



SMART RESTAURANT

NETWORK AND COMPUTER SECURITY

ALAMEDA - GROUP 12

79018	Diogo Ferreira	
79620	Tiago Vicente	
79719	Francisco Santos	

DECEMBER 09TH 2016
INSTITUTO SUPERIOR TÉCNICO

1 Problem

A SmartRestaurant can be everything customers dream for, orders will never be wrong again and they will not be worried about payments again. Although this may seem exciting from the user's point of view, it can be extremely hard to ensure that everything works with high security.

In this case, we will need to make sure that, from the customer side, he cannot be fooled into communicating with a fake restaurant or that someone else is speaking with the server on his behalf, that he can be assured that the order is not tampered with, and that his payment and consumer data is not public. From the restaurant side, it needs to be sure that a client cannot repudiate an order, that their network and infrastructure is not compromised and that there are not clients using their service to attack property of others.

2 Requirements

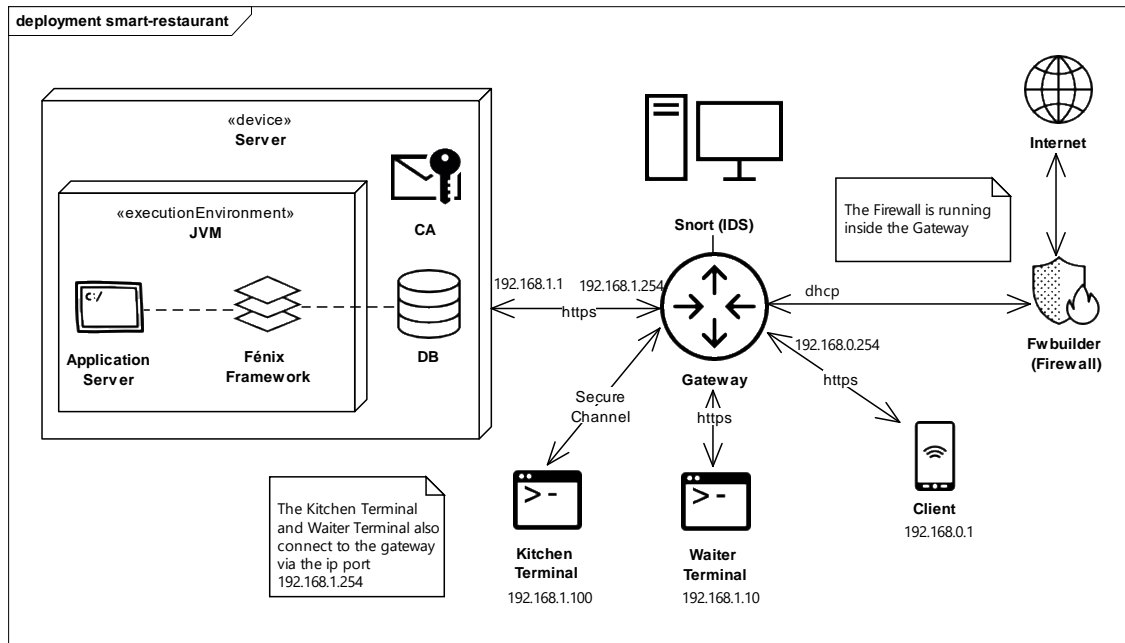
- i Connection between nodes and server must be secure. Therefore we must assure:
 - (a) **Authenticity**: ensure that restaurant's clients don't make requests on behalf of others;
 - (b) **Integrity**: avoid tampering with orders made by the clients;
 - (c) **Freshness**: avoid obsolete orders to be processed;
 - (d) **Non-repudiation**: guarantee that a client cannot deny making an order;
 - (e) **Confidentially**: help guarantee integrity and avoid eavesdropping from other clients of: orders or payment data.
- ii The service cannot be victim of **Denial of Service**;
- iii Protect the network and infrastructures against internal and external attacks due to the mobility of the client's devices;
- iv Restaurant clients must have access to the Internet;
- v Outgoing requests must be "**well behaved**", i.e. don't cause harm to the outside;
- vi Clients need to have unique accounts registered in the system;
- vii Keep clients records with **ACID** properties.

