# Defense In-depth security framework for Netflix OSS Micro Services

A dissertation submitted for the Degree of Master of Science in Information Security

P.A Walpita

University of Colombo School of Computing

2017

# Declaration

The thesis is my original work and has not been submitted previously for a degree at this or any other university/institute.

To the best of my knowledge it does not contain any material published or written by another person, except as acknowledged in the text.

Students  Name: P.A Walpita

______________________

Signature:                                                  Date:

This is to certify that this thesis is based on the work of Mr. P.A Walpita under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Certified  by:

Supervisor  Name:          Dr. Kasun De Zoysa

______________________

Signature:                                                  Date:

# Abstract

Micro Services architectural pattern has emerged in recent years mainly because of its capabilities to handle high data volumes in a robust manner. The perceptions like Dev Ops and Domain Driven Design also helped to develop this architectural pattern in to its current heights. Many enterprise systems which has large amount of transactional data volumes adopting Microservices architecture because of many enablers it provides. The Security of Microservices considered as utmost important feature because of the security threats escalated in recent years. The threats that are targeting Microservices eco system can be categorised as external and internal threats.

Many industrial level Microservice implementations taken precautions about protecting the Microservices eco system from external attacks. The security measurements that are taken to protect a Microservice eco system from internal attacks are also an important aspect if the internally communicating data are sensitive in nature. Internal threats can be identified as vulnerabilities which can be exploit by an adversary internal to the organization. Netflix is one of the early adopters of Microservices architectural pattern and the Netflix OSS emanates as an open source platform with a practical Micro Services success story. This Paper discusses about hardening the Internal service calls of the Netflix OSS Microservices and discusses the possibilities of eliminating vulnerabilities within the internal perimeter. The measurements that are taken to protect internal microservices in Netflix OSS can be adopted generally in any other Microservice eco system as well.

# Acknowledgement

Firstly, I would like to express my sincere gratitude to my supervisor Dr. Kasun De Zoysa, for continuous support in mentoring me and for encouraging my research, guidance and suggestions throughout the research.

I would also like to thank my lecturer panel including Dr. Chamath Keppitiyagama , Dr. Ajantha Atukorale, and Dr.Gihan Senevirathne for serving as my panel members even at hardship. I also want to thank you for letting my defence be an enjoyable moment, and for your brilliant comments and suggestions.

I would thank Mr. Magnus Larsson for hosting the Netflix OSS sample source code in the Git hub and providing detailed explanations about the Netflix OSS ecosystem in his blog post.

I would especially like to thank my wife (L.P Dilrukshi) for the support she gave me during the endless nights and countless hours I had to spend doing my research. The sacrifices you have made, the love and guidance you gave and your prayers have made me who I am today.

# Table of Contents

# List of Figures

# List of Tables

# List of Abbreviations

| Acronym | Full Name |
|---|---|
| ADO | Advanced Data Object |
| AES | Advanced Encryption Standard |
| AMQP | Advanced Message Queuing Protocol |
| AOL | America On Line |
| API | Application program Interface |
| AWS | Amazon Web Servers |
| CA | Certificate Authority |
| CORS | cross-origin resource sharing |
| DDD | Domain Driven Design |
| HIPPA | Health Insurance Portability and Accountability Act |
| HMAC | hash-based messaging code |
| HOK | Holder of Key |
| HTML | Hyper Text Markup Language |
| HTTP | Hypertext Transfer Protocol |
| IIS | Internet Information Services |
| IPC | Inter Process Communication |
| JSON | Java Script Object Notation |
| JWT | JSON Web Token |
| LB | Load Balancer |
| MiTM Attack | Man in The Middle Attack |
| MQ | Message Queue |
| MSA | Micro Services Architecture |
| MVC | Model View Controller |
| Netflix OSS | Netflix Open Source Software |
| ORM | Object Relational Mapper |
| OWIN | Open Web Interface for .NET |
| PC | Personal Computer |
| PKI | Public Key Infrastructure |
| POJO | Plain Old Java Objects |
| POP | Point Of Presence |
| REST | Representational State Transfer |
| SAML | Security Assertions Markup Language |
| SCB | Spring Cloud Bus |
| SOA | Service Oriented Architecture |
| SSL | Secure Socket Layer |
| SSO | Single Sign On |
| SSO | Single Sign On |
| TLS | Transport Layer Security |

| | |
|---|---|
| UAA | User Account and Authentication |
| URI | Uniform Resource Identifier |
| URL | Universal Resource Locater |
| URL | Uniform Resource Locator |
| XSLT | Extensible Stylesheet Language |

# 1 Introduction

With the advancements in internet infrastructure and bandwidth, content streaming over the Internet has increased during the last decade. Netflix is one of the leading video content providers who uses streaming technologies to deliver the video content to the end user. End clients are using wide range of endpoint devices such as Smart TVs, Streaming media players, Gaming consoles, Set-top boxes, Blu-ray players, Smartphones , tablets,PC and Laptops [1]. Netflix supports over 80 million streaming subscribers worldwide and average of 20 million access the streaming services simultaneously [2].

Netflix is using the Netflix OSS open source framework to support the streaming services. To support high demand and high volume of data exchange, Netflix needs more resilient and robust architecture. Netflix adopted the Microservices architecture pattern in its early stage of development and formalized a comprehensive standard of industrial usage. Netflix uses the Amazon Web Services (AWS) to host Micro Services back end and its characteristics of on-demand provisioning helps to spawn any number of services when the demand is high.

## 1.1. Research Domain

### 1.1.1 Research Problem

Many industrial level implementations of the Micro Services architecture focuses only on the perimeter level security. This is true in the context of Netflix OSS as well. The Netflix OSS secure the perimeter level Edge servers using a JWT token from unauthorized access. This is the only programming level secure precaution been made in order to protect the Edge service. But the internal Micro Services are left unsecured. The internal Micro Services are vulnerable to internal attacks. This research focuses on how to secure the internal micro services and implement proper Authentication and Authorization mechanism in order to reduce internal attack vulnerabilities.

### 1.1.2 Significance of the Research

The identified problem is the insecure nature of the Netflix OSS internal Micro Services which can lead into internal attacks and data breaches. The research is concentrated on the REST API security hardening in the context of Micro Services.

Even though the research is based on Netflix OSS framework, the research outcome can be applied to any generic MSA which is based on the REST API.

The problem this research attempts to address is “How to secure Netflix OSS Micro Services REST APIs from internal attacks”. There are plenty of open source Microervices architecture frameworks in the industry where develpers can adhere to. But most of these frameworks do not implement any internal security mechanishms to protect Microservices from unauthorized access. Hence, those frameworks do have vulnerabilities of internal attacks. This is a crucial factor if the internal microservices are comminicate sensitive information. The reseach is focusing on finding an optimal method to secure internal Microservice API calls from internal attacks. The Netflix OSS is used to apply selected secure mechanisms and test the outcome of the reaserch.

### 1.1.3 Goals and Objectives

The goal of this project is identifying the best suitable methodology and technology to secure Netflix OSS Micro Services from internal attacks and data breaches. Authentication and Authorization of Internal Micro Services calls is the main objective and maintaining the integrity of the service call is also comes as a secondary objective of the project.

Selected security mechanism should not hinder performance and scalability of the existing Microservices architecture.

### 1.1.4 Limitations and Assumptions

The research will be carried out to secure Netflix OSS Micro services from internal vulnerabilities.

Netflix OSS source code is an open source project and the code is available at Git hub[13]. The Proposed solution will be implemented and tested against the source code available at Netflix OSS Git repository.

The streaming technology used at Netflix is a propriety technology and hence, it is not available in the open source Netflix OSS platform. The available code in the Git hub[13] represent sample textual data which is hard coded in the end point services to demonstrate the behaviour of the Netflix OSS Micro Services framework. Security implementations and testing will be using the same sample data shipped with the available source code.

The project assumes the researching security technologies and methodologies are independent from the payload within the Netflix OSS Micro Services calls.

# 2 Literature Review

This section covers the related literature for this research.

## 2.1 Microservices

Microservices security is the focused research area in this study where the main objective is to implement and introduce optimal security mechanism to prevent from internal attacks.

The Micro Services Architecture (MSA) is evolved from the Service Oriented Architecture (SOA) as a specialization[14]. Concepts such as Domain Driven Design (DDD) and Dev Ops helped to enable the age of MSA. Time to the market is a crucial factor in nowadays software and the Dev Ops concepts enable it with continuous delivery. Monolithic system architecture hinders the flexibility of using concepts such as Dev Ops and DDD. The main MSA features are as follows[3].

- Domain-driven design.
- Continuous delivery.
- On-demand virtualization.
- Infrastructure automation.
- Small autonomous teams.
- Systems at Scale

In MSA, services and protocols should be lightweight, smaller and different to Monolithic SOA. Each Micro Service should adhere to the Single Responsibility principle in order to improve the independency of each Micro Service. This in turn makes it easier to add qualities and functions to the service systems at any given juncture. It also enables the continuous independent delivery[4].

The Microservices architectural pattern adopted by many industry leading companies including Netflix, Amazon, Tyro Payments, eBay and Uber. Most of these organizations moved from a single Monolith architectural pattern to Microservices[24]. Requirements for architectural features such as high Resilience, High Availability, Service Oriented Architecture, and Rapid service provisioning during peak hours are common among services provided by said organization. The Microservices architectural pattern guarantees the delivery of such features in a dynamic environment where demand fluctuates rapidly by the client usage.
Following sections illustrate usage of Microservice architectural pattern in the industry in order to get understand how other Microservice frameworks are operating compared to the selected Netflix OSS Microservices.

### 2.1.1 Uber Microservices

Uber moved from an N-tier Monolith architectural pattern to a Microservices architectural pattern when more clients are joined as Drivers and Passengers[25]. Uber migrated the legacy Python based back-end technology stack into the Microservices using Tornado which is an asynchronous framework for Python. This migration helped the Uber to move into the Microservices based architecture without re-writing the existing business logic but just modularising and redefining new communication architecture.

