage

- This process can be automated:

```
$ sudo nmap -p 22 --script ssh-publickey-acceptance --script-args
'ssh.usernames={"root", "alice"}, publickeys={"./id_rsa1.pub", "./id_rsa2.pub"}' puffy

Nmap scan report for puffy (10.5.23.42)
22/tcp open  ssh     syn-ack
| ssh-publickey-acceptance:
|   Accepted Public Keys:
|_    Key ./id_rsa1 accepted for user alice
```

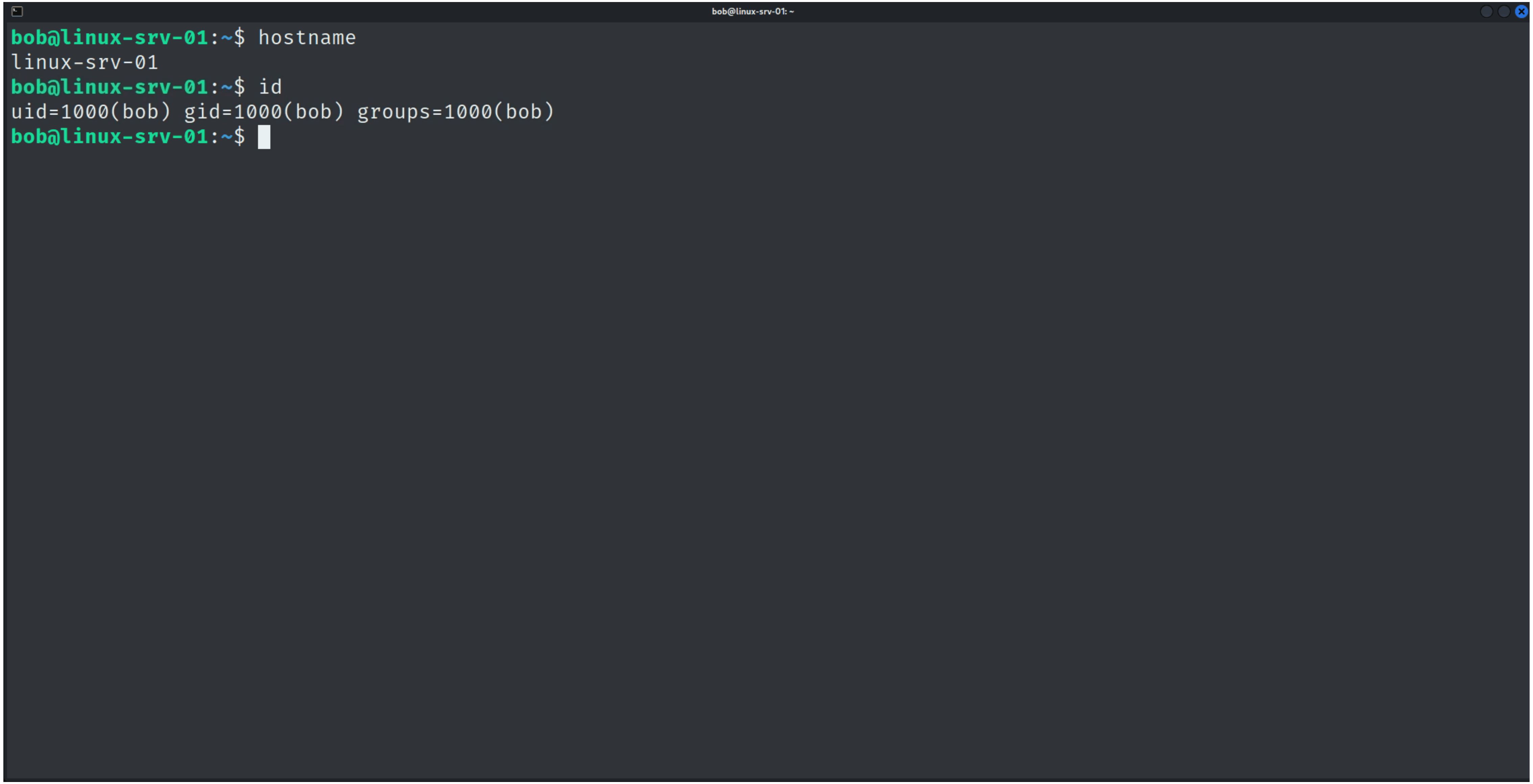
- Use Case
  - You find 50 passphrase encrypted SSH keys during an internal pentest
  - The pentest ends tomorrow and you only want to crack keys which are useful for you.
  - Which one do you want to crack?

Public Key Information Leakage

- **Attacks**
  - An attacker can get access to your public key when you login on an attacker-controlled system.
- **Recommendation**
  - Use different keys for different services (e.g. internal systems, external systems, external partners, 3rd party services, …) and always only use one to connect.
- This can be done via the following SSH config (`~/.ssh/config`):

```
Host github.com
  IdentityFile .ssh/id_ed25519_github
Host *.example.net
  IdentityFile .ssh/id_ed25519_internal
Host *
  IdentityFile .ssh/id_ed25519_external
```

Exercise Solution



```
bob@linux-srv-01:~$ hostname
linux-srv-01
bob@linux-srv-01:~$ id
uid=1000(bob) gid=1000(bob) groups=1000(bob)
bob@linux-srv-01:~$
```

Private Key Information Leakage

- You can specify a comment during key generation (default is `username@hostname`):

```
$ ssh-keygen -t ed25519 -C mykey
Generating public/private ed25519 key pair.
[...]
```

- The comment is stored inside the public key file:

```
$ cat .ssh/id_ed25519.pub
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIBPhRbyUNQirYCo6GrODJ++Jl/MUtTIPW1dBafg8vLu+ mykey
```

- The comment is also stored inside the private key as well and can be shown:

```
$ rm .ssh/id_ed25519.pub
$ ssh-keygen -y -f .ssh/id_ed25519
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIBPhRbyUNQirYCo6GrODJ++Jl/MUtTIPW1dBafg8vLu+ mykey
```

- This is usually no problem, since private keys should be kept private.
- But keep this in mind when you generate private keys which someone else could read (e.g. for trainings, CTF challenges, OPSEC during red team engagements, …)

2FA Authentication: OTP

- One example of 2FA authentication using public keys + authenticator app (OTP)
- SSH server configuration (`/etc/ssh/sshd_config`)

```
AuthenticationMethods publickey,keyboard-interactive
PasswordAuthentication no
UsePAM yes
ChallengeResponseAuthentication yes
```

- Install `libpam-google-authenticator`
- For every, user, generate OTP configuration (creates `~/.google_authenticator`):

```
alice@puffy:~$ google-authenticator
```

- PAM config for SSH (`/etc/pam.d/ssh`)

```
# Standard Un*x authentication.
#@include common-auth
auth required pam_google_authenticator.so nullok
```

