



ELIMINATE DANGEROUS "MASTER KEYS" BY BUILDING AN SSH KEY MANAGEMENT SYSTEM



www.splashtop.com/foxpass



WHAT ARE MASTER KEYS?

In many organizations, engineers are granted unrestricted access to every server and all of the data each server contains. This is often called "administrator" or "root" level access. Such unfettered access is considered a "master key", just like a "master key" to an office building can open every door.

And just like losing the "master key" to an office building can have disastrous consequences, giving engineers "master keys" to servers and data can be more catastrophic when one is lost, stolen, or abused.

WHY ARE MASTER KEYS A MISTAKE?

CASE 1: MALICIOUS INSIDER – UNLIKELY BUT LETHAL

A malicious insider is defined as an employee who has the necessary means to gain access to a company's assets, and the motive to cause harm. This case is highly unlikely since only employees who have gained a certain amount of trust inside the organization are given root access, but this scenario cannot be ruled out as an impossibility.

Take, for instance, the malicious insider who stole gigabytes of data from Tesla and transferred it to third parties. According to a study, more than 49% of companies are worried about malicious attackers as this is a real and growing concern.

CASE 2: NEGLIGENT INSIDER – LIKELY AND UNTRACKABLE

Despite all the training given to IT professionals, most companies are still vulnerable and likely to be a victim of phishing and other forms of social engineering attacks. In fact, over 34% of all breaches in 2018 were caused by insiders according to a study. Moreover, these kinds of attacks are usually untrackable, and a company's infrastructure may have already been penetrated before anyone knows what has happened.

CASE 3: STOLEN CREDENTIALS – LIKELY AND TRACKABLE

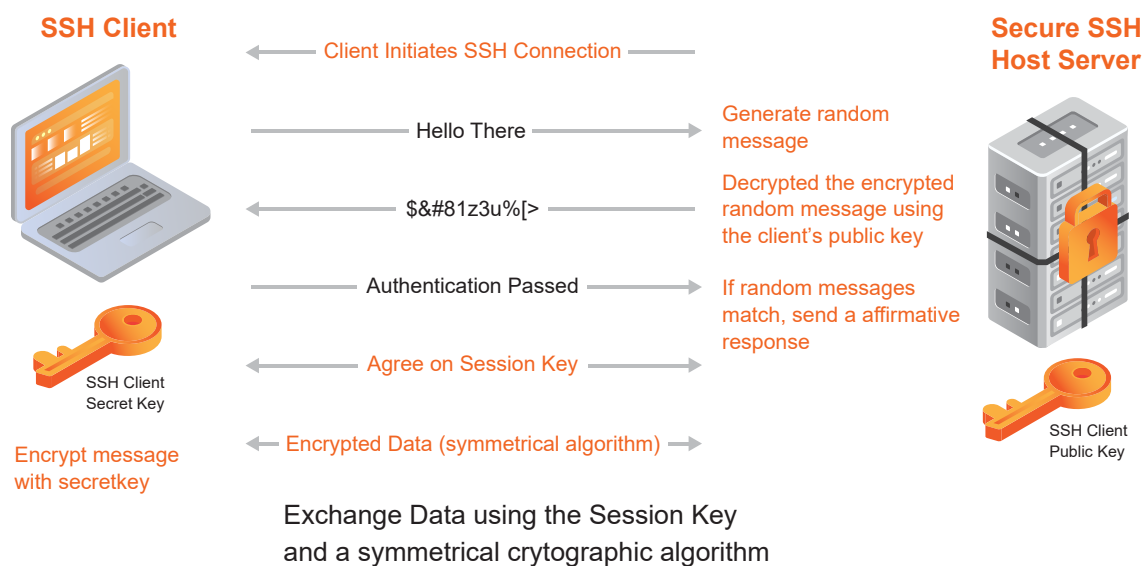
Stolen laptops or stolen credentials are also likely outsider attacks that target employees with the highest level of access rights, i.e. the engineers and the internal team. Once an outsider obtains these pilfered credentials, they can easily access the company's infrastructure. If these credentials were shared amongst several employees, it must be changed immediately, which locks out all other employees from accessing the servers until they can be securely informed of the new credentials. If each person had their own credentials for the server, only the affected person's credentials would have to be reset.

HOW CAN THE POSSIBILITY OF DATA BREACHES BE REDUCED?

Tech-savvy companies know the hazards of insider attacks, and restrict their engineer's access by roles. Role-Based Access Control (RBAC) limits user access based on their roles within an organization.