2.1 Assumptions

We assume: QR Codes cannot be tampered with, smartphones will never be in promiscuous mode, the routing table in the gateway is static, Fénix Framework

and Tomcat are safe, the client never cancels an order, the client always pays and the interaction of the system begins with a user reading a QR code and inserting his credentials in order to login and create a session.

3 Solution



The interaction of the system begins with a logged in user reading a QR code, creating a session. The user can make requests using QR codes in the menu. When the user makes a request, it passes through the gateway (ii) where the routing mechanism and IDS come into place. The routing assures that the clients can connect to the server or the outside “world”. (iv) The IDS enforces good internal behavior of the network. (iii) The CA distributes the public key certificates for the internal components except the user devices.

Once the request from the client reaches the server, it processes the request checking its validity within the business model. The server will use the database to log payments and the amount owed, maintains the state of a request, checks the validity of a user account and on start-up gets the menu QR code translation. (vi) After, the server routes the request to the KitchenTerminal. When ready, it routes back to the server. On the server, it updates the DB setting the request to ready and notifies the WaiterTerminal. The payment is done after the meal is over, recurring to PayPal API.

If the message is directed to the outside the firewall will filter it according to a packet-filter logic (v). When the response comes, it should behave the same way. (iii)

3.1 Initial Proposal

- ✓ Application, server and terminals with **HTTPS** connections and sanitization of inputs;
- ✓ Notion of user accounts, including creation and payment operations;
- ✓ Using QR codes to choose the food and create a session between account-table;
- ✓ Database storage for the user accounts, using transactional and persistent domain model;
- ✓ Run the system in a local network with no Internet access.

3.2 Intermediate Proposal

- ✓ Implement the connection between the server and the waiters terminal, using the connection requirements **(i)**;
- ✓ Configure a Gateway for the internal network in order to intermediate the connection between the clients and the server.

3.3 Final Proposal

- ✓ Configure a Network-based Intrusion Detection System on the gateway;
- ✓ Enable Internet access to the client;
- ✓ Configure a firewall between the internal network and the external network.

4 Results

- The reading of a QR code to establish a session was simulated by a random number from 1-10, and to request food from the menu were simulated by writing the name of the dishes when asked;
- PayPal payment was simulated by just pressing enter;
- The shared keys are distributed **manually** and the certificates via **CA**;
- **Two-factor authentication** is simulated with a static code (always 123456);
- The **HTTPS** connections were established using **[a]**, it assures **validation of the server**;
- The main server was implemented using **[b]**, a persistent domain model, assures **(viii)** and **(vi)**;

- The connection between MainServer-Kitchen and MainServer-Waiter was established using [c], which guarantees all the requirements in [2.i];
- Implementation of **salt** for register and password verification on the server side [d];
- The **firewall** [e] in conjunction with the **gateway** allows us to meet the requirements (iii), (iv) and contributes for (ii) since the client can only talk with the server - **static routing**;
- The **IDS** [f] was not implemented due to bugs in the virtualization host-guest. It's a known bug, but we didn't know. It affected the performance of the overall system, because messages were getting duplicated resulting in a potential **DOS** of the client. Also, we discovered that to achieve the requirements (ii) and (v) we needed snort to run in **IPS** (Intrusion Prevention System) mode.

5 Evaluation

Our solution provides answers to all the requirements except for (v) since we don't have the IDS/IPS, although it has some flaws:

- A client can only request one of each dish;
- There's a problem on making sure that the client pays for its order, so we assume that the client always pays;
- Due to problems separating jax-ws handlers on the MainServer module we had to implement handlers on the communication MainServer-Waiter.

6 Conclusion

Our project presents a solution for a Smart Restaurant that focus on providing a secure implementation of the business model. Provides assurance that the servers won't suffer outside attacks, assures that if the DB is leaked the passwords of users won't be compromised and is secured against known web vulnerabilities (SQL injection, XSS, etc).

7 References

The tools that were used: [a] [Tomcat](#) - HTTPS connection, [b] [Fénix Framework](#) - persistent domain model, [c] [JAX-WS](#) - "home made" channels, [d] [Crackstation](#) - salt implementation, [e] [Fwbuilder](#) - GUI for iptables, [f] [Snort](#) - IDS/IPS.