Three Tier Data Security Model for Cloud Computing

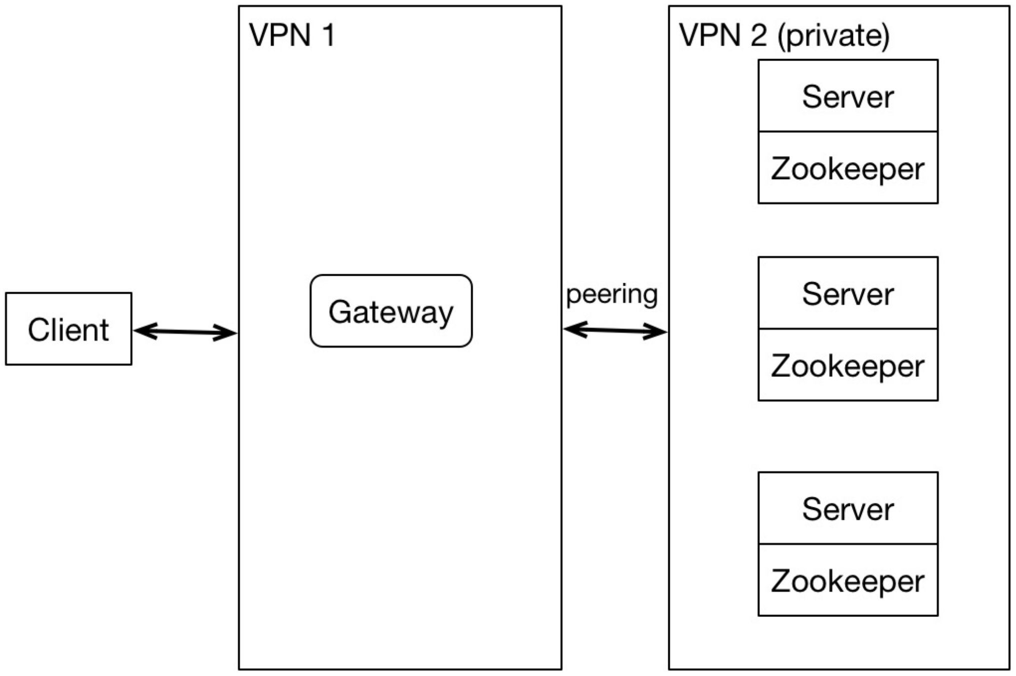
1. Problem Definition

In cloud computing, users typically do not deploy their application on their own computers or servers. Instead they tend to rent infrastructures from cloud providers. The users depends on the security services of cloud providers to design their own security policies. On the other side, a cloud provider usually has a single security architecture but has mulitple customers with different security requirement. The project try to find a data security model applied in the cloud side to ensure the security of cloud users.

2. Conceptual Design

This project is based on the paper *Enhanced Data Security Model for Cloud Computing*[1] , which proposes a three tier structure of cloud security, including user authentication layer, data encryption layer, and backup & recovery layer.
This project basically follows the structure suggested in the paper, and also adds functions to enhance the security. The major function includes,

- OTP (i.e. one time password) authentication. To use this system, each user needs to register an account and pass the user name and password every request. Each password can only be used once so that it is not vulnerable to replay attacks.
- Multiple encrytion choice. The system provides both symmetric and asymmetric encryption algorithms, including RC2, DES, and RSA, users can specify the algorithm when starting the server.
- Data synchronization and data recovery. By default, the application uses zookeeper as the database, it provides a strong consistency. As long as half of the nodes are alive, there is no data loss. Also, if the user chooses to use other databases, the system also provides an interface for data recovery. When the connection of a server is lost, an event will be triggered. User can backup current data so that the data can be recovered when the server reconnect.
- Network isolation. Servers are isolated from the Internet. The client communicates with the server through the gateway which will validate the authenicaiton and decrypt data.



The picutre above shows the architecture of the system.

- Client - The client provides simple writing and reading operation by connecting to the gateway.
- Gateway - The gateway is located in VPN1 which is open to Internet. It accepts the requests from clients, validates authentication, and redirect to application server through peering connection.
- Servers - Servers are the application server developed by user. In each server, there also deployed a zookeeper server which monitor the connection of each server.

3. Implementation Description

Environment

The project is written in python and runs on Linux. It depends on the following python packages: kazoo (zookeeper client), pyblake2 (inreversible hash), pydes (DES)

Client

The client accepts three arguements to start - user name, cipher algorithm, and gateway address. It provides the following functions:

1. initalization - When initalization, the client will exchange the key with server. Depends on the encryption algorithm chosen, the key could be either symmetric or asymmetric. The client will send the following message to *http://your-gateway-address/key* and get the public key of server if it is a asymmetric algorithm.

```
{
  "key": "DN3N39D83733-1=3';",
  "mode": "symmetric"   # or asymmetric
}
```

2. register - This method accepts a seed and then generate 1000 passwords saved in a file. Each password is generated by computing the inreversible hash of the previous one. Then it sends the username and the last password to gateway, and remove the password from the password file. Define cp as the last password and gp as the password saved in gateway, they follow the relation:

   $$gp = hash(cp)$$

   Everytime client makes a request, it sends its username and password to gateway. The Gateway will validate if the passwords match the euqation, then replace the password saved in gateway with the new password. The client will also remove the last pasword in its password list.