2FA Authentication: OTP

- Example session

```
alice@beastie:~$ ssh -v puffy
[...]
debug1: Authentications that can continue: publickey
debug1: Next authentication method: publickey
debug1: Offering public key: /home/carol/.ssh/id_ed25519 ED25519
SHA256:/qM8Kw1JwTx/ijOG6k1Z2ILe/l2/K0lyAr0/zUGLqW8
debug1: Server accepts key: /home/carol/.ssh/id_ed25519 ED25519
SHA256:/qM8Kw1JwTx/ijOG6k1Z2ILe/l2/K0lyAr0/zUGLqW8
Authenticated with partial success.
debug1: Authentications that can continue: keyboard-interactive
debug1: Next authentication method: keyboard-interactive
Verification code: 500230
Welcome to puffy.
alice@puffy:~$
```

2FA Authentication: OTP

- **Attack**
  - An attacker with root access on the server can
    - read the password when a user authenticates and
    - Extract the secret of the OTP file
  - If the same password and OTP seed is used on another system, an attacker can use this information for lateral movement.
- **Recommendation**
  - Use public key authentication instead of passwords.
  - Use a different OTP on every system.
  - Use another 2FA method which is not vulnerable (like FIDO2)

2FA Authentication: FIDO2

- FIDO2 is an open authentication standard
- FIDO authenticator contains cryptographic key pairs inside hardware
  - e.g. Yubikey, Nitrokey
- Native support in newer OpenSSH versions (≥ 8.2p1)
- Key types: `ecdsa-sk` or `ed25519-sk` (these can also be passphrase protected)
- A FIDO PIN or biometrics must be set on the FIDO key to generate keys
- Discoverable / resident keys
  - Private key is stored on FIDO key (private key protected by the FIDO key can be copied from the key)
  - Only FIDO key is required to login
- Non-discoverable / non-resident
  - Private key stored in `~/.ssh`, protected using FIDO key
  - FIDO key & generated key file is required to login

2FA Authentication: FIDO2 with Resident Key

- Generate resident key:

```
alice@beastie:~$ ssh-keygen -t ed25519-sk -O resident -O application=ssh:alice-work
Generating public/private ed25519-sk key pair.
You may need to touch your authenticator to authorize key generation.
Enter PIN for authenticator:
You may need to touch your authenticator again to authorize key generation.
Enter file in which to save the key (/home/alice/.ssh/id_ed25519_sk):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/alice/.ssh/id_ed25519_sk
Your public key has been saved in /home/alice/.ssh/id_ed25519_sk.pub
[...]
```

- New key entry on authenticator:

```
alice@beastie:~$ ykman fido credentials list
Enter your PIN:
ssh:alice-work 0000000000000000000000000000000000000000000000000000000000000000 openssh
```

2FA Authentication: FIDO2 with Resident Key

- Private key is stored in `~/.ssh`:

```
alice@beastie:~$ ls -l .ssh/id_ed25519_sk*
-rw------- 1 alice alice 529 Mar 28 14:29 .ssh/id_ed25519_sk
-rw------- 1 alice alice 157 Mar 28 14:29 .ssh/id_ed25519_sk.pub
```

- Private key can only be accessed with key material on the FIDO key (tied to FIDO key).

- Login using the passphrase protected private key and the FIDO key:

```
alice@beastie:~$ ssh puffy
Enter passphrase for key '.ssh/id_ed25519_sk’:
Confirm user presence for key ED25519-SK
SHA256:uQwkqxPZO1bTj3ARWxa/6EL6PdQemQQ2X4pViE/je1w
User presence confirmed
Welcome to puffy.
alice@puffy:~$
```

2FA Authentication: FIDO2 with Resident Key

- The private key can be downloaded to another client (FIDO PIN is required):

```
alice@dragonfly:~$ ssh-keygen -K
Enter PIN for authenticator:
You may need to touch your authenticator to authorize key download.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Saved ED25519-SK key ssh:alice-work to id_ed25519_sk_rk_alice-work
```

- A new passphrase for the key can be defined.

- Instead of copying the key, the key can also be loaded into the SSH agent:

```
alice@dragonfly:~$ ssh-add -K
Enter PIN for authenticator:
Resident identity added: ED25519-SK SHA256:kl63KizwgVqe4RtCPhiMqnExygduOTMQdqLJRJfXKZg
```

2FA Authentication: FIDO2 with Resident Key

- This key can then again be used to login:

```
alice@beastie:~$ ssh puffy
Enter passphrase for key '.ssh/id_ed25519_sk’:
Confirm user presence for key ED25519-SK
SHA256:uQwkqxPZO1bTj3ARWxa/6EL6PdQemQQ2X4pViE/je1w
User presence confirmed
Welcome to puffy.
alice@puffy:~$
```

2FA Authentication: FIDO2 with Resident Key

- **Attacks**
  - An attacker with
    - a) knowledge of the FIDO key PIN
    - b) physical access to the FIDO key
  - can copy the private key to an own machine and use it to authenticate.
- **Recommendation**
  - For higher security, resident keys should not be used.
  - Instead, non-resident keys should be used.
- Generate non-resident keys:

```
alice@beastie:~$ ssh-keygen -t ed25519-sk -O application=ssh:alice-work
```

- The generated private keys are protected via the FIDO key.
- They are not stored on the FIDO key itself and must be copied manually to other systems
  - `~/.ssh/id_ed25519_sk` and `~/.ssh/id_ed25519_sk.pub`

2FA Authentication: FIDO2 Without Touch

- By default, FIDO keys require user presence (key touch) for every key access.
- A user can generate a key which does not require user presence:

```
alice@beastie:~$ ssh-keygen -t ed25519-sk -O resident -O no-touch-required -O
application=ssh:alice-work
```

- By default, SSH servers require user presence.
- A user can overwrite this in their personal authorized keys file:

```
alice@puffy:~$ cat .ssh/authorized_keys
no-touch-required sk-ssh-ed25519@openssh.com
AAAAGnNrLXNzaC1lZDI1NTE5QG9wZW5zc2guY29tAAAAIEvUpHBQeQCE4OuuTnTijntxMFdknEzPD06tKkfa88M
nAAAADnNzaDphbGljZS13b3Jr alice@beastie
```

- The user can then login without touch (SSH key passphrase is required if set):

```
alice@beastie:~$ ssh puffy
Enter passphrase for key '.ssh/id_ed25519_sk’:
Welcome to puffy.
alice@puffy:~$
```

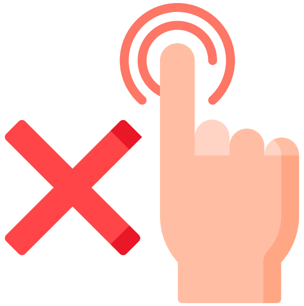
2FA Authentication: FIDO2 Without Touch

- **Attacks**
  - An attacker with
    - a) access to the FIDO key protected private key (no passphrase set, weak passphrase set, loaded into SSH agent)
    - b) code execution on the client
    - c) FIDO key plugged in
  - can establish an SSH connections without the user noticing.
- **Recommendation**
  - The server should enforce user presence.
- The server can enforce user presence (`sshd_config`):

```
PubkeyAuthOptions touch-required
```

- The user’s cant override this anymore in the authorized keys file.