The Uber Microservices architecture featuring Hyperbahn network multiplexing framework along with HAProxy as the routing engine within Microservices. Hypebahn also solves the typical Microservice problems such as Service discovery, Fault tollarence and Real time circuit breaking.

The Uber Mciroservices are provisioned using Docker containers.

Following picture depicts an example of Uber surge pricing system is operating using sevaral Microservices[28]

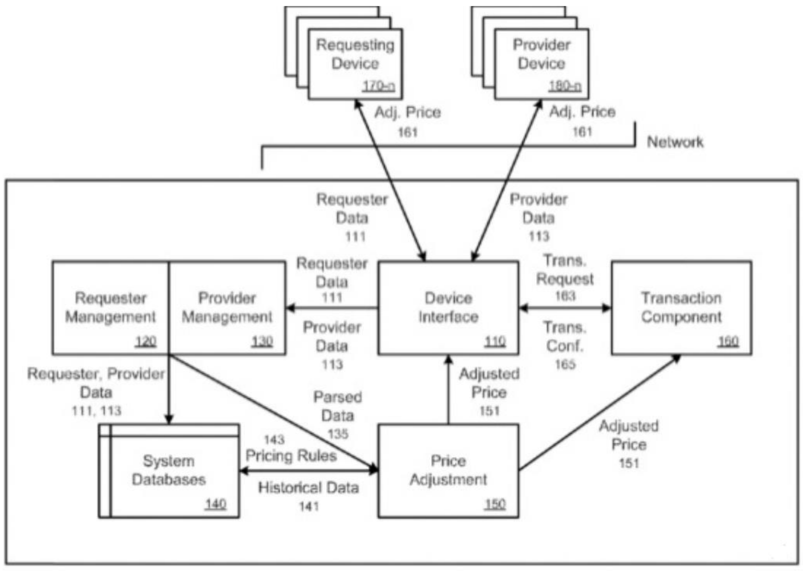


*Figure 2.1-1 Uber Surge Pricing Microservices[28]*

### 2.1.2 eBay Microservices

eBay started  business 1995 using a Monolith system architecture. This system architecture contains Perl, C++ and XSLT.

Eventually eBay has evolved to the Microservices architecture using Java platform. eBay is using Hadoop infrastructure leveraging Storm, Kafka, Spark as the Data-centric back-end of the application[30]. The eBay Front-end application is developed using HTML5 along with JQuery Ajax and Java[31].

The middle layer of the eBay application stack is using Raptor.io , Spring Boot , Embedded Tomcat containers and Java[32].

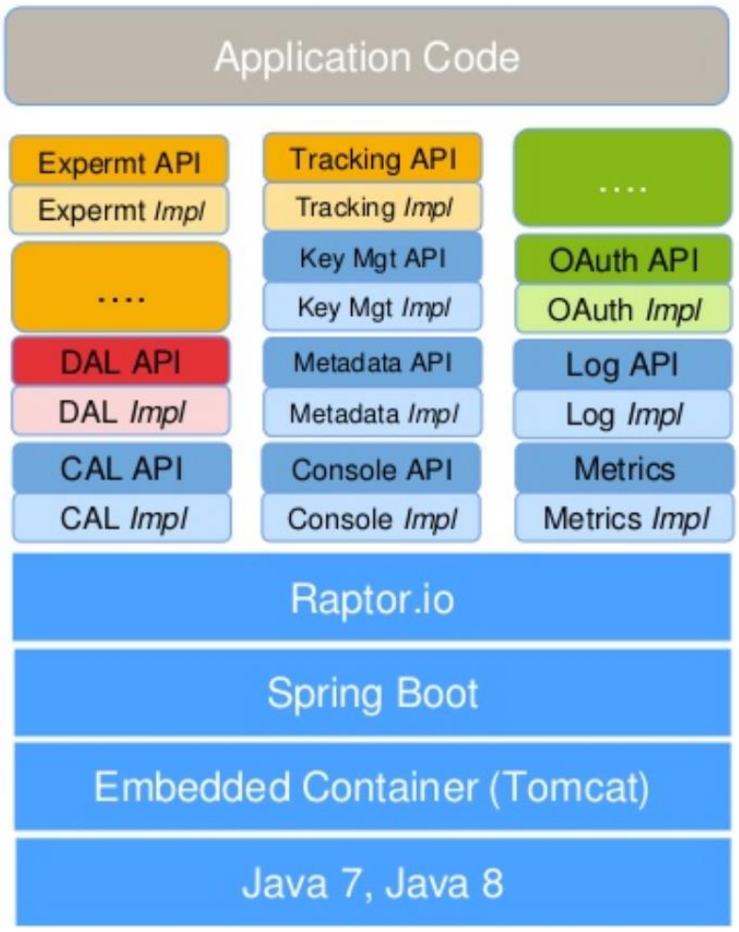


*Figure 2.1-2 eBay Technology stack*

The figure 2.1-2 depicts the technology stack of eBay along with set of Microservices. Microservices are modularised based on the functionality.

## 2.2 Netflix Micro Services

Netflix Microservices are based on Java Spring MVC framework and using numerous other tools such as Eureka , Hytrix , Ribbon and Zuul edger server I nthe Microservices ecosystem.

The Netflix OSS Microservices' using perimeter level security using OAuth2 JWT token[5]. OAuth2 tokens are issued using Spring MVC based Authentication Server. The Edge server communicates with the Spring Authentication server upon receiving of JWT via Rabbit MQ message queue. Authentication server authenticates the incoming JWT and notifies the Edge server.

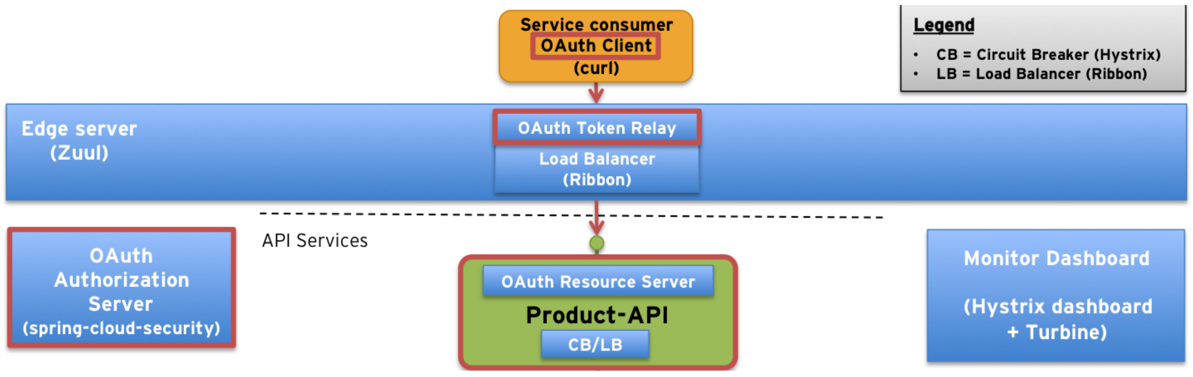


*Figure 2.2-1 Netflix OSS Edger Server Security*

But the internal services are not secured for Authentication and Authorization. Hence it is prone to internal attacks as depicted in the following picture.

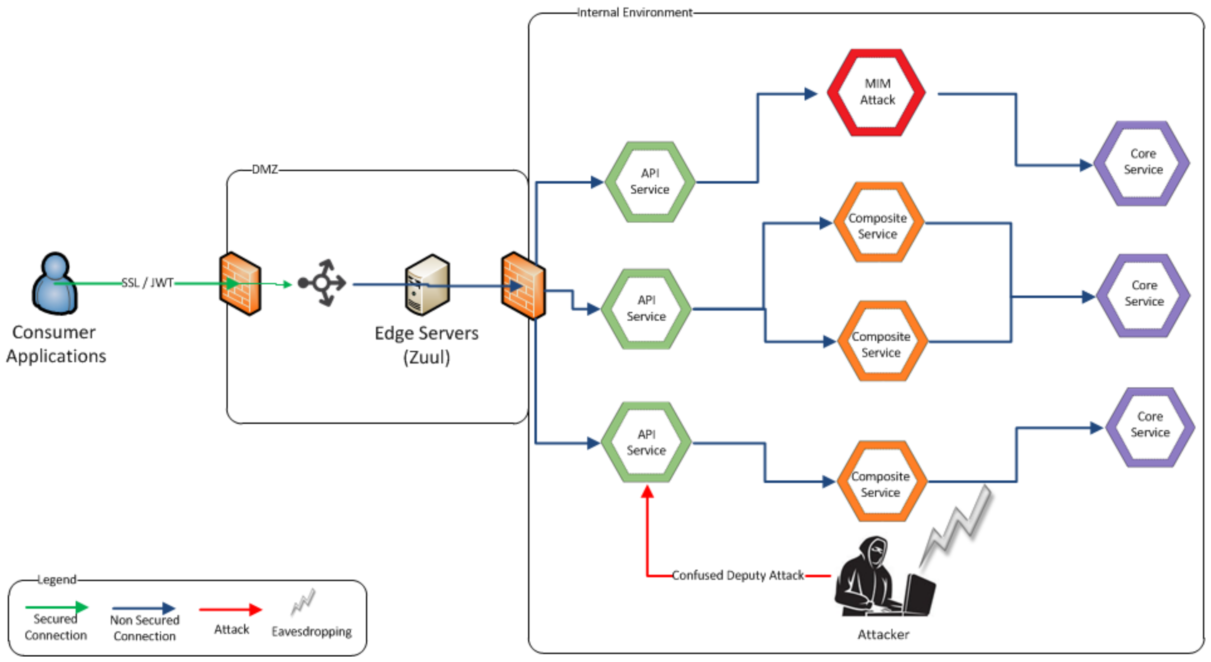


*Figure 2.2-2 Attack vectors to the Internal Micro Services*

Figure 2.2-2 depicts the graphical view of Internal attack vectors of Netflix OSS Microservices.

