Authors: Floran NARENJI-SHESHKALANI (643166) & Jean-Marcellin TRUONG (643357)

1 General idea

The original design is composed of several IOT clients exchanging with a single server on the same local area network (called IOT LAN further on). The stated objective is to move that server to the cloud, and make it so that this server can be scaled up while keeping the communications secure without usage of HTTPS. The servers in the cloud will have their own local area network (called cloud LAN further on). While cloud servers would normally be accessible over the internet, we will assume in this case that neither LAN network is connected to the Internet.

To enable safe communications, we will setup a site-to-site VPN over the Internet which will permit both sites to communicate with one another.

For our scalability goal, we will use Docker containers to host our back-end servers.

2 Proof of Concept limitations

2.1 Docker

While the scenario intends for this setup to be used in a corporate environment, we have for obvious reasons to simulate such an environment. For that purpose, we chose to use Docker containers, which allow us a quick and easy way to setup multiple simulated machines and to script them to our needs. The main advantage of Docker over traditional virtual machines is of course reproducibility.

2.2 IOT device & server

The specifications state the limitations of the IOT device but nothing about it's actual functions. Therefore, believing this to be of no interest, we have created a simple python script that is running inside its own Docker container as a way to simulate the IOT client. This script simply sends a plain-text HTTP GET request to a specified IP address/port containing the current date.

Just the same, the server is a simple stateless python script running inside its own container. The server receives the data from the client over HTTP, prints it out and does nothing more with it. This makes the server very easy to scale, but it also serves no obvious purpose.

3 Technical choices

3.1 VPN

We considered various options for the VPN software, but it finally boiled down to picking between OpenVPN and Strongswan. It seemed that both softwares could satisfy our usecases and seemed equally strong and safe.

First of all, we already had past experience setting up OpenVPN on our own dedicated servers. Then, Strongswan implements IPSec, an industry standard that is quite widespread on common routing infrastructure. A client who would like to use its own router could readily configure it so that it connects to our VPN (such configuration however is outside the scope of this project), which is a good selling point for the customer and a decrease in cost for our company.

Thus, we elected to use Strongswan as we felt it was both a sound choice and a good learning experience.

3.1.1 Authentication

For authentication, we initially wanted to set up our own public key infrastructure such that we would have a root certificate and one certificate for each of our gateways. Unfortunately, as of now, we have not yet managed to get it working and authentication systematically fails, even though we believe all the certificates are signed adequately.

Consequently, we've fallen back onto onto Pre-Shared Key (PSK) authentication, where the password is stored in clear-text in the Strongswan's configuration file. Because our VPN usage scenario is so simple (there will be at most two participants in the VPN), we believe this to be as easily maintained as a PKI-based authentication solution.

In terms of security, we believe both solutions to be equivalent in most regards: if the machine is compromised, then a PSK or a private key can be extracted in the same manner, and both can then be replaced easily by an admin. If the private key is password protected, Strongswan will ask for it on startup, but the same can also be applied to PSK. For an ongoing connection, having access to the original mean of authentication is useless as it is only used for initial authentication and not as a session key.

However, a PKI would still be best for multiple reasons. First of all, an existing organization is likely to have its own PKI already, and it would be best to leverage such an existing system instead of "rolling one's own" through a PSK.

Then, while our current needs are quite simple and are easily satisfied by a PSK, our future needs might not be: we might have more sites, more clients and potentially a need for revocation. Such a change in need is clearly a usecase of the PKI: it provides a tremendous advantage in scalability and maintenance.

Additionally, in a configuration with multiple customers, a successful attack against of the cloud gateway compels a full replacement of all PSKs, whereas it would suffice to replace the certificate and key when using a PKI.

Finally, a key provided through a PKI will always be fairly long, whereas a PSK chosen by sloppy students might only be 7 characters long (where it should be a very long random string).

3.1.2 Communications security

Strongswan accepts 3 paramaters related to communications security. For encryption, we picked AES256-CBC for its MAC properties.

For integrity, we picked prfsha512, which is sha512 combined with a pseudorandom function.

Finally, we use elliptic curve based Diffie-Hellman with curve ecp512bp.

3.2 Routing

For every customer network, we host a Docker container that runs the customer's back-end server. Such a setup allows us to efficiently use the resources of our cloud servers: a Docker container has a very small baseline resource cost and our application is not computationally heavy anyways.

It also avoids using a reverse proxy: because each container has its own IP in the cloud LAN, the IOT devices can contact it's target directly.

Moreover, this is also an advantage from a security standpoint: for each VPN tunnel, we can ensure using the firewall on the cloud gateway that communications are only flowing between the devices belonging to the same customer.