2FA Authentication: FIDO2 Without User Authentication



- **Attacks**
  - An attacker with
    - a) access to the FIDO key protected private key (no passphrase set, weak passphrase set, loaded into SSH agent)
    - b) Physical access to the FIDO key
  - can establish an SSH connections using the private key and by touching the FIDO key.
- **Recommendation**
  - The server should enforce user authentication on every FIDO key access (PIN/biometrics).
- The server can enforce user authentication (`sshd_config`):

```
PubkeyAuthOptions verify-required
```

- The user’s cant override this anymore in the authorized keys file.

2FA Authentication: FIDO2 With User Authentication

- A user then has to generate keys which require authentication:

```
alice@beastie:~$ ssh-keygen -t ed25519-sk -O resident -O verify-required -O
application=ssh:alice-work
```

- The user then must enter the SSH key passphrase, the FIDO key PIN and touch the FIDO key:

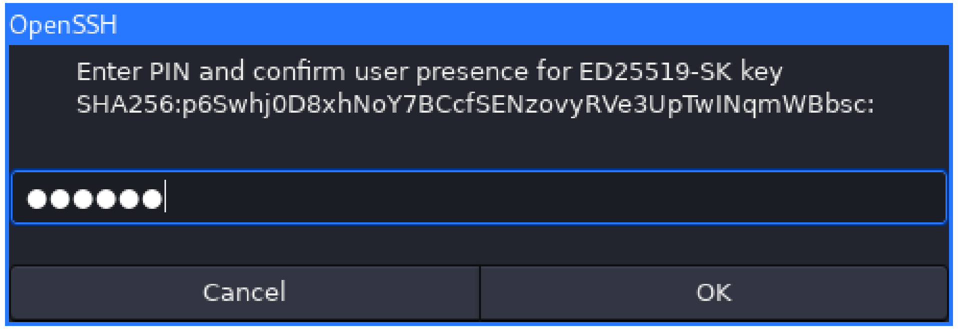
```
alice@beastie:~$ ssh puffy
Enter passphrase for key '/home/alice/.ssh/id_ed25519_sk':
Confirm user presence for key ED25519-SK
SHA256:11vUu+qDwFaIOZvBgODlzmcr5d60+7ljmzxJp/8KxAc
Enter PIN for ED25519-SK key /home/alice/.ssh/id_ed25519_sk:
Confirm user presence for key ED25519-SK
SHA256:11vUu+qDwFaIOZvBgODlzmcr5d60+7ljmzxJp/8KxAc
User presence confirmed
Welcome to puffy.
alice@puffy:~$
```

2FA Authentication: FIDO2 With User Authentication

- Example session on another client using resident keys & `ssh-agent` (requires `ssh-askpass`):

```
alice@beastie:~$ ssh -K
Enter PIN for authenticator:
Resident identity added: ED25519-SK SHA256:p6Swhj0D8xhNoY7BCcfSENzovyRVe3UpTwINqmWBbsc

alice@beastie:~$ ssh -v puffy
```

ssh-askpass



OpenSSH

Enter PIN and confirm user presence for ED25519-SK key SHA256:p6Swhj0D8xhNoY7BCcfSENzovyRVe3UpTwINqmWBbsc:

●●●●●●

+

```
Welcome to puffy.
alice@puffy:~$
```

No passphrase required, since private key file is not copied to machine.

SSH Agent

SSH Agent

- With passphrase protected keys, the key must be unlocked for each connection.
- To address this, keys can be loaded once into a so-called SSH agent.
  - Loading the key requires the passphrase.
- SSH agent is a process running in the background on the user’s client.
- Holds private keys used for public key authentication.
- The key can then be used without entering the passphrase again

SSH Agent

- Uses environment variables to connect to the agent socket.
- Example for Linux

```
alice@beastie:~$ eval $(ssh-agent)
Agent pid 1318

alice@beastie:~$ env | grep ^SSH
SSH_AUTH_SOCK=/tmp/ssh-PmBPRK9DcVkb/agent.2305
SSH_AGENT_PID=1318

alice@beastie:~$ ls -la /tmp/ssh-TUKDBryLDJUV/agent.1347
srw------- 1 alice alice 0 Feb 21 13:14 /tmp/ssh-TUKDBryLDJUV/agent.2305
```

SSH Agent

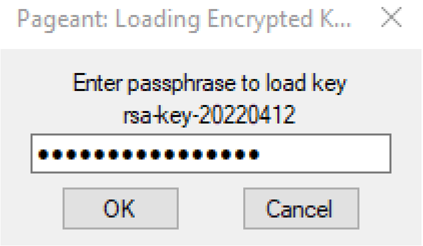
```
alice@beastie:~$ ssh puffy
Enter passphrase for key '/home/alice/.ssh/id_ed25519':
^C

alice@beastie:~$ ssh-add
Enter passphrase for /home/alice/.ssh/id_ed25519:
Identity added: /home/alice/.ssh/id_ed25519 (alice@beastie)

alice@beastie:~$ ssh-add -l
256 SHA256:4CbWpsIxO1X+xvhHAZwVyPU50dRyV8i0skV2S09G21g alice@beastie (ED25519)

alice@beastie:~$ ssh puffy
Welcome to puffy.
alice@puffy:~$
```

SSH Agent

- Example for Windows

```
PS > Get-Service ssh-agent | Set-Service -StartupType Automatic
PS > Get-Service ssh-agent
[...]
Running  ssh-agent          OpenSSH Authentication Agent

PS > ssh-add ~\.ssh\id_ed25519
256 SHA256:4CbWpsIxO1X+xvhHAZwVyPU50dRyV8i0skV2S09G21g alice@beastie (ED25519)

PS > ssh puffy
Welcome to puffy
alice@puffy:~$
```

- PuTTY also has an SSH Agent (pageant)

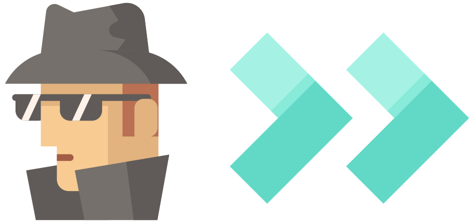


Pageant: Loading Encrypted K...