## 2.3 Netflix OSS Internal Micro Service Vulnerabilities

This section illustrates existing vulnerabilities of the Netflix OSS Microservices in-terms of an internal attack.

### 2.3.1 Eavesdropping

Since the Netflix OSS internal micro services do not implement any privacy or security mechanism, an internal attacker can listen to the communication in between Microservices. This can be demonstrated using a packet capturing tool as follows.

In this attack, the Microservice ‘A’ calls to the Microservice ‘B’ using TCP protocol and the attacker can see the request from A to B and the response from B to A with its content. Hence any unauthorized internal attacker can gain access to any messages or content flowing in between Microservices.

### 2.3.2 Confused Deputy Attack

The Confused deputy attack is , in the context of internal micro services refers to a situation where an adversary can trick a calling service (deputy) into making calls to a downstream service on his behalf that the calling service is not authorized to [3]. This is possible in the Netflix OSS micro services because the called micro service do not perform any authorization about calling Microservice[3]. Hence an Adversary can act as a legitimate internal micro service and perform successful data retrieval upon calling other micro services.

### 2.3.3 Man-in The Middle Attack(MiTM)

An adversary who has access to the network can spawn in an arbitrary Microservice and intercept an ongoing Inter-micro service call. The malicious Microservice can forward the intercepted communication with malicious content or any required modifications to the request. The Victim Microservice will reply to the Malicious micro service assuming it is a legitimate Service. The Malicious microservice can alter the request as well and reply to the Service originator as a legitimate endpoint.

### 2.3.4 Replay attacks

An adversary can eavesdrop an internal Microservice communication and can perform a Replay attack later. This is possible because the Netflix OSS Microservices do not perform any authentication or Authorization in calling Microservice.

## 2.4 Possible Technologies to Secure Netflix OSS Micro Services

There are few possible ways to provide the solution effectively to secure the Micro Services from internal attacks. The following section is elaborating these possibilities.

### 2.4.1 HTTP Basic Authentication

In HTTP basic Authentication, the client is sending credentials using a standard HTTP header to the server or endpoint service. The Receiver checks the received credentials, perform authentication and allowed / disallowed access to the service.

The main advantage of this method is, it is a well understood and well supported protocol[3]. But performing the HTTP basic Authentication over HTTP is problematic, because the credentials are passed to the endpoint server in plain text. The remedy to this problem is using TLS in the channel. TLS is the standardized method to communicate sensitive data such as credentials[23]. HTTPS guarantees the integrity and authenticity of the request and the payload. In order to use TLS, the end point services need to maintain Server certificates. The Certificate Authority needs to be managed and functionalities such as Certificate Issuing and

Revocation are also needed to be managed. This is a problematic scenario in a Microservices environment because of its characteristics. Microservices characteristics such as rapid provisioning and ability to shut down services rapidly hinder managing Certificates and related functionalities.

### 2.4.2 Open Id Connect

Open Id connect which is a framework built on top of OAuth 2.0 protocol is an identity framework and it extends the process of authorization of OAuth 2.0 for authentication mechanism implementation. Open Id Connect can support SSO for any identity provider or any website. OAuth 2.0 framework of authorization can provide an access resource to the customer to gain access on behalf of owner resource.

Open Id Connect authorizes many types of customers for example Java customers, Mobile customers and web based customers in order to verify their user with server based authorization with OAuth 2.0 as its base. Open Id Connect is accessible on over 50,000 plus websites globally and over one billion URLs enabled by Open Id Connect are able due to providers such as AOL, Yahoo, Google and Facebook [9].

The Open Id Connect supports both Authorization and Authentication of the calling service. It is a light weight protocol and have minimum impact to the Microservice eco system. The credentials need to be managed and secured in the client or in the calling service. One of the main disadvantages of using Open Id Connect framework is , it is not a matured technology in the market yet. Hence there could be unidentified security flaws and vulnerabilities.

### 2.4.3 JSON Web Token ( JWT )

JSON objects are transmitted between two or more parties using JSON Web tokens. JWT is an open standard defines by RFC7519. The compact and self-contained JWT tokens can be verified and trusted by the receiver because it is digitally signed[15]. JWT can be signed using a shared secret key (HMAC) or by Public / Private key pairs using RSA.

Authentication and Information exchange are the main usage of the JWT. The Identity server generates a JWT upon successful verification of user credentials and returned to the client. The client needs to store the acquired JWT securely and should send it when the client needs to access protected route or resource. In HTTP the JWT can be passed using the Authorization header using the Bearer schema.

Because JWT is a stateless authentication mechanism, the user state is not required to save in server memory. The server's protected routes or resources will check for a valid JWT in the Authorization header. If the header contains a valid JWT, the calling client will be allowed to access protected resources. JWT self-contained all the necessary information required to perform authentication or authorization. Hence, it is reducing the need to query the database multiple times [15]. The JWT contains less payload compared to SAML and XML. These two aspects are few of the main advantage in the context of Microservices in-terms of performance.

JSON parsers are common in modern programming languages and frameworks. Hence, it can be integrated in to any languages with less effort.

The JWT mainly contains Header , Payload and the Signature. The header contains the type of the Token and the hashing algorithm that used such as HMAC SHA256 or RSA. The Payload contains claims such as issuer , expiration time and subject. Signature verifies the owner of the Token who signs it.

The Netflix OSS Edge server is secured using the JWT tokens in the perimeter level.

### 2.4.4 HMAC Over HTTP

An HMAC is a hash function where applies to the body of a message along with a secret key. [16]. This approach using a hash-based messaging code (HMAC) to sign the request. The client sending an identifier for the shared key and HMAC. This operation uses standard HTTP header. The server then uses its own copy of the shared key and the request body to recreate the hash. If it matches, it allows the request.

One of the definite advantage in HMAC is , it is hard to do a MiTM attack. If a man in the middle alters the request, then the hash won't match and the server knows the request has been tampered with. And the private key is never sent in the request, so it cannot be compromised in transit.

The request is easy to cache when using the HMAC authentication because it is using a shared key. Hence the same request can be persist in the cache.

Both the client and server need a shared secret that needs to be communicated using another secure protocol. This is hard to perform when requiring rapid provisioning which is one of the mandatory requirement when it comes to the Microservices. HMAC is still not developed and recognized as a standard yet. It is still using just as a pattern[3]. Hence, most of the implementations are not yet standardized. Since the HMAC using the same key, it is vulnerable to replay attacks if the request content is same.

### 2.4.5 API Keys

All public APIs from services like Twitter, Google, Flickr, and AWS make use of API keys. API keys allow a service to identify who is making a call, and place limits on what they can do. Often the limits go beyond simply giving access to a resource, and can extend to actions like rate-limiting specific callers to protect quality of service. A more common approach is to use a public and private key pair and manage the key server centrally.

The API Keys solution is easy to development and easy to perform a deployment and managing as well. It also supports rapid provisioning and decommissioning of Microservices as well because of minor footprint to the underlying eco system.

The major disadvantages of this method are , relying on 3rd party tools and centralized key management. Centralized key management can lead in to central point of failure problem while relying on 3rd party tools is not an industrial level standard practice.

### 2.4.6 Holder of Key Tokens (HOK)

The Presenter of the JWT could declare as this specification defines a JSON Web Token (JWT) and processes a (PoP) ‘Proof of Possession key which could be cryptographically confirmed by the recipients’ proof of possession of the key by the presenter. Presenter being a ‘Holder of Key’ is described from Proof of possession of a key scenario[10].

```
+--------------+
|              |                         +--------------+
|              |--(3) Presentation of -->|              |
|              |      JWT w/ Encrypted   |              |
|  Presenter   |      PoP Key            |              |
|              |                         |              |
|              |<-(4) Communication ---->|              |
|              |      Authenticated by   |              |
+--------------+      PoP Key            |              |
  ^          ^                           |              |
  |          |                           |              |
 (1) Sym.   (2) JWT w/                   |  Recipient   |
  |  PoP     |  Encrypted                |              |
  |  Key     |  PoP Key                  |              |
  v          |                           |              |
+--------------+                         |              |
|              |                         |              |
|              |                         |              |
|              |<-(0) Key Exchange for ->|              |
|   Issuer     |      Key Encryption Key |              |
|              |                         |              |
|              |                         |              |
|              |                         +--------------+
+--------------+
```

*Figure 2.4-1 The Holder Of Key token flow[10]*

Security Assertion Makeup Language 2.0 (SAML 2.0) could be considered as another used technology as a Holder of Key Token.

There is no commercial implementations of the HOK yet. Hence, it is not a battle tested method.