Deploying RBAC means that a user only has access to what they frequently need in order to do their job. However, "frequently" does not necessarily have to be on a daily, weekly, or even monthly basis. The most secure option is to give engineers the lowest level of access possible. When higher-level access is required, it should be monitored, controlled, and timed.

How can a well-monitored, controlled, and timed access control mechanism be achieved? The best solution is a system by which each engineer has his/her own key, and one that allows access to a server for the pre-set duration of time — the shorter the better.



WHEN SHOULD PERMISSIONS EXPIRE?

Typically, an engineer is granted permissions from the time they start at a company until they leave. However, a more dynamic option would be to grant permissions:

- Only during an on-call rotation
- Until a Jira type ticket is closed
- Only if the permission was used at least once in the last month

WHAT IS AN SSH KEY?

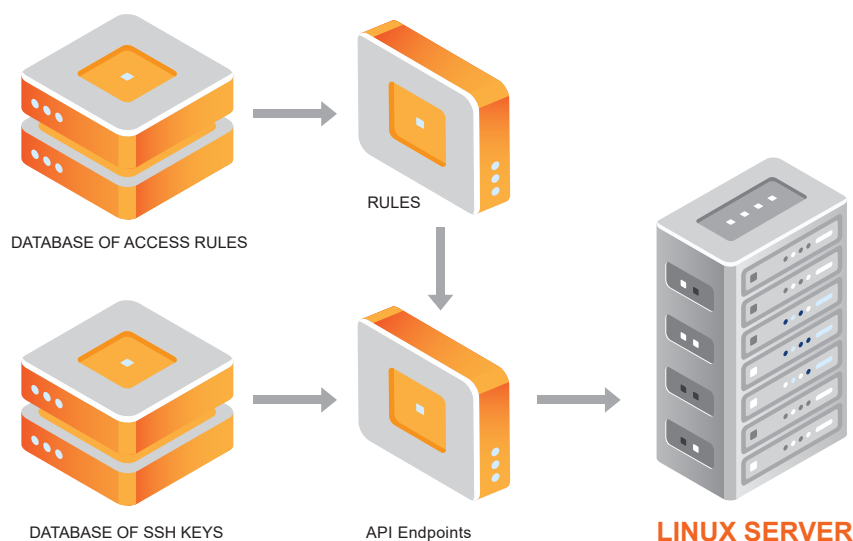
An "SSH Key" is a cryptographically secure mechanism to prove one's identity to a server when logging in over the network. It is similar in that respect to a long password, but that's where the similarities end. An "SSH key" is actually composed of two parts: a private key in one file and a public key in another. The contents of these two files are very different but they complement each other and therefore are created at the same time.

The private half of the SSH key should be guarded and never, for example, leave the system on which it is generated. The public half, however, can be freely shared. The private key cannot be regenerated with any knowledge of the public key. By copying the public portion of the key to any remote server where this user should have log-in permissions, the owner of the private half will be granted access.

HOW TO CREATE AN SSH KEY MANAGEMENT SYSTEM?

The components required for creating an SSH key management system are:

-  A database of SSH keys where the public portions of the users' SSH keys are stored
-  A database of access rules describing which users can access which servers
-  An application server that supports simple web applications and API endpoints



HOW TO SET THIS UP:

- 1 Traditionally, sshd daemon looks on-disk for a user's `authorized_keys` file in the user's `~/.ssh` directory.

However, in this setup, sshd directive called `AuthorizedKeysCommand` is leveraged (available in OpenSSH starting with version 6.2) to instruct sshd to look to the results of the script located in `/usr/local/bin/get_keys.sh`. To do so, change the location inside the `sshd_config` file. Note that if this fails, sshd still falls back to the on-disk `authorized_keys` file. To do this, edit `sshd_config` file to:

```
ubuntu@host:~$ vi /etc/ssh/sshd_config

[. . .]
AuthorizedKeysCommand /path/to/get_keys.sh
AuthorizedKeysCommandUser nobody
```

- 2 The script in `/usr/local/bin/get_keys.sh` can be written in any language, but Shell script or Python is recommended. The first and only argument for this script is the Linux username of the user requesting the keys. The script needs to re-issue the same query to the API endpoint that is built. Use the `curl` command to do so. Eg. `curl https://<endpoint>/get_keys/?user=$1`. An example of the `get_keys.sh` (AWS specific) is given below:

```
#!/bin/sh

user="$1"
hostname=`hostname`
instance_id=`curl -s -q -f
http://169.254.169.254/latest/meta-data/instance-id`
curl -s -q -f -m 5 "https://keyserver.example.com/sshkeys/?
user=${user}&hostname=${hostname}&instance_id=${instance_id}" 2>/dev/null

exit $?
```