Enter passphrase to load key
rsa-key-20220412

●●●●●●●●●●●●●●●●●

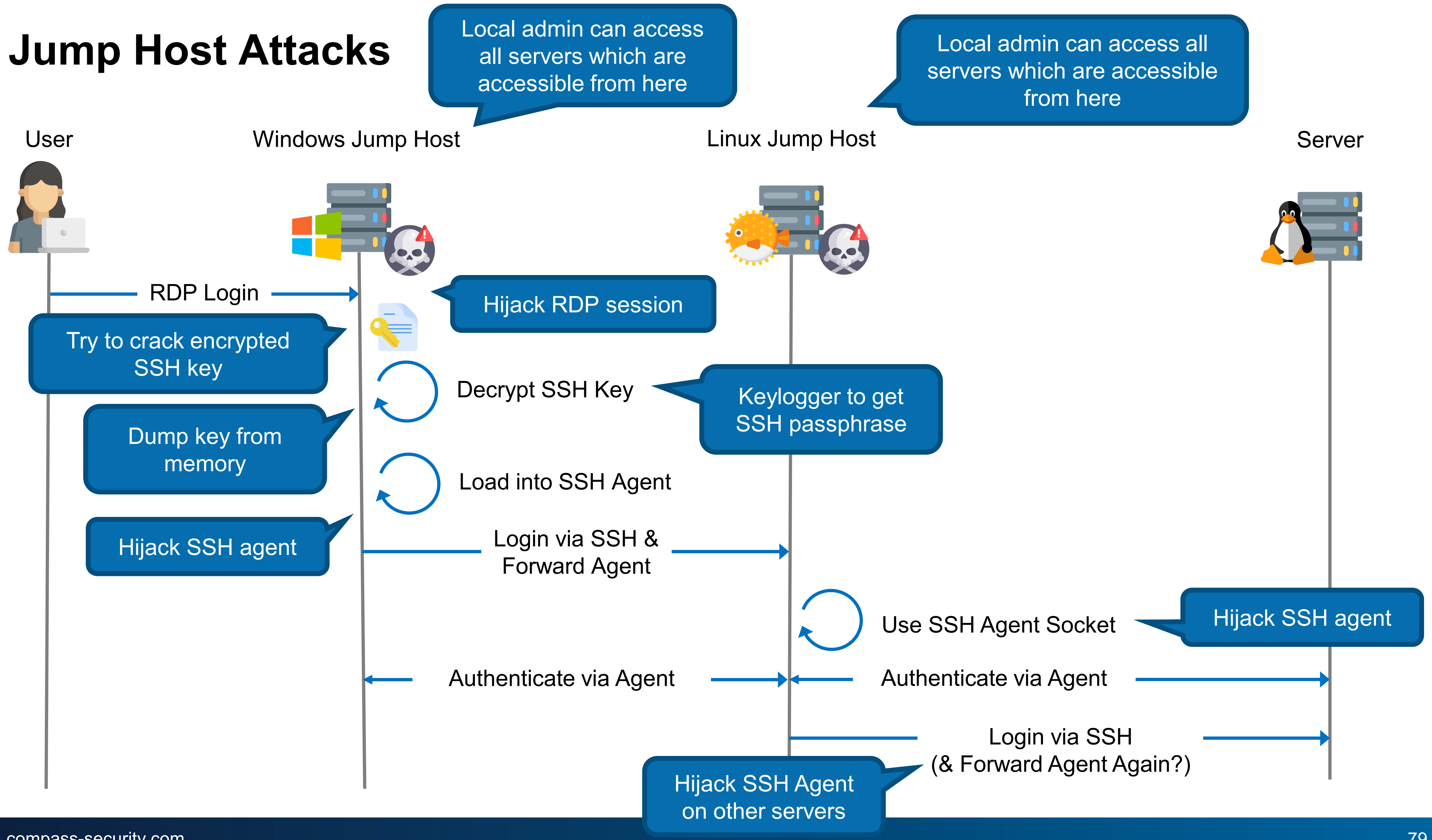
SSH Agent Forwarding



- SSH agent can be forwarded to a remote server
- This makes the loaded keys available on the remote server.

```
alice@beastie:~$ ssh -A jumphost
Welcome to jumphost.

alice@jumphost:~$ echo $SSH_AUTH_SOCK
/tmp/ssh-10bpIHPZjF/agent.1365

alice@jumphost:~$ ls -l $SSH_AUTH_SOCK
srwxr-xr-x 1 alice alice 0 Feb 21 13:22 /tmp/ssh-10bpIHPZjF/agent.1365

alice@jumphost:~$ ssh-add -l
256 SHA256:4CbWpsIxO1X+xvhHAZwVyPU50dRyV8i0skV2S09G21g alice@beastie (ED25519)

alice@jumphost:~$ ssh puffy
Welcome to puffy.
alice@puffy:~$
```

Jump Host Attacks

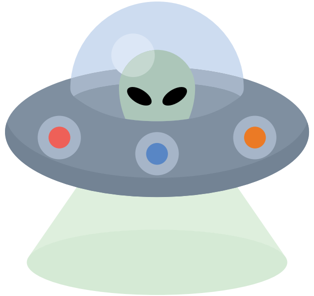


Local admin can access all servers which are accessible from here
Local admin can access all servers which are accessible from here
User
Windows Jump Host
Linux Jump Host
Server
RDP Login
Hijack RDP session
Try to crack encrypted SSH key
Decrypt SSH Key
Keylogger to get SSH passphrase
Dump key from memory
Load into SSH Agent
Hijack SSH agent
Login via SSH & Forward Agent
Use SSH Agent Socket
Hijack SSH agent
Authenticate via Agent
Authenticate via Agent
Login via SSH (& Forward Agent Again?)
Hijack SSH Agent on other servers

SSH Agent Hijacking

- **Attacks**
  - Whoever has access to the SSH agent socket, can use it to authenticate (but not obtain key material).
    - Low privileged users who can perform privilege escalation.
    - External partners with admin access to only one machine.
    - Sysadmins with only admin access on limited machines.