# 3 Design of Netflix OSS Microservices

## 3.1 Netflix OSS Architecture and Product Components

The Netflix OSS adopted a Micro Services based SOA architecture. The main components and their responsibilities are as follows

| **Netflix OSS Component** | **Usage** |
|---|---|
| Netflix Eureka | Service Discovery Server |
| Netflix Ribbon | Dynamic Routing and Load Balancer |
| Netflix Hystrix | Circuit Breaker |
| Netflix Turbine | Microservice Monitoring |
| Netflix Zuul | Edge Server |
| Log Stash | Centralized Logging |
| Security Monkey | Monitor and secure Netflix OSS perimeter network |
| Scumblr | An Intelligence gathering tool about functionality of Netflix OSS Microservice eco system |

*Table 3.1-1 Netflix OSS Components*

Figure 3.1-1 depicts the Netflix OSS layered architecture

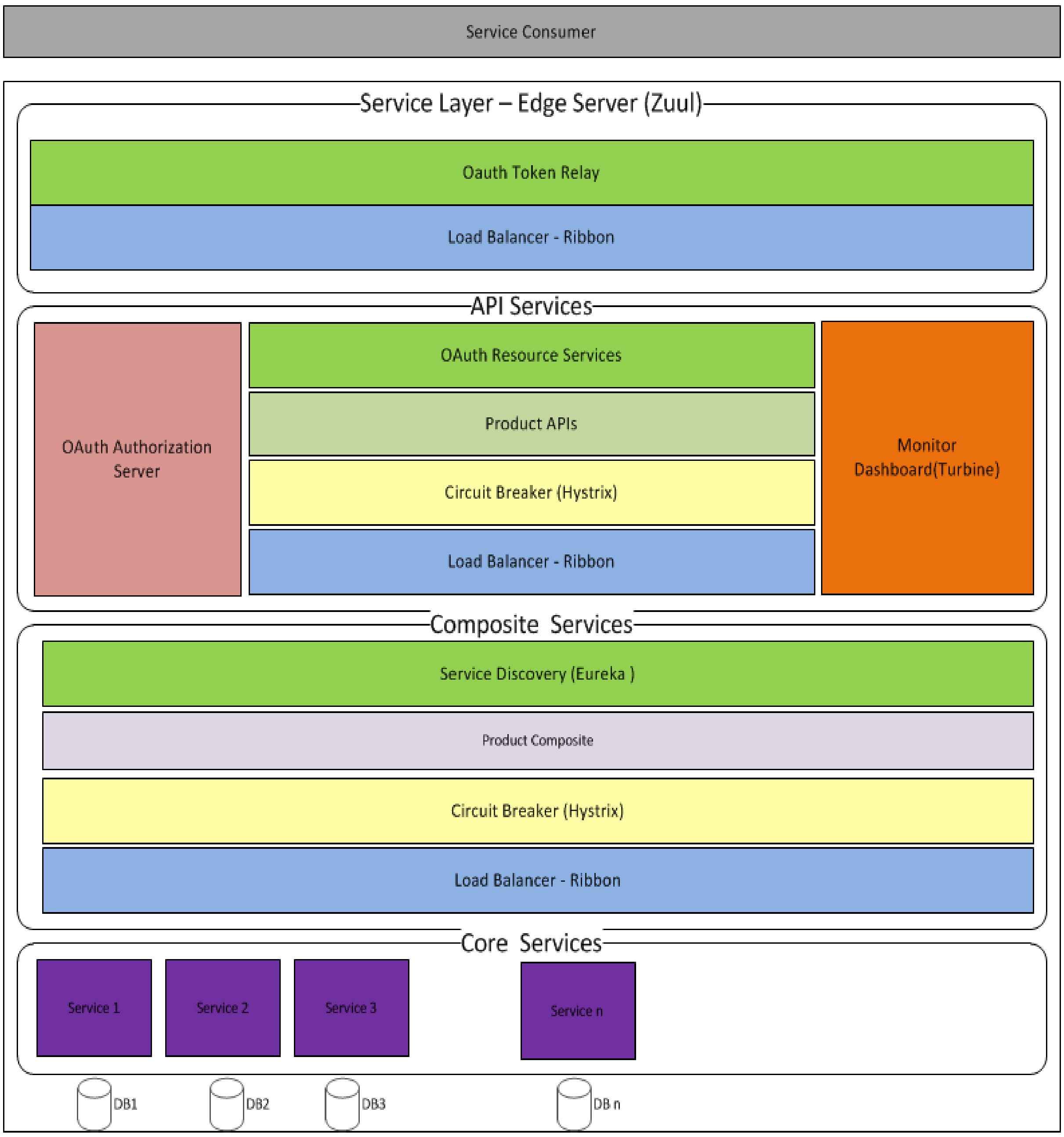


*Figure 3.1-1 Netflix OSS layered architecture*

The service consumer making a REST API service call to the service layer's Edge server. The Edge server is implemented using Java Spring MVC and contains RESTful API interface to integrate with external requests. The service layer Authenticate the request using the OAuth Authorization Server in the API Service layer.

The Ribbon load balancer manages the communication in between Microservices along with Eureka service discovery. The Hystrix circuit breaker is responsible for graceful failover when there is an error in a functionality of a Microservice.

The Composite service is responsible for amalgamate payloads returned from the Core service. Core services are considered as the resource end-points where it communicates with the data layer. The Turbine dashboard is using to monitor the Microservice ecosystem using its own Web interface.

Following section discusses the Netflix OSS components in detail.

## 3.2 Netflix OSS Components

This section illustrates important components of the Netflix OSS Microservice ecosystem.

### 3.2.1 Spring Cloud Framework

The Netflix OSS is built on top of the Spring Cloud framework. Spring Cloud framework delivers tools to quickly build some of the common patterns in distributed systems such as Circuit breakers, Configuration management , Service discovery , Control bus and leadership election.
The Spring cloud contains following two main components to support the Distributed application development

#### 3.2.1.1 Spring Cloud Configuration Server

The Spring Cloud Configuration Server enables to horizontal scalability of the framework with centralized configuration. The Java properties and YML files are used to represent the configuration. The Configuration Server merges these files into environment objects. These Configurations can be accessed as REST APIs and can be queried by any application directly to obtain configuration data.

#### 3.2.1.2 Spring Cloud Bus

The Spring Cloud Bus handles the technical management aspects of the application instances. The Advanced Message Queuing Protocol (AMQP ) is used for messaging and it is responsible for client side bindings as well. The pluggable architecture of SCB enables seamless communication of new Microservices which spawning on-the-fly.

The Netflix OSS components using wrappers for the Spring Cloud components. The Eureka discovery service, Ribbon load balancer, Hystrix Circuit breaker and The Zuul edge server are the main Netflix OSS components which are implemented as wrappers to the Spring cloud components.

### 3.2.2 Eureka

The Eureka implements the Service Discovery pattern [11]. It is using a service registry which is updating dynamically upon spawning of a new service.

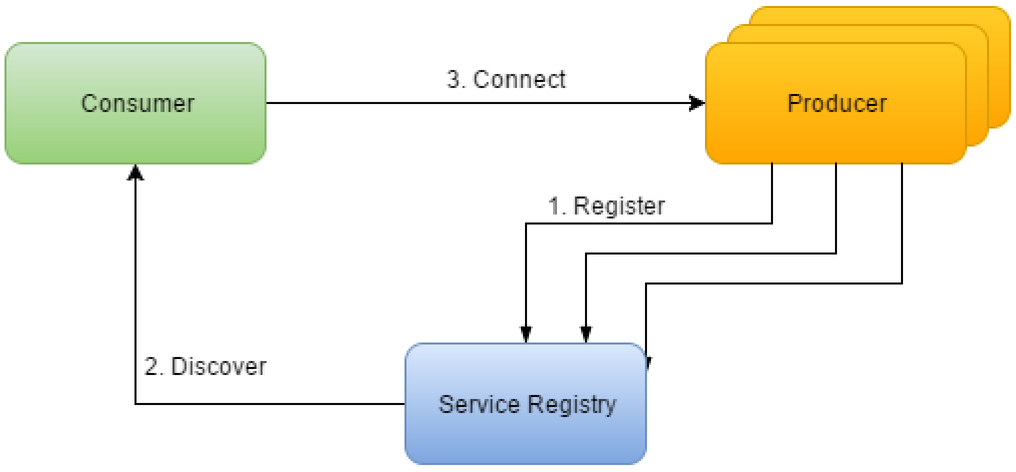
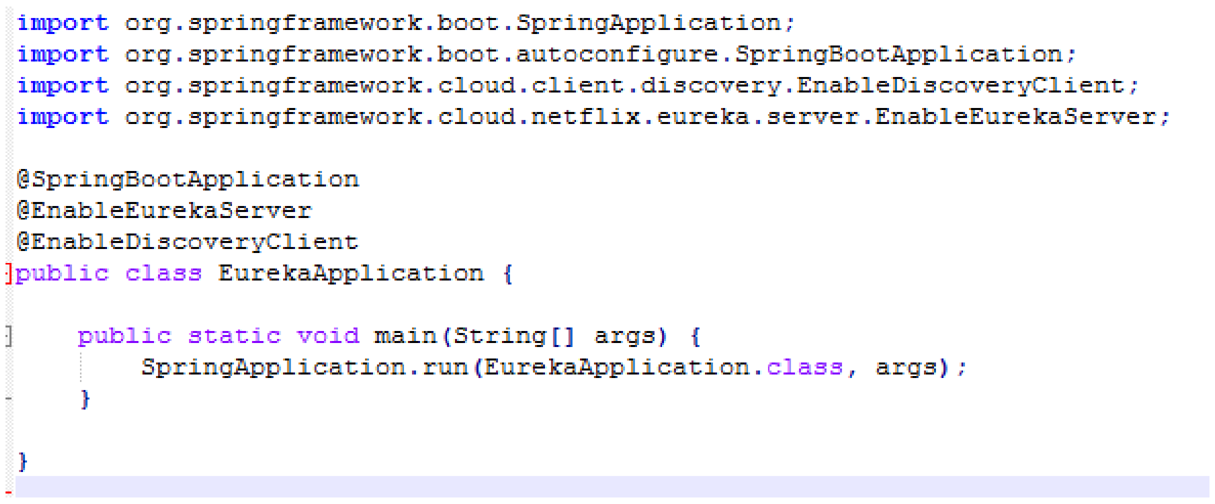


*Figure 3.2-1 Eureka Discovery Service*

Adding the spring-cloud-starter-eureka-server dependency to the Spring boot application enables the Eureka Server deployment to the application. The @EnableDiscoveryClient annotation enables client to probe the Eureka server by adding the server instance to the Configuration.



```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@SpringBootApplication
@EnableEurekaServer
@EnableDiscoveryClient
public class EurekaApplication {

    public static void main(String[] args) {
        SpringApplication.run(EurekaApplication.class, args);
    }

}
```

