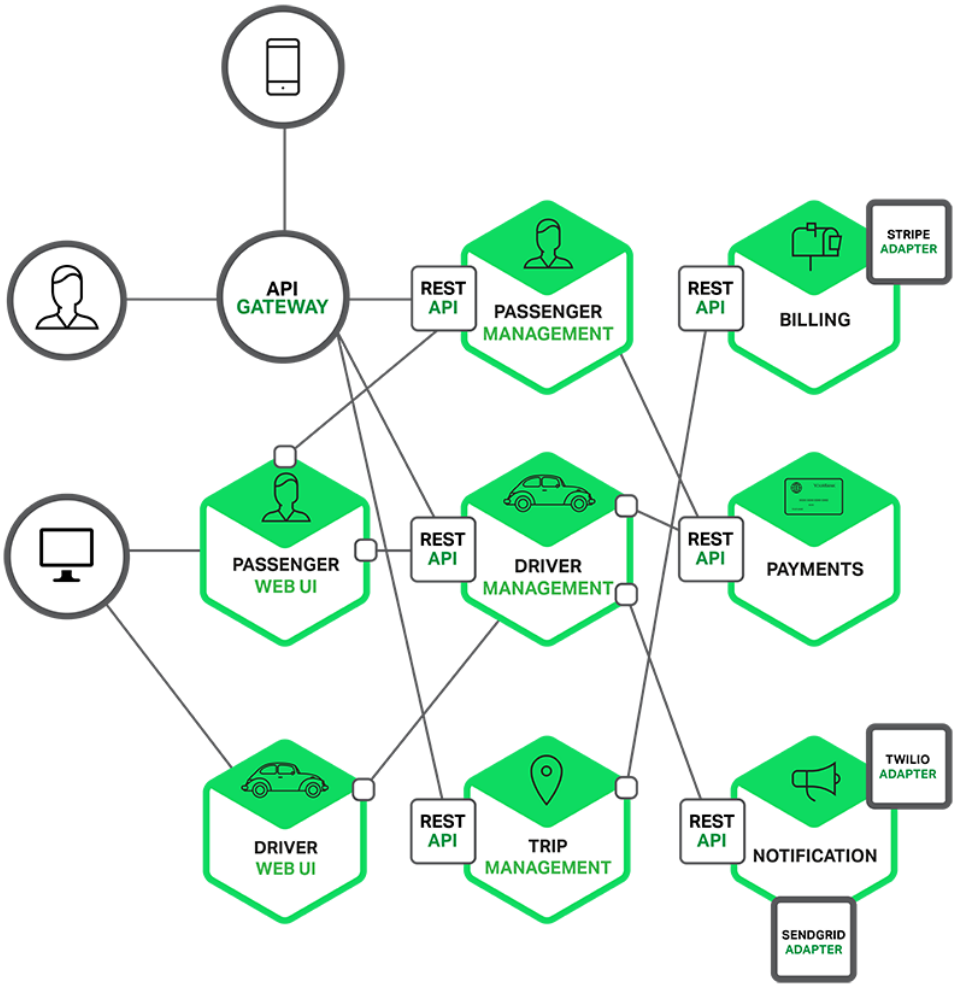
Haystack.im

SECURITY ANALYSIS

Piyush Jadhav, August Tan | Application Security | October 6, 2015

Architecture Overview

- Haystack is based on Microservices architecture.
- The application is decomposed into individual services, and deployed separately from one another on separate hosts.
- Each microservice is aligned with a specific business function, and only defines the operations necessary to that business function.
- These microservices are run on containers in Docker.
- These Docker Containers are deployed on Amazon Web Services EC2 instances.
- Uses discovery service. When a new service instance is created, the rest of the network can automatically find it and start to communication with it.



1: Microservices Architecture Example

Security Concerns

Unauthorized access to the services

- Every call to each service should be authenticated. Many systems have an authentication gateway that authenticates the initial API call. The Internal calls are not authenticated. Thus if anyone has access to the network, they can access any service inside including the one that contains the sensitive data.
- There should be a secure way for the services to authenticate each other. One way to do that is to user SSL certificates for authentication
- The Services should always pass the identity of person who originated each call while making calls to internal services. Even if the information is not directly required by the service. This ensures all calls in the systems are accounted for.
- Services should NOT let their callers access all the APIs that a service offers. They should provide access to just the ones it needs to fulfil its function.
- If an attacker owned a service, there should be some mechanism to stop them from requesting anything from its downstream services.
- The request received, should be checked for tampering.
- Services should be protected from replay attacks. HTTPS and TLS/SSL can be used to protect against replay attack.

Messaging Middleware Security

- The application may use a messaging middleware like apache Kafka for asynchronous messaging between the services.
- Attacker who gets hold of one of the services messaging credentials should be limited access to sensitive data.

Password Security

- Keep up to speed with the state of the art in password storage
- Using just salted password is not enough these days. The attackers, with today's hardware can calculate ridiculously high number of hashes quickly. Below are the numbers that demonstrate how many hashes can be generated in a second on a 25-GPU rig.

Scheme	**Tries/sec**
NTLM	350,000,000,000
MD5	180,000,000,000
SHA1	63,000,000,000
SHA512Crypt	364,000
Bcrypt	71,000

- Therefore it advised that you use modern techniques like scrypt ,bcrypt or PBKDF2
- When a password is entered wrong, what feedback do you give? Is there a difference in response time if the user does not exist in the database? This could be used for username enumeration.

Security against DDOS Attacks

- With the sheer volume of today DDOS attacks, it is very difficult to keep them out. Akamai Technologies shared new details recently of an existing botnet that is now capable of launching 150+ gigabit-per-second (Gbps) DDoS attacks from Linux systems infected by the XOR DDoS Trojan.
- There should be an clear process defined in event of a Denial of service attack.

Service Discovery Security

- Is the connection information into a globally accessible location?
- Are the connections encrypted with SSL/TLS. ? satisfactory. However, most applications would probably benefit from additional security.
- There are a number of different ways to address this issue. One solution is to continue to allow open access to the discovery platform itself, but to encrypt the data written to it. The application consumer must have the associated key to decrypt the data it finds in the store. Other parties will not be able to access the unencrypted data.
- For a different approach, some service discovery tools implement access control lists in order to divide the key space into separate zones. They can then designate ownership or access to areas based on the access requirements defined by a specific key space. This establishes an easy way of providing information for certain parties while keeping it private from others. Each component can be configured to only have access to the information it explicitly needs.

Database checks

- Each Service has its own database. This can be a major security issue, because then you don't even have a centralized way to determine if bad things are happening. It creates a lot of issues around where you would go for a single source of truth for validating compliance, or validating any type of check or rule that you're trying to implement for security control.

Script Insertion

- Various Script injection attacks are possible. Therefore it is very important to ensure that the input data is properly sanitized.