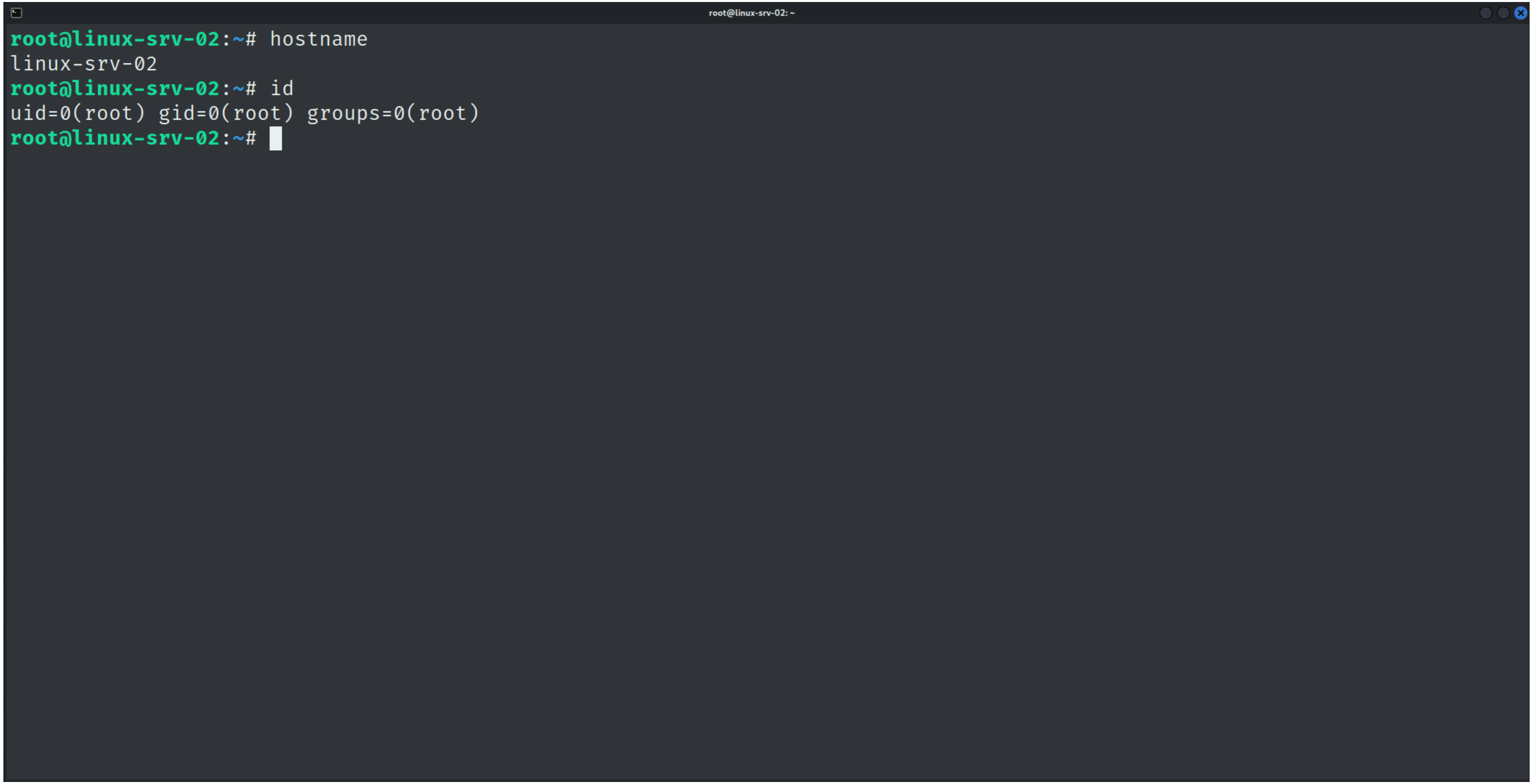
SSH Agent Hijacking



- Example

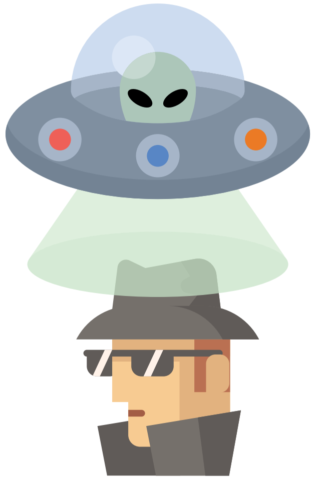
```
external-partner@aix:~$ ssh puffy
external-partner@puffy: Permission denied (publickey).
external-partner@aix:~$ ssh -l alice puffy
alice@puffy: Permission denied (publickey).

external-partner@aix:~$ sudo –i
root@aix:~# find / -type s -ls 2>/dev/null
/tmp/ssh-10bpIHPZjF/agent.1365

root@aix:~# export SSH_AUTH_SOCK=/tmp/ssh-10bpIHPZjF/agent.2305
root@aix:~# ssh-add -l
256 SHA256:4CbWpsIxO1X+xvhHAZwVyPU50dRyV8i0skV2S09G21g alice@beastie (ED25519)

root@aix:~# ssh -l alice puffy
Welcome to puffy.
alice@puffy:~$
```

Exercise Solution



```
root@linux-srv-02:~# hostname
linux-srv-02
root@linux-srv-02:~# id
uid=0(root) gid=0(root) groups=0(root)
root@linux-srv-02:~#
```

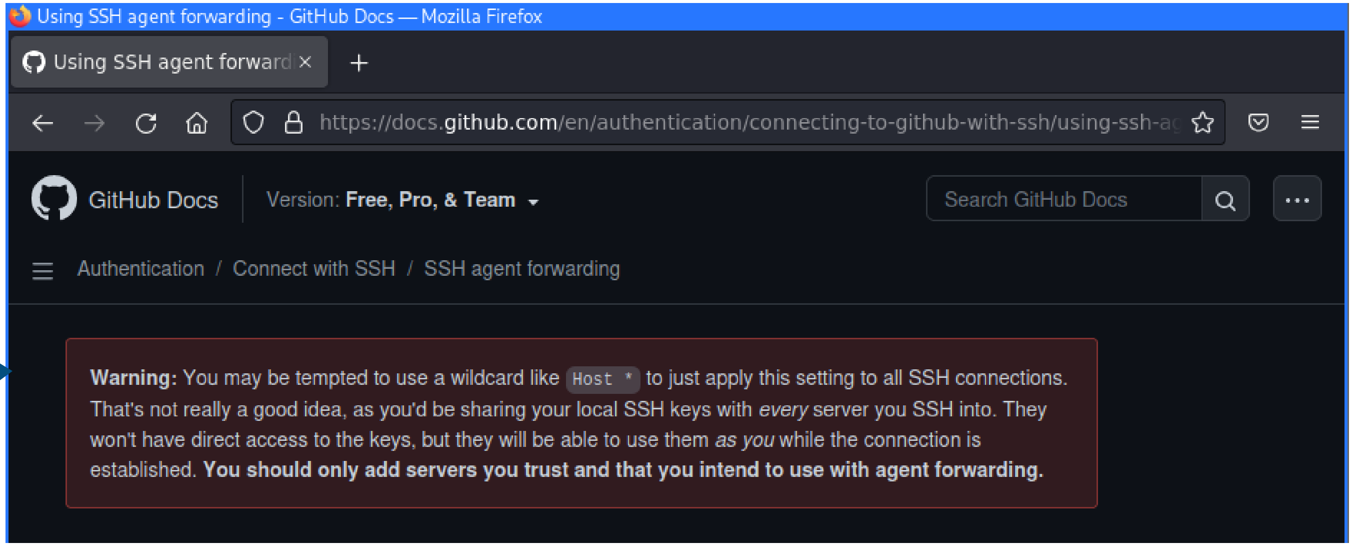
SSH Agent Hijacking



- **Recommendation**
  - Again: Don't store private keys on jump hosts.
  - Again: Encrypt private keys using a passphrase.
  - Don't use SSH agent forwarding
  - Explicitly deny SSH agent forwarding on the server
  - Use SSH jump proxy feature `ProxyJump`
    (Connect stdio on the client to a single port forward on the server.)
  - Don't allow interactive login on jump proxy
- Example server config

```
AllowAgentForwarding no
```

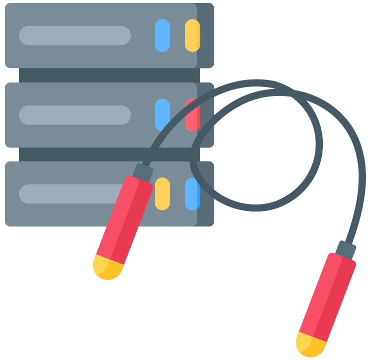
GitHub also warns from using this feature.