3. forward - forward function will make a request to the application server. In the header of the http request, the client puts the relative url and authentication. So the header will look like
   URL: "/service/get"
   Authentication: JXUE93F839SNEJ-=3;I82M
   In the body, the client puts the data needed for the request then send the request to *http://your-gateway-address/forward*

4. encryption - When initialization, client and gateway have exchange the key of the chosen algorithm. Before making requests, data will be encrypted by specified algorithm which is a argument when starting the client. It is very easy to add a new encryption algorithm, in the file *util/cipher.py*, just add your code like this,

```python
##
# RSA
##
class RSA:
    is_symmetric = False
    key_size = 1024

    @staticmethod
    def encrypt(key, plaintext):
        cipher = RSA.importKey(key)
        msg = plaintext.encode('ascii')
        return cipher.encrypt(msg, None)[0]

    @staticmethod
    def decrypt(key, cipher_text):
        cipher = RSA.importKey(key)
        return cipher.decrypt(cipher_text).strip(b'\n').strip(b'\x00')

```

Gateway

1. connection detection - In zookeeper, data are stored in tree structure. For an example, /nodes/node1 means there is a node named "node1" under "nodes". When the application server starts, it will append a ephemeral node under "nodes". For example, If there are two servers exist, then there exist two nodes, "node1" and "node2", under "/nodes". Ephemeral node means the data is hold by the current zookeeper client (here is the application server), whenever the client disconnects, this node is deleted. By monitoring

the nodes under "/nodes", gateway can acquire which servers are actually alive.
2. forward - In zookeeper, each node contains a value, the gateway obtains server IPs by fetch the value of nodes under "/nodes". e.x. the value of node "/nodes/node1" could be "10.1.0.33:80". When the gateway receive the forward request from client, it will randomly choose a alive application server and forward the message to it. The target url contains two parts: the ip address, which is fetched from zookeeper, and the service url, which is fetch of the http header.
3. authentication & encryption - These are already descibed in client section.

Application Server

Application servers are actually developed by users. This project contains a simple server for test purpose, which has the following function.

1. echo - print the http body of the request
2. put - write a key-value pair into zookeeper
3. get - get the value by key

4.User Guide

Basic Environment

1. Install python 3.5+ and python-dev
2. Install pip
3. Install pydes, kazoo, pyblake2

Zookeeper

1. Install zookeeper
2. Modify your-zookeeper-dir/conf/zoo.cfg, add all zookeeper nodes
3. Create a file named "myid" under your-data-dir/. The content of the file is the identifier of this zookeeper server.
4. Start zookeeper
5. After all zookeeper servers are started, run zookeeper client and run command `> create /nodes ''`

Application server

1. Copy the whole project to server.
2. Open /server.py
3. Modify port and ip address under `Server.__init__()`
4. Modify node name under `Server.register_server()`. e.x. db.append("nodes", "`node1`", self.ip+":"+str(self.port), ephemeral=True). Modify `node1` to name of the server.
5. Run command `> sudo python server.py`, make sure the version of python ≥ 3.5

Gateway

1. Copy the whole project to gateway server.

2. Open /gateway.py
3. Modify zookeeper_nodes list to the name of servers under `GatewayConfiguration.__init__()`. e.x. self.zookeeper_nodes = ["node1", "node2", "node3"]. The name here is the name you set in step 4 of Application Server.
4. Modify the port in `Gateway.run()`
5. Run command `> sudo python gateway.py [RC2|DES|RSA]`

Client

1. Copy the whole project to client.
2. Run command `> python client.py [username] [RC2|DES|RSA] [gateway_ip]:[gateway_port]`

The commands of the client includes:

1. `> register [password_seed]`, genrate a new password list and register the user on gateway
2. `> forward [data]`, send data to server, which will be printed on server
3. `> put [key] [value]`, send a key-value pair to server and save them in database.
4. `> get [key]`, get the value of key.
5. `> show log`, enable debug logs, more information will be printed.
6. `> show password`, show the current password in OTP.

5.Self Evaluation

The goal of the project is to make a security framework for cloud computing. It focus on four part.

1. Authenication
   OTP is adopted in the system which is stronger than other repeatedly used passwor. Every request must have the valid authentication header otherwise it will receive http code 401.
2. Encryption
   The system provide both symmetric and asymmetric encryption methods. All data send to gateway and receive from gateway are encypted. Also, an interface is provided that users can add more encryption algorithm easily.
3. Data backup and recovery
   Zookeeper is used to ensure the availability and minimize the data loss. If users choose not to use zookeeper as database, the system also trigger a event when some servers are down, so that proper measures could be preformed and reduice the loss.
4. Network isolation
   All application servers are in a private network (VPC), which can not connect to Internet. Only gateway server can connect to application servers by peering connection, so the application servers are not vulnerable to the attackers from Internet.

Also as a cloud computing framework, it is also adepted at dealing with a group of servers. The gateway can automatically detect the existing server and distributed flow evenly.

7.References

[1]. Mohamed, Eman M., Hatem S. Abdelkader, and Sherif El-Etriby. "Enhanced data security model for cloud computing." In Informatics and Systems (INFOS), 2012 8th International Conference on, pp. CC-12. IEEE, 2012.