Example request:

`https://keyserver.example.com/sshkeys/?user=aren&hostname=prod-db-1&instance_id=i-0b263919b6498b123`

- 3 Create the /sshkeys/ endpoint in the API such that it:
- Gets the username argument from the query string.
 - Looks up the ssh keys for that user in the database, and
 - Returns the ssh keys in the same format as an authorized_keys file: each SSH public key on its own line.

Example:

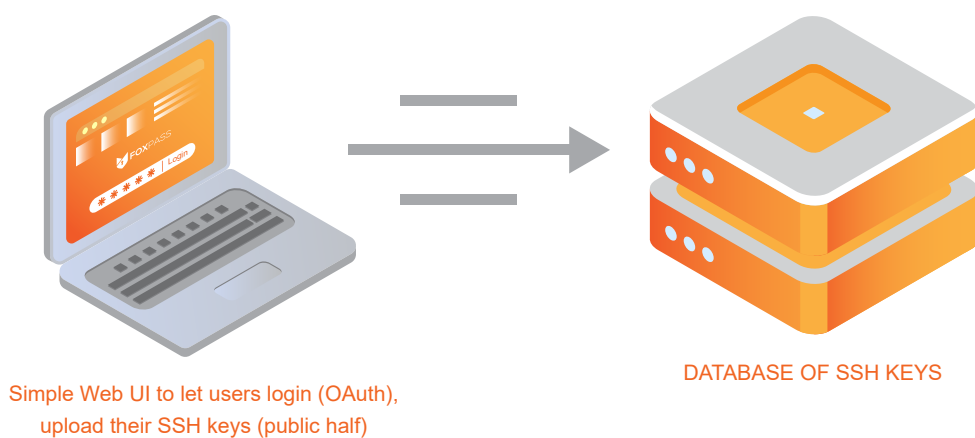
```
def sshkeys(request):
    hostname = request.GET['hostname']
    aws_instance_id = request.GET['aws_instance_id']
    username = request.GET['user']

    user = get_user(username)

    keys = []
    if is_user_allowed_on_host(user, hostname, aws_instance_id):
        keys = user.keys

    return HttpResponse("\n".join(key_text) for key in keys),
    content_type='text/plain')
```

- 4 Build a web application that users can use to manage SSH keys. It is easy to authenticate a user into the web application using OAuth from GSuite or Office 365. The user should be allowed to add a new key and deactivate an old key.



This basic setup can be extended by introducing a database of access rules that govern and control access to the servers. By integrating with external APIs, access can be modified based on conditions, and access can be reduced or removed accordingly.

- To make sure only current employees' SSH keys are distributed: cross-reference against GSuite, Office365, Okta, or OneLogin's API
- To enforce SSH key rotation: filter out SSH keys that are over 90/180/360 days old
- To restrict which servers a user may access: check the access rights defined for each user. If the permission for the particular server does not exist, then prevent the user from accessing the system

Example:

<https://keyserver.example.com/sshkeys/?user=aren&hostname=prod-db-1>

```
hostname=prod-db-1
```

```
"Production machine" match regex = ^prod-.*
```

```
Does the user "aren" have rights to any "Production machine" now?
```

or

```
hostname=prod-db-1
```

```
"Database machine" match regex = -db-\d+$
```

```
Does the user "aren" have rights to any "Database machine" now?
```

CONCLUSION

The possibilities of using this setup for SSH key management and access control are endless. Security vulnerabilities and insider attacks are always a big concern for organizations that have sensitive data that can be shared with engineers as they can have a monumental impact if leaked. It always makes sense to offer employees the least privileges possible, and to limit access to critical resources when access is given.

If Ashley Madison's site, Target, or Tesla had better access control, it is possible they would not have had such costly and public security breaches. SSH key management is a useful mechanism to secure infrastructure and help prevent internal teams from leaking insider knowledge to the outside world.

HOW FOXPASS CAN HELP

Foxpass offers an easy-to-use, full-featured SSH Key Management and privileged access control products that effectively allows engineers to access the required servers for a predefined amount of time. Additionally, Foxpass is a complete user and group management system for Linux that synchronizes with popular directories like GSuite, Office365, and Okta.