Using SSH agent forwarding - GitHub Docs — Mozilla Firefox
Using SSH agent forward ×
https://docs.github.com/en/authentication/connecting-to-github-with-ssh/using-ssh-a
GitHub Docs
Version: Free, Pro, & Team
Search GitHub Docs
Authentication / Connect with SSH / SSH agent forwarding

Warning: You may be tempted to use a wildcard like `Host *` to just apply this setting to all SSH connections. That's not really a good idea, as you'd be sharing your local SSH keys with *every* server you SSH into. They won't have direct access to the keys, but they will be able to use them *as you* while the connection is established. **You should only add servers you trust and that you intend to use with agent forwarding.**

SSH Jump Proxies



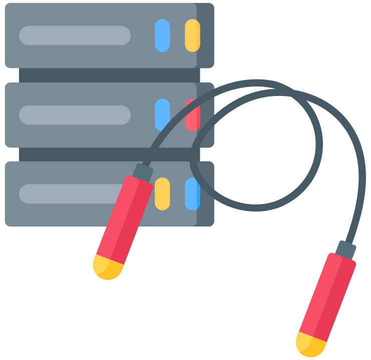
- Example session

```
alice@beastie:~$ ssh-add
Enter passphrase for /home/alice/.ssh/id_ed25519:
Identity added: /home/alice/.ssh/id_ed25519 (alice@beastie)

alice@beastie:~$ ssh-add -l
256 SHA256:4CbWpsIxO1X+xvhHAZwVyPU50dRyV8i0skV2S09G21g alice@beastie (ED25519)

alice@beastie:~$ ssh -J jumphost puffy
Welcome to puffy.
alice@puffy:~$
```

- It’s possible to use multiple jump hosts:

```
alice@beastie:~$ ssh -J jumper,bouncy puffy
Welcome to puffy.
alice@puffy:~$
```

SSH Jump Proxies



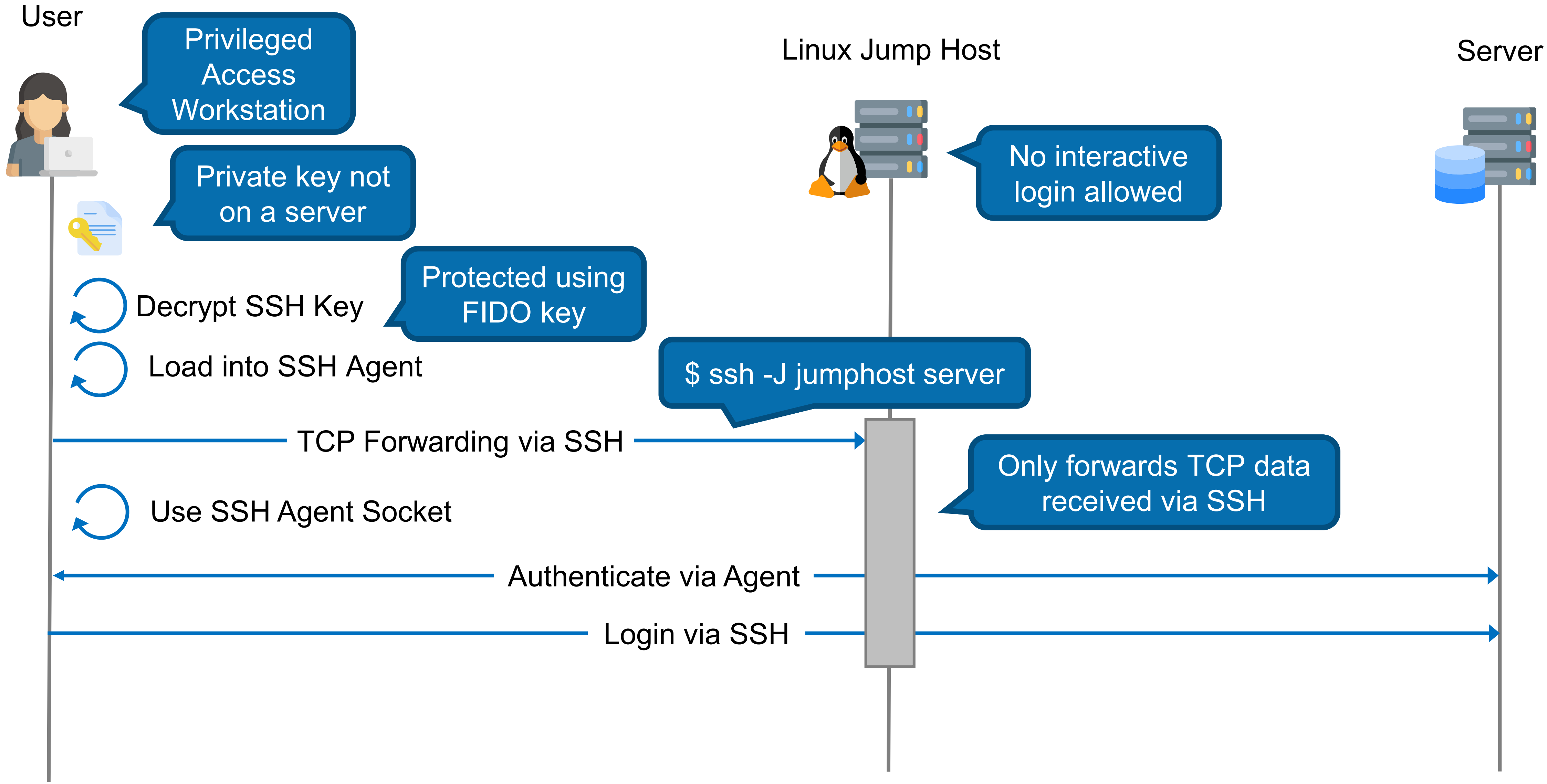
- Example client config

```
Host puffy linux-srv-?? aix-srv-??
   HostName %h.example.net # Add domain for internal systems

Host *.example.net !jumphost.example.net
    ProxyJump jumphost.example.net # connect via the JumpHost
```

- Example session

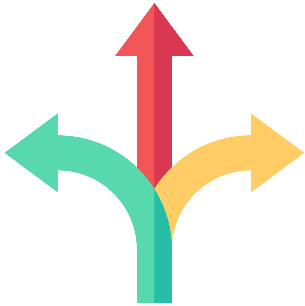
```
alice@beastie:~$ ssh puffy
Welcome to puffy.
alice@puffy:~$
```

Remediation



SSH Session Multiplexing

SSH Session Multiplexing

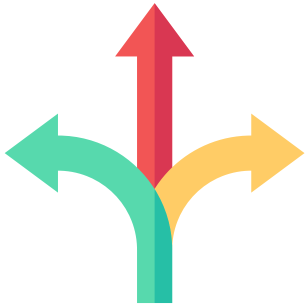


- It’s possible to reuse one TCP connection for multiple SSH sessions.
- Only establish one TCP connection and authenticate once on the server.
- Faster, because further SSH sessions will use the already established SSH session.
- Example Use Case: Speed up connections via jump proxy

```
Host jumphost.example.net
    ControlMaster auto
    ControlPath ~/.ssh/cm-%r-%h-%p
    ControlPersist 0
Host *.example.net !jumphost.example.net
    ProxyJump jumphost.example.net # connect via the JumpHost
```

- Example Use Case: Speed up running multiple Ansible Playbooks

SSH Session Multiplexing



- Establishing first session

```
alice@beastie:~$ time ssh puffy true
real    0m1.000s
```

- Control socket exists

```
alice@beastie:~$ ssh -O check puffy
Master running (pid=49960)
alice@beastie:~$ ls -l .ssh/cm-alice-puffy-22
srw------- 1 alice alice 0 Feb 23 14:41 .ssh/cm-alice-puffy-22
```

- Establishing second session

```
alice@beastie:~$ time ssh puffy true
real    0m0.080s
```

- Terminating control socket:

```
alice@beastie:~$ ssh -O stop puffy
Stop listening request sent.
alice@beastie:~$ ssh -O check puffy
Control socket connect(/home/alice/.ssh/cm-alice-puffy-22): No such file or directory
```

SSH Session Multiplexing

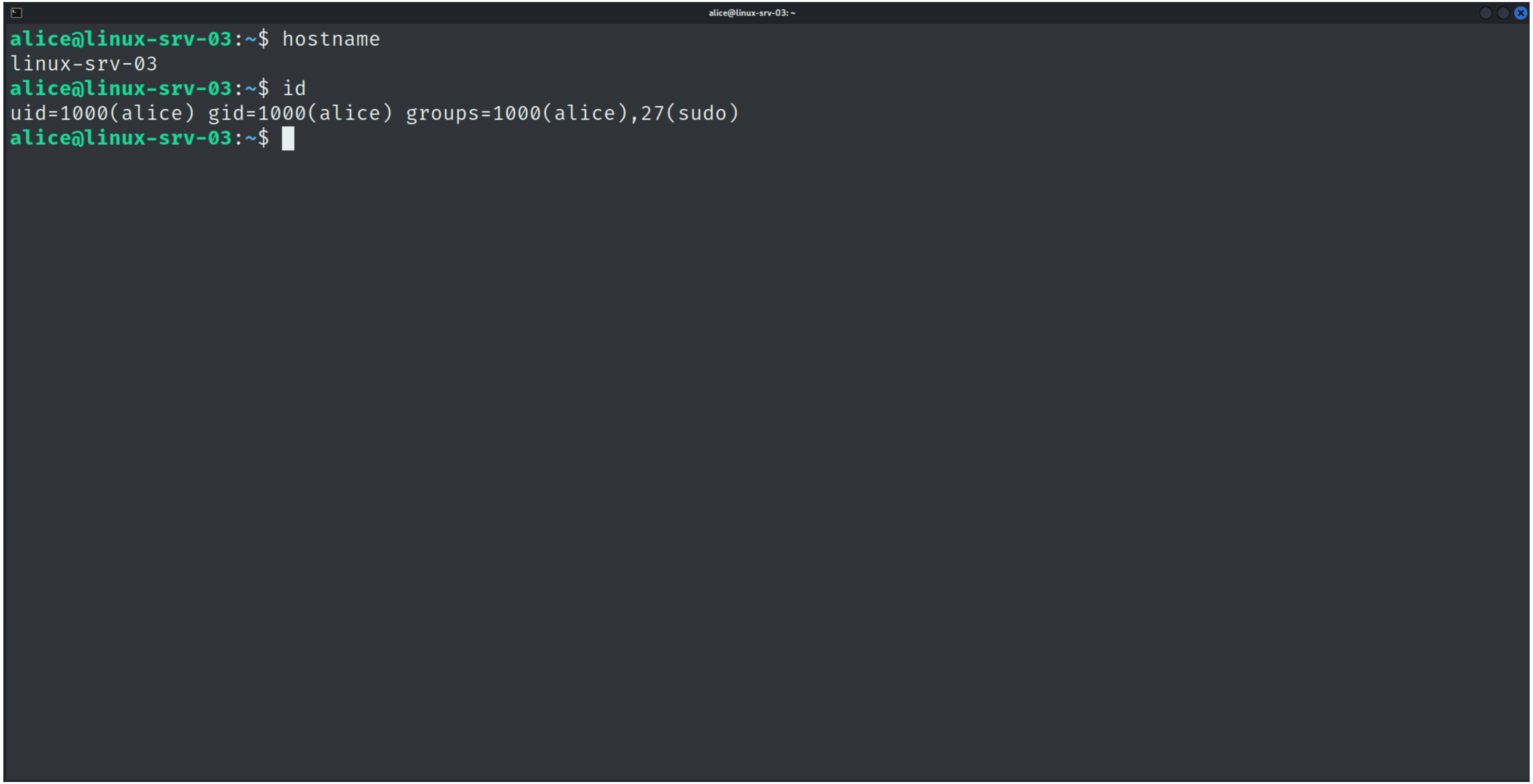
- **Attacks**
  - Whoever has access to the SSH control socket, can use it to reuse the connection and establish a new SSH session
    - Low privileged users who can perform privilege escalation.
    - External partners with admin access to only one machine.
    - Sysadmins with only admin access on limited machines.
  - Since the connection is already authenticated, no further authentication is required
    - No need for passwords, private keys, passphrase for keys
    - Even bypasses 2FA

SSH Session Multiplexing

- Example

```
external-partner@aix:~$ ssh puffy
external-partner@puffy: Permission denied (publickey).
external-partner@aix:~$ ssh -l alice puffy
alice@puffy: Permission denied (publickey).

external-partner@aix:~$ sudo –i
root@aix:~# find / -type s -ls 2>/dev/null
/home/alice/.ssh/cm-alice-puffy-22

root@aix:~# ssh -l alice -S /home/alice/.ssh/cm-alice-puffy-22 puffy
alice@puffy:~$
```

Exercise Solution



```
alice@linux-srv-03:~$ hostname
linux-srv-03
alice@linux-srv-03:~$ id
uid=1000(alice) gid=1000(alice) groups=1000(alice),27(sudo)
alice@linux-srv-03:~$
```

SSH Session Multiplexing

- **Recommendation**
  - Generally, don’t use SSH control sockets.
  - Only allow one session per connection on the server to deny control sockets
- Example server config:

```
MaxSessions 1
```

Cryptographic Algorithms

Cryptographic Algorithms

- SSH supports various cryptographic algorithms
  - Host Key
  - Key Exchange
  - Encryption
  - Message Authentication
- Recent OpenSSH servers use sane default, but some ciphers are “more secure” than others.
- The Internet is full of recommendations / guides and tools.