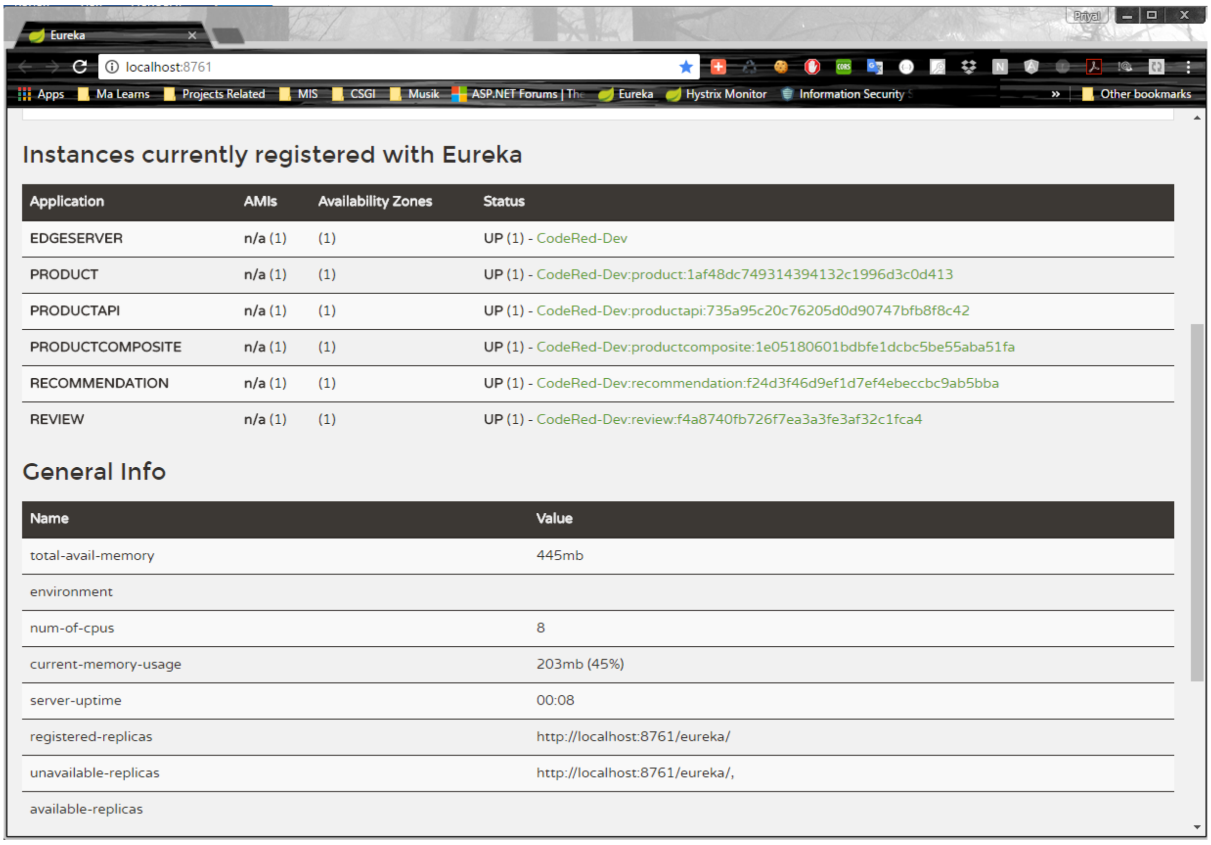
*Figure 3.2-2 Eureka Server implementation*

The clients can participate the service discovery by using the @DiscoveryClient annotation which includes in the spring-cloud-starter-eureka dependency. The Discovery client provides IP addresses, ports, and other relevant details about the service instances registered with Eureka by using the service's logical identifier.

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@SpringBootApplication
@EnableDiscoveryClient
public class RecommendationServiceApplication {

    public static void main(String[] args) {
        SpringApplication.run(RecommendationServiceApplication.class, args);
    }
}
```

*Figure 3.2-3 Eureka Discovery Client Implementation*

Upon successful deployment, the Eureka Dashboard provides the health status and other useful information about the Microservices eco system.  The main Dashboard displays registered instances with Eureka Discovery Service and server details.



Instances currently registered with Eureka

| Application | AMIs | Availability Zones | Status |
|---|---|---|---|
| EDGESERVER | n/a (1) | (1) | UP (1) - CodeRed-Dev |
| PRODUCT | n/a (1) | (1) | UP (1) - CodeRed-Dev:product:1af48dc749314394132c1996d3c0d413 |
| PRODUCTAPI | n/a (1) | (1) | UP (1) - CodeRed-Dev:productapi:735a95c20c76205d0d90747bfb8f8c42 |
| PRODUCTCOMPOSITE | n/a (1) | (1) | UP (1) - CodeRed-Dev:productcomposite:1e05180601bdbfe1dcbc5be55aba51fa |
| RECOMMENDATION | n/a (1) | (1) | UP (1) - CodeRed-Dev:recommendation:f24d3f46d9ef1d7ef4ebeccbc9ab5bba |
| REVIEW | n/a (1) | (1) | UP (1) - CodeRed-Dev:review:f4a8740fb726f7ea3a3fe3af32c1fca4 |

General Info

| Name | Value |
|---|---|
| total-avail-memory | 445mb |
| environment | |
| num-of-cpus | 8 |
| current-memory-usage | 203mb (45%) |
| server-uptime | 00:08 |
| registered-replicas | http://localhost:8761/eureka/ |
| unavailable-replicas | http://localhost:8761/eureka/, |
| available-replicas | |

*Figure 3.2-4 Eureka Service Monitor*

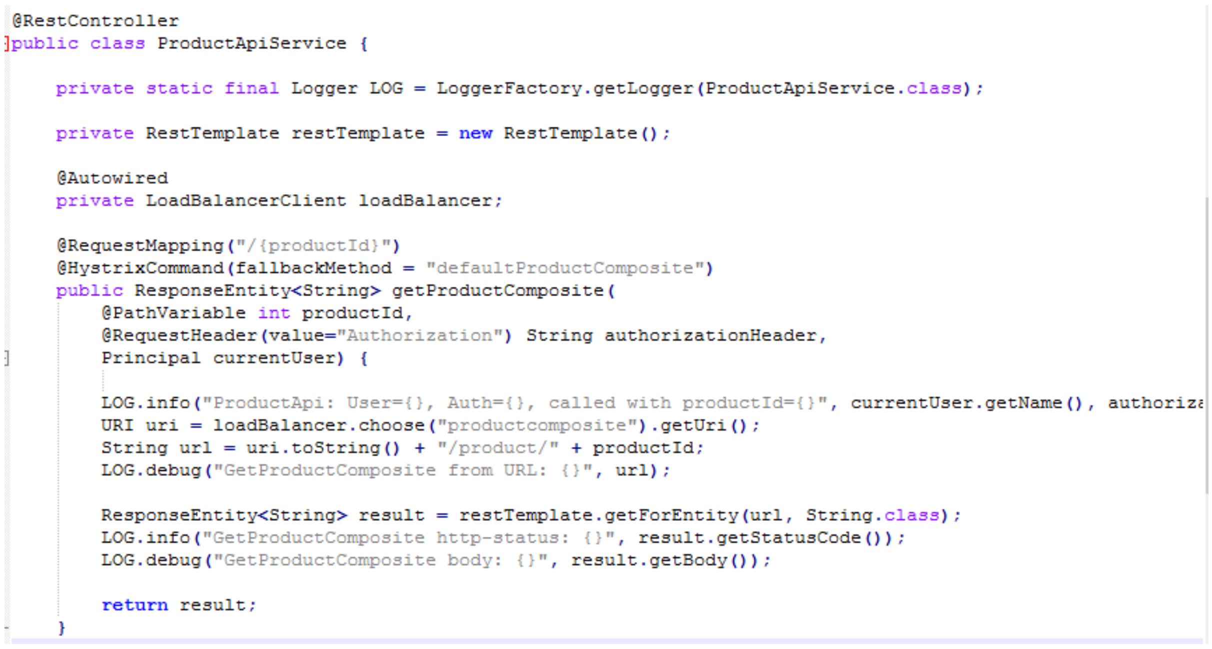
The Eureka service monitor is running on a predefined port (8761) and contains a Web interface. It is capable of displaying all the Microservices instances that are running in the Netflix Microservices ecosystem. General information such as Memory consumption of all Microservices, server uptime, Number of CPUs in the hosting environment and available memory.

User can navigate to the particular Microservice instance’s link in the Eureka monitoring tool to identify the respective port which that Microservice is running.

### 3.2.3 Ribbon Load Balancer

The Ribbon load balancer is populated with the dynamically spawned server list obtained from Eureka server. It provides a sophisticated client side IPC library with configurable load balancing and fault tolerance.

The Netflix OSS integration can be made by adding the spring-cloud-starter-ribbon dependency to the Spring Boot application. Ribbon is using client side load balancing by using the LoadBalancerClient in the client application as below.



```java
@RestController
public class ProductApiService {

    private static final Logger LOG = LoggerFactory.getLogger(ProductApiService.class);

    private RestTemplate restTemplate = new RestTemplate();

    @Autowired
    private LoadBalancerClient loadBalancer;

    @RequestMapping("/{productId}")
    @HystrixCommand(fallbackMethod = "defaultProductComposite")
    public ResponseEntity<String> getProductComposite(
        @PathVariable int productId,
        @RequestHeader(value="Authorization") String authorizationHeader,
        Principal currentUser) {

        LOG.info("ProductApi: User={}, Auth={}, called with productId={}", currentUser.getName(), authoriza
        URI uri = loadBalancer.choose("productcomposite").getUri();
        String url = uri.toString() + "/product/" + productId;
        LOG.debug("GetProductComposite from URL: {}", url);

        ResponseEntity<String> result = restTemplate.getForEntity(url, String.class);
        LOG.info("GetProductComposite http-status: {}", result.getStatusCode());
        LOG.debug("GetProductComposite body: {}", result.getBody());

        return result;
    }
```

*Figure 3.2-5 Ribbon Load Balancer Implementation*

Availability filtering and weighted response time are also featured in the Ribbon LB as additional load balancing algorithms.

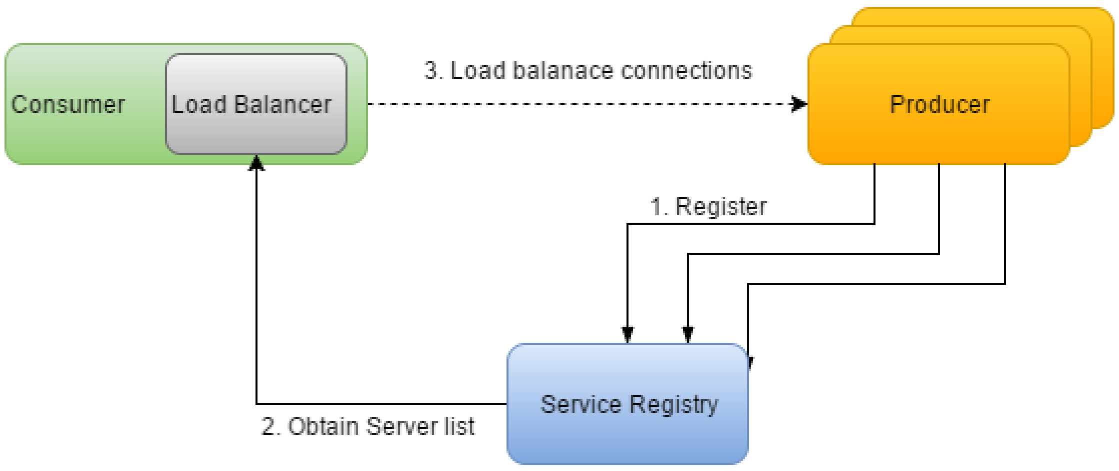


*Figure 3.2-6 Ribbon Client Side Load Balancing*

### 3.2.4 Hystrix Circuit Breaker

Hystrix implements the fault tolerance design pattern “Circuit Breaker “for distributed systems. Hystrix placed in-between the service and its remote dependency. Hystrix counts number of subsequent failed requests within a configurable time period and if the failure threshold is reached, the circuit is tripped to open. When the circuit status is open, calls are no longer made to the dependency and customized behaviour (Notification, Exception, returning null data or calling a different dependency) would take place. Hystrix using RabbitMQ as the distributed message queue to perform AMQP message passing.

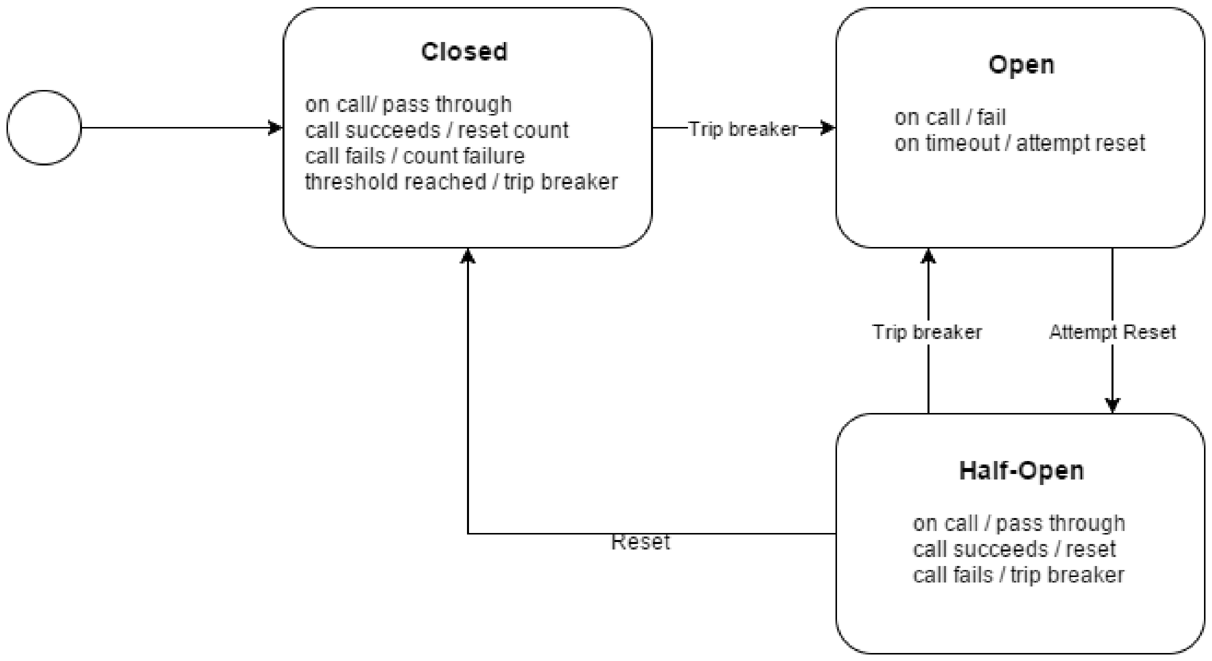


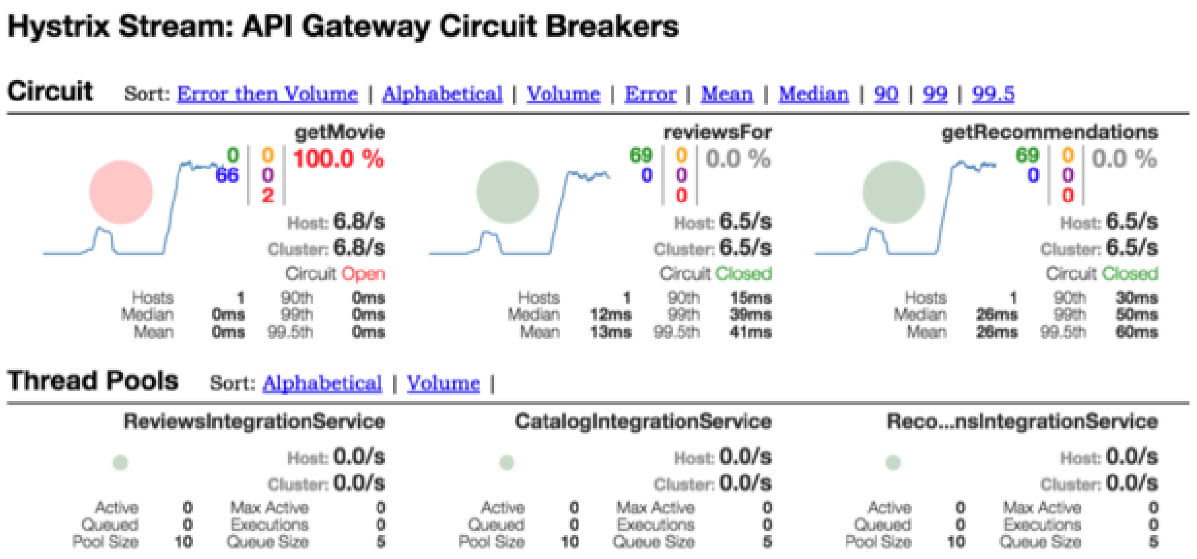
*Figure 3.2-7 Hystrix Circuit Breaker State Transition*

The state machine will transform into the “half open” state if the dependency is healthy again. Requests will be passing through to the dependency again and if succeeds , the state would be transformed to the closed state and the circuit is tripped to close.

The spring-cloud-starter-hystrix dependency contains the Hystrix implementation and annotating the @EnableCircuitBreaker and @HystrixCommand enables the circuit breaker in any spring bean method.

```
 @HystrixCommand(fallbackMethod = "defaultProduct")

  public ResponseEntity<Product> getProduct(int productId) {

      URI uri = util.getServiceUrl("product", "http://localhost:8081/product");
      String url = uri.toString() + "/product/" + productId;
      LOG.debug("GetProduct from URL: {}", url);

}
```

Request metering, number of failed, successful and short circuited request and response time histogram are important telemetry that Hystrix provides apart from its main functionality.



Hystrix Stream: API Gateway Circuit Breakers

Circuit Sort: Error then Volume | Alphabetical | Volume | Error | Mean | Median | 90 | 99 | 99.5

```
getMovie                          reviewsFor                        getRecommendations
0  | 0 | 100.0 %                  69 | 0 | 0.0 %                    69 | 0 | 0.0 %
66 | 0 |                          0  | 0 |                          0  | 0 |
   | 2 |                             | 0 |                             | 0 |
Host: 6.8/s                       Host: 6.5/s                       Host: 6.5/s
Cluster: 6.8/s                    Cluster: 6.5/s                    Cluster: 6.5/s
Circuit Open                      Circuit Closed                    Circuit Closed
Hosts   1    90th   0ms           Hosts   1     90th   15ms         Hosts   1     90th   30ms
Median  0ms  99th   0ms           Median  12ms  99th   39ms         Median  26ms  99th   50ms
Mean    0ms  99.5th 0ms           Mean    13ms  99.5th 41ms         Mean    26ms  99.5th 60ms
```

Thread Pools Sort: Alphabetical | Volume |

```
ReviewsIntegrationService          CatalogIntegrationService          Reco...nsIntegrationService
Host: 0.0/s                        Host: 0.0/s                        Host: 0.0/s
Cluster: 0.0/s                     Cluster: 0.0/s                     Cluster: 0.0/s
Active     0   Max Active  0       Active     0   Max Active  0       Active     0   Max Active  0
Queued     0   Executions  0       Queued     0   Executions  0       Queued     0   Executions  0
Pool Size 10   Queue Size  5       Pool Size 10   Queue Size  5       Pool Size 10   Queue Size  5
```

*Figure 3.2-8 Hystrix Dashboard*

## 3.2.5 Zuul Edge Service

Zuul acts a perimeter service in combination with Netfix OSS Ribbon and Hystrix to provide resilient API services where clients can access. Zuul implements the API gateway design pattern and it avoid need to manage CORS(cross-origin resource sharing). Authentication is handled independently for all back end support services.

The Zuul performs following functionality as the Edge service.

1. Dynamic Routing : Request are routed dynamically to back end services as needed.
2. Load Shedding
3. Static Reponses handling : If needed building responses independently without using of back end services.
4. Authentication and Security: using OAuth 2 tokens to secure edge services using an Identity server.
5. Monitoring

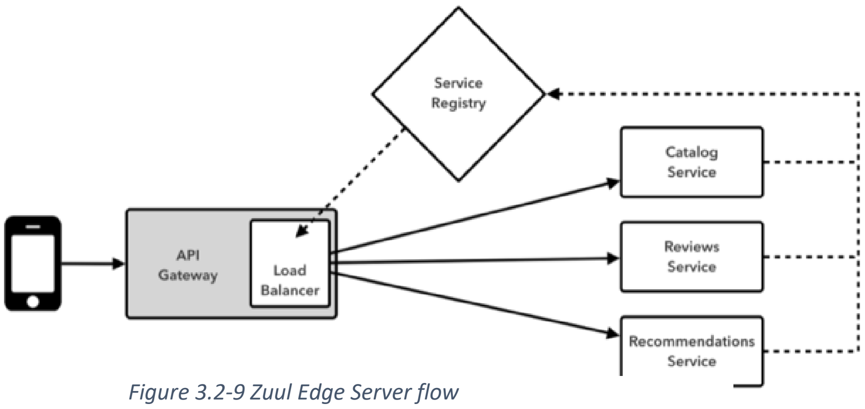


*Figure 3.2-9 Zuul Edge Server flow*

Zuul is using dynamically allocated ports and avoiding port conflicts to minimize administration.

### 3.2.6 Undertow

Undertow is the web server used by Netflix OSS to host its service APIs. Undertow is written in Java and operated in a very light weight manner. The undertow.jar file is less than 1MB and with a simple embedded server it uses less than 4MB of memory. Undertow’s composition based architecture allows to build a web server by combining small single purpose handlers [12]. It can be operated as fully fledged Java EE servlet container or low level non-blocking handler. Undertow provides support for the Web sockets as well.

## 3.3 Functionality of Netflix OSS Microservices

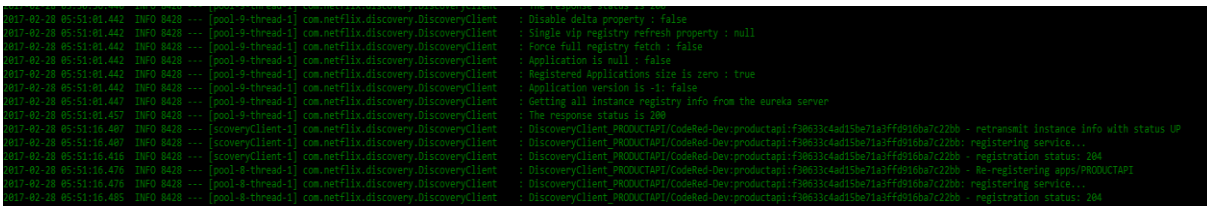
This section illustrates functionality of Netflix OSS Microservices such as spawning of Microservices and internal communication mechanism.

### 3.3.1 Spawning Microservices

Each of the Microservices can be spawned with spring bootrun command as follows.

```
start /D microservices\core\product-service          gradlew bootRun
```

The successful service spawning can be monitored using respective command windows.



```
2017-02-28 05:51:01.442  INFO 8428 --- [pool-9-thread-1] com.netflix.discovery.DiscoveryClient    : Disable delta property : false
2017-02-28 05:51:01.442  INFO 8428 --- [pool-9-thread-1] com.netflix.discovery.DiscoveryClient    : Single vip registry refresh property : null
2017-02-28 05:51:01.442  INFO 8428 --- [pool-9-thread-1] com.netflix.discovery.DiscoveryClient    : Force full registry fetch : false
2017-02-28 05:51:01.442  INFO 8428 --- [pool-9-thread-1] com.netflix.discovery.DiscoveryClient    : Application is null : false
2017-02-28 05:51:01.442  INFO 8428 --- [pool-9-thread-1] com.netflix.discovery.DiscoveryClient    : Registered Applications size is zero : true
2017-02-28 05:51:01.442  INFO 8428 --- [pool-9-thread-1] com.netflix.discovery.DiscoveryClient    : Application version is -1: false
2017-02-28 05:51:01.447  INFO 8428 --- [pool-9-thread-1] com.netflix.discovery.DiscoveryClient    : Getting all instance registry info from the eureka server
2017-02-28 05:51:01.457  INFO 8428 --- [pool-9-thread-1] com.netflix.discovery.DiscoveryClient    : The response status is 200
2017-02-28 05:51:16.407  INFO 8428 --- [scoveryClient-1] com.netflix.discovery.DiscoveryClient    : DiscoveryClient_PRODUCTAPI/CodeRed-Dev:productapi:f30633c4ad15be71a3ffd916ba7c22bb - retransmit instance info with status UP
2017-02-28 05:51:16.407  INFO 8428 --- [scoveryClient-1] com.netflix.discovery.DiscoveryClient    : DiscoveryClient_PRODUCTAPI/CodeRed-Dev:productapi:f30633c4ad15be71a3ffd916ba7c22bb: registering service...
2017-02-28 05:51:16.416  INFO 8428 --- [scoveryClient-1] com.netflix.discovery.DiscoveryClient    : DiscoveryClient_PRODUCTAPI/CodeRed-Dev:productapi:f30633c4ad15be71a3ffd916ba7c22bb - registration status: 204
2017-02-28 05:51:16.476  INFO 8428 --- [pool-8-thread-1] com.netflix.discovery.DiscoveryClient    : DiscoveryClient_PRODUCTAPI/CodeRed-Dev:productapi:f30633c4ad15be71a3ffd916ba7c22bb - Re-registering apps/PRODUCTAPI
2017-02-28 05:51:16.476  INFO 8428 --- [pool-8-thread-1] com.netflix.discovery.DiscoveryClient    : DiscoveryClient_PRODUCTAPI/CodeRed-Dev:productapi:f30633c4ad15be71a3ffd916ba7c22bb: registering service...
2017-02-28 05:51:16.485  INFO 8428 --- [pool-8-thread-1] com.netflix.discovery.DiscoveryClient    : DiscoveryClient_PRODUCTAPI/CodeRed-Dev:productapi:f30633c4ad15be71a3ffd916ba7c22bb - registration status: 204
```

Figure 3.3-1 Spawned Microservice

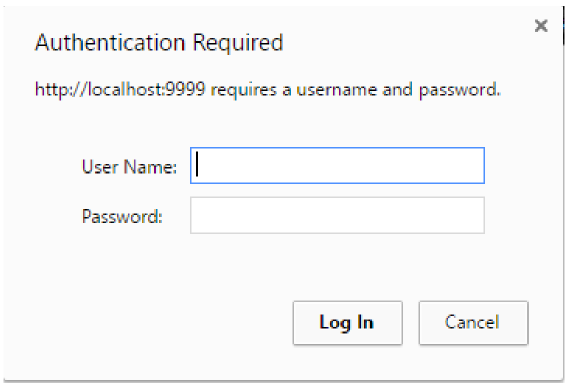
### 3.3.2 Edge Service Security

Netflix OSS  Edge service is secured using OAuth 2.0 access token . Adding dependencies spring-cloud-security and spring-security-oauth2. Adding @EnableAuthorizationServer enables the usage of the Auth server in the required application.

Appendix A2 contains the source code for the OAuth Authorization implementation.

The Sample Netflix OSS application using in-memory allowed grant flow, scopes , grant types and approved clients. This is a simulation of a OAuth Authorization server.

The Netflix OSS Demo code contains an in-memory authentication server. The user credentials are persisted in a Configuration file for simplicity.
The Authentication server request credentials in order to generate an OAuth token for the Edge server.


Authentication Required

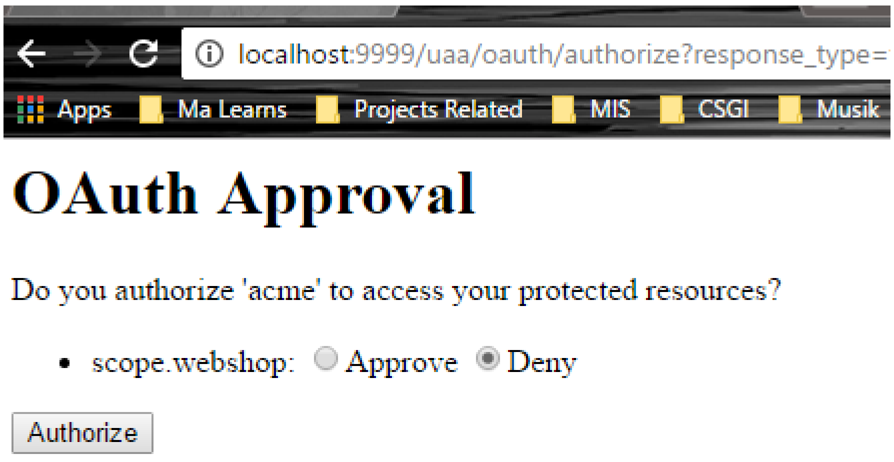
http://localhost:9999 requires a username and password.

User Name:

Password:

*Figure 3.3-2 Edge Server Authentication prompt*

Upon providing valid credentials, the Authentication server generates a consent to confirm by user.


localhost:9999/uaa/oauth/authorize?response_type=

OAuth Approval

Do you authorize 'acme' to access your protected resources?

- scope.webshop: Approve [✓ Deny]

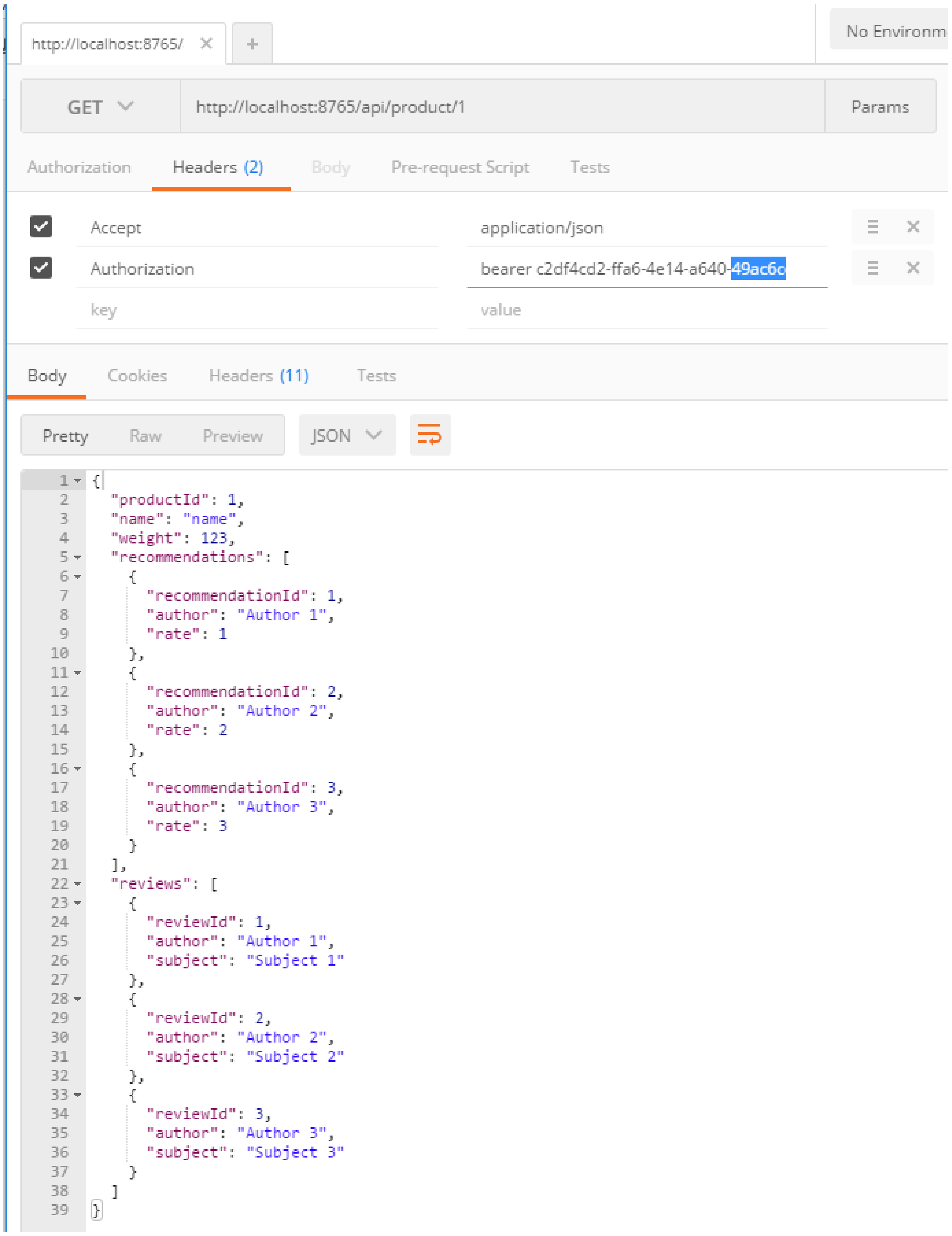
*Figure 3.3-3 Edge Server Authentication Consent*

Upon Approving, the Authorization server generates a JWT access token.

```
access_token=c2df4cd2-ffa6-4e14-a640-
49ac6ce3d231&token_type=bearer&state=48532&expires_in=43199
```

User can access Netflix OSS resources using the generated an access token. The resource access is demonstrated using the Chrome Postman tool. The access token is generated with expiration time, type and token state parameters.

The Netflix OSS resource access is demonstrated here using the Postman tool. The request requires HTTP headers “Accept” and “Authorization”. The generated bearer token by the Netflix OSS OAuth Identity server needs to pass-in as the Authorization HTTP header. The edger server using this bearer token to perform Authentication of the incoming request. The result is displayed in the Postmen tool’s response section.



http://localhost:8765/api/product/1

| | |
|---|---|
| Accept | application/json |
| Authorization | bearer c2df4cd2-ffa6-4e14-a640-49ac6c |
| key | value |

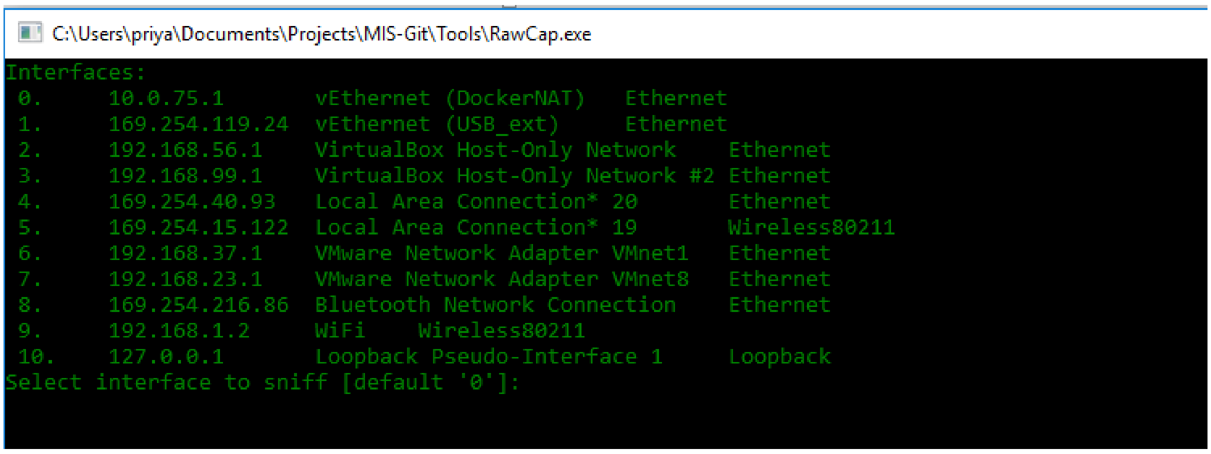
```
{
  "productId": 1,
  "name": "name",
  "weight": 123,
  "recommendations": [
    {
      "recommendationId": 1,
      "author": "Author 1",
      "rate": 1
    },
    {
      "recommendationId": 2,
      "author": "Author 2",
      "rate": 2
    },
    {
      "recommendationId": 3,
      "author": "Author 3",
      "rate": 3
    }
  ],
  "reviews": [
    {
      "reviewId": 1,
      "author": "Author 1",
      "subject": "Subject 1"
    },
    {
      "reviewId": 2,
      "author": "Author 2",
      "subject": "Subject 2"
    },
    {
      "reviewId": 3,
      "author": "Author 3",
      "subject": "Subject 3"
    }
  ]
}
```

*Figure 3.3-4 Netflix OSS Resource Access*

## 3.4 Security Issues in the current application

Even though, Netflix OSS is secured in the perimeter level using a JWT access token, internal security is not considered in the current implementation. Following section illustrates possible internal attacks that can be carried out to Netflix OSS Microservices.

The RAWCap, Wireshark and Fiddler are used to conduct these attacks to the internal Netflix OSS Microservices. Wireshark is not able to capture loopback traffic and hence, RAWCap is used to capture the loop back tcp port traffic. The generated pcap files are analyzed using the Wireshark tool.

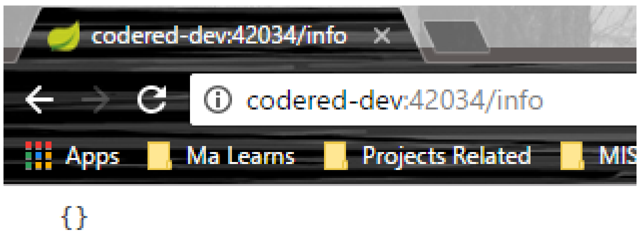


```
C:\Users\priya\Documents\Projects\MIS-Git\Tools\RawCap.exe
Interfaces:
 0.     10.0.75.1       vEthernet (DockerNAT)    Ethernet
 1.     169.254.119.24  vEthernet (USB_ext)      Ethernet
 2.     192.168.56.1    VirtualBox Host-Only Network    Ethernet
 3.     192.168.99.1    VirtualBox Host-Only Network #2 Ethernet
 4.     169.254.40.93   Local Area Connection* 20       Ethernet
 5.     169.254.15.122  Local Area Connection* 19       Wireless80211
 6.     192.168.37.1    VMware Network Adapter VMnet1   Ethernet
 7.     192.168.23.1    VMware Network Adapter VMnet8   Ethernet
 8.     169.254.216.86  Bluetooth Network Connection    Ethernet
 9.     192.168.1.2     WiFi    Wireless80211
 10.    127.0.0.1       Loopback Pseudo-Interface 1     Loopback
Select interface to sniff [default '0']:
```

*Figure 3.4-1 RAWCap Interface*

### 3.4.1 Attack 1 : Eavesdropping

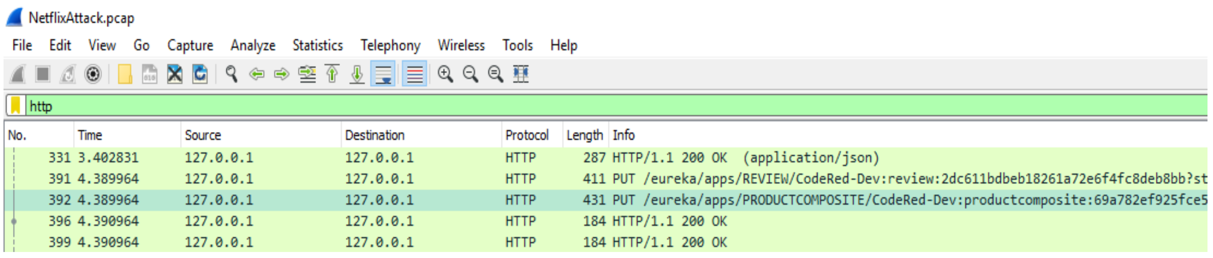
The attack is conducted against the Composite service Microservice. The Composite service Microservice is spawned in the port 42034 during the testing attack phase.



```
{}
```

*Figure 3.4-2 Composite Service Microservice*

