SSH Handler

Description

An admin user (SSH_Handler) run the script python and the script wait for connection from the employee. The employee connect to the server and import his public key. The script can now populate the authorized_keys files in every machines of the company he has access to, thanks to the data given by the admin user.

Server side

A thread is created to listen for incoming connections from the employee. When a connection is etablished, the server thread it and start the discussion. When we receive the OK message, we can start to send informations. More informations about the discussion between server and client in a future section.

A second thread is created to populate the machines with the public keys. The thread read the json files and populate the machines. More informations about the population in a future section.

Installation

The installation is done manually by the admin. He install the script and make it run at startup with a process manager like systemd or pm2.

Configuration

The configuration is done by the admin in `settings.py`. He can change the port of the server, the path of the json files, the name of the admin user, etc. Because this is the main script, the configuration here must be done with care.

Employee side

The client side is a simple script that connect to the server and import the public key of the employee. The script is run by the employee. More about it later.

Installation

The employee can install the script by downloading it from the server, or by cloning the repository. The script is run by the employee on every machine he wants to work from.

Configuration

Actually, it's difficult to configure the script on *every OS*. The generate_key script works differently on Windows and Linux.

Machine side

Configuration

The different machines in the pools have a special configuration.

- Firstly, they have a special user `ssh-admin`, with a special *password or a key*, and the user is in the sudoers group. This user is used to connect the handler via SSH to the machine and run a root shell to populate.
- Secondly, they must have the SSH service running, and the port must be open in the firewall.

- Thirdly, the ssh config file must be configured to forbid password authentication and to allow only key authentication.

```
sudo useradd -m -g sudo -p $(openssl passwd -1 $password) -s /bin/bash ssh-admin
```

Sudoers

```
ssh-admin ALL=(ALL) ALL
```

ssh config

```
PasswordAuthentication no
PubkeyAuthentication yes
```

Installation

Actually, the installation is done manually. The admin create the ssh-admin user, and add it to the sudoers group. He configure the ssh config file to allow only key authentication. It can be done with a script, but it was not the priority.

Discussion between server and client

The server is listening on port X for incoming connections. When a client etablish a connection, the server thread it and start the discussion. When we receive the OK message, we can start to send informations.

Steps :

- 1. Server send the OK message, the discussion is started
- 2. Client send his email address
- 3. Server look for the email address in the json file, if it's not found, the server send the error message and close the connection, else the server send the OK message and wait for the password.
- 4. Client send the password given by his password manager.
- 5. Server check the password, if it's not correct, the server send the error message and close the connection, else the server send the OK message and wait for the key.
- 6. Client generate a key pair and send the public key to the server.
- 7. Server write the public key in the json file, for populate it afterwhile, and send the OK message and close the connection.

Checks to do

2/3 Email address

- [ ] Check if the email address is valid

4/5 Password

- [ ] Check if the password valid (format)
- [ ] Check if the email is in the json file, if not, send the error message and close the connection

- [ ] Check if the password is correct (compare with the json file), if not, send the error message and close the connection
- [ ] Check if the email already have a public key, if yes, ask to the user if he want to replace it.

I moved those two last checks to the fourth step, because the error message can help a potential attacker to know if the email address is valid or not.

6/7 Public key

- [ ] Check if the public key is valid

Population and revocation

When the public key is added to the json file, the handler script populate the machines with the public key. The handler script read the json file and populate the machines.

The population is done by the handler script, which connect to the machines via SSH with the ssh-admin user. The handler script create the user and add the public key to the machine.

```
    ssh.connect(hostname=HOST, username=USERNAME, password=PASSWORD)

    stdin, stdout, stderr = ssh.exec_command(f'sudo -S -i')
    stdin.write('AZERTY\n')
    stdin.write(f'useradd -m -g {user_group} -p $(openssl passwd -1 {user_passwd}) -
s /bin/bash {user_name}')
    stdin.write(f'mkdir /home/{user_name}/.ssh\n')
    stdin.write(f'touch /home/{user_name}/.ssh/authorized_keys\n')
    stdin.write(f'echo {public_key} >> /home/{user_name}/.ssh/authorized_keys\n')
    stdin.close()
```

When a public key is removed from the json file, the handler script revoke the public key. The handler script read the json file and revoke the public key.

Admin commands

The handler script can be used by the admin to add or remove an employee from the json file. Those commands are :

- Create a new employee
- Remove an employee (wich will remove his public key from authorized_keys files)
- Add a new pool of machines in pools, maybe with the employee who has access to it
- Add a new machine in pool (wich will populate the machine with the public key)
- Remove a machine from a pool (wich will revoke the public key)
- Remove a pool of machines (wich will revoke the public key of every machines in the pool)
- Revoke an employee (wich will revoke the public key of every machines in the pools, used if the employee is still there but lost his key)
- Launch a check up of the machines (wich will check if the ssh-admin user is present, if the authorized_keys file is present, if the public key is present, if the public key is correct, if the public key is the same as the one in the json file)

Pools

The pools are used to sort every machines by their type. The pools are defined in the json file.

```
{
    "WebServer": {
        "name": "Web Server",
        "machines": [
            "192.168.163.135"
        ]
    },
    "ProxmoxServer": {
        "name": "Proxmox Server",
        "machines": [
            "192.168.163.13X",
            "192.168.163.13Y"
        ]
    }
}
```

When a machine is added to a pool, he fetch any public key he is related to, and add it to his authorized_keys file.

When a key is added to the handler, the handler populate the pools with the new key using both json files.

To remember

- [x] Admin commands
- [ ] TLS Discussion between server and employee
- [ ] The password must be ciphered in the json file, and gived to the employee in a secure way
- [ ] The password must be given by the employee to configure the account on the machines
- [ ] Verify the data file on every ssh connection
- [ ] The key generation must work on every OS
- [ ] Cron job to verify the authorized_keys files, update them if needed
- [ ] The configuration on the machines must be done automatically, not manually, maybe the handler script can do it when a machine is added to a pool
- [ ] Change the json file format to be more readable, or use a database
- [ ] When the admin add a new machine to a pool, the handler must populate the machine with the public keys of the employees who have access to the pool
- [ ] Generate the password of the employee automatically, and send it to him automatically too ?
- [ ] Add many pools when creating a new employee
- [ ] JSON acces must be verified to not crash the program
- [ ] Verify if the pool exist, if the ip exist, etc
- [ ] Admin.py must use jsonFunctions.py
- [ ] Verify add and user deletion
- [ ] Handle many pools when creating a new employee or populating a machine