Cryptographic Algorithms

- **Attacks**
  - If weak algorithms are used, attackers who can intercept your communication could decrypt or even manipulate it.
  - This is however not that easy as it sounds, especially for non-state/nation-level attackers.
- **Recommendation**
  - Audit your SSH config and only enable secure cryptographic algorithms.
  - SSH-Audit Hardening Guide: https://www.ssh-audit.com/hardening_guides.html
  - Tool to audit your SSH config: `ssh-audit`

Cryptographic Algorithms

```
$ ssh-audit linux-srv-01
# general
(gen) banner: SSH-2.0-OpenSSH_9.1p1 Debian-2
(gen) software: OpenSSH 9.1p1
(gen) compatibility: OpenSSH 8.5+, Dropbear SSH 2018.76+
(gen) compression: enabled (zlib@openssh.com)

# key exchange algorithms
(kex) sntrup761x25519-sha512@openssh.com  -- [warn] using experimental algorithm
                                          `- [info] available since OpenSSH 8.5
(kex) curve25519-sha256                   -- [info] available since OpenSSH 7.4, Dropbear SSH 2018.76
(kex) curve25519-sha256@libssh.org        -- [info] available since OpenSSH 6.5, Dropbear SSH 2013.62
(kex) ecdh-sha2-nistp256                  -- [fail] using weak elliptic curves
                                          `- [info] available since OpenSSH 5.7, Dropbear SSH 2013.62
(kex) ecdh-sha2-nistp384                  -- [fail] using weak elliptic curves
                                          `- [info] available since OpenSSH 5.7, Dropbear SSH 2013.62
(kex) ecdh-sha2-nistp521                  -- [fail] using weak elliptic curves
                                          `- [info] available since OpenSSH 5.7, Dropbear SSH 2013.62
(kex) diffie-hellman-group-exchange-sha256 (2048-bit) -- [info] available since OpenSSH 4.4
(kex) diffie-hellman-group16-sha512       -- [info] available since OpenSSH 7.3, Dropbear SSH 2016.73
(kex) diffie-hellman-group18-sha512       -- [info] available since OpenSSH 7.3
(kex) diffie-hellman-group14-sha256       -- [info] available since OpenSSH 7.3, Dropbear SSH 2016.73
```

Example for
default installation
on Debian.

Cryptographic Algorithms

```
# host-key algorithms
(key) rsa-sha2-512 (3072-bit)            -- [info] available since OpenSSH 7.2
(key) rsa-sha2-256 (3072-bit)            -- [info] available since OpenSSH 7.2
(key) ecdsa-sha2-nistp256                -- [fail] using weak elliptic curves
                                         `- [warn] using weak RNG could reveal the key
                                         `- [info] available since OpenSSH 5.7, Dropbear 2013.62
(key) ssh-ed25519                        -- [info] available since OpenSSH 6.5

# encryption algorithms (ciphers)
(enc) chacha20-poly1305@openssh.com -- [info] available since OpenSSH 6.5
                                         `- [info] default cipher since OpenSSH 6.9.
(enc) aes128-ctr                         -- [info] available since OpenSSH 3.7, Dropbear SSH 0.52
(enc) aes192-ctr                         -- [info] available since OpenSSH 3.7
(enc) aes256-ctr                         -- [info] available since OpenSSH 3.7, Dropbear SSH 0.52
(enc) aes128-gcm@openssh.com             -- [info] available since OpenSSH 6.2
(enc) aes256-gcm@openssh.com             -- [info] available since OpenSSH 6.2

# fingerprints
(fin) ssh-ed25519: SHA256:W3Ypt7WQZWeq9XueVDqTfJVzIaly/4KkYSwFzvlgecM
(fin) ssh-rsa: SHA256:CjyhXmy2WEHJu6Pr/O85XG6Kh41SL8pCyZgi/ZR3BoM
```

Cryptographic Algorithms

```
# message authentication code algorithms
(mac) umac-64-etm@openssh.com        -- [warn] using small 64-bit tag size
                                     `- [info] available since OpenSSH 6.2
(mac) umac-128-etm@openssh.com       -- [info] available since OpenSSH 6.2
(mac) hmac-sha2-256-etm@openssh.com  -- [info] available since OpenSSH 6.2
(mac) hmac-sha2-512-etm@openssh.com  -- [info] available since OpenSSH 6.2
(mac) hmac-sha1-etm@openssh.com      -- [warn] using weak hashing algorithm
                                     `- [info] available since OpenSSH 6.2
(mac) umac-64@openssh.com            -- [warn] using encrypt-and-MAC mode
                                     `- [warn] using small 64-bit tag size
                                     `- [info] available since OpenSSH 4.7
(mac) umac-128@openssh.com           -- [warn] using encrypt-and-MAC mode
                                     `- [info] available since OpenSSH 6.2
(mac) hmac-sha2-256                  -- [warn] using encrypt-and-MAC mode
                                     `- [info] available since OpenSSH 5.9, Dropbear SSH 2013.56
(mac) hmac-sha2-512                  -- [warn] using encrypt-and-MAC mode
                                     `- [info] available since OpenSSH 5.9, Dropbear SSH 2013.56
(mac) hmac-sha1                      -- [warn] using encrypt-and-MAC mode
                                     `- [warn] using weak hashing algorithm
                                     `- [info] available since OpenSSH 2.1.0, Dropbear SSH 0.28
```

Questions?

👨‍💼 emanuel.duss@compass-security.com

📧 me@emanuelduss.ch

🌍 https://emanuelduss.ch

🐘 @emanuelduss@infosec.exchange

🐦 ❌ @emanuelduss

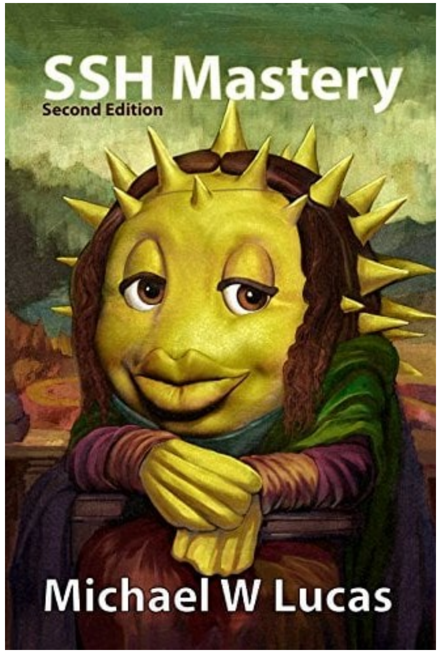
References

References

- Standards
  - SSH Protocol Architecture, RFC 4251, 2006: https://datatracker.ietf.org/doc/html/rfc4251
  - Secure Shell (SSH) Protocol Parameters, IANA, 2005: https://www.iana.org/assignments/ssh-parameters/ssh-parameters.xhtml
  - Using DNS to Securely Publish Secure Shell (SSH) Key Fingerprints, RFC 4255, 2006: https://datatracker.ietf.org/doc/html/rfc4255
- Manpages
  - `sshd_config(5)`: `DebianBanner`, `Banner`, `VerifyHostKeyDNS`, `AuthenticationMethods`, `AuthorizedKeysFile`, `PermitRootLogin`, `AllowUsers`, `AllowGroups`
  - `ssh_config(5)`: `GlobalKnownHostsFile`, `UserKnownHostsFile`, `HashKnownHosts`
  - `ssh(1)`: `AUTHENTICATION`
  - `ssh-keygen(1)`
  - `ssh-agent(1)`
  - `ssh-add(1)`

References

- General
  - SSH Mastery. 2nd Edition. Michael W Lucas. 2018.
- Session spying
  - https://www.infosecmatter.com/ssh-sniffing-ssh-spying-methods-and-defense/
- FIDO Keys
  - https://developers.yubico.com/SSH/Securing_SSH_with_FIDO2.html
- Jump Proxy
  - https://www.redhat.com/sysadmin/ssh-proxy-bastion-proxyjump
  - https://wiki.gentoo.org/wiki/SSH_jump_host
  - https://www.cyberciti.biz/faq/create-ssh-config-file-on-linux-unix/



COMPASS
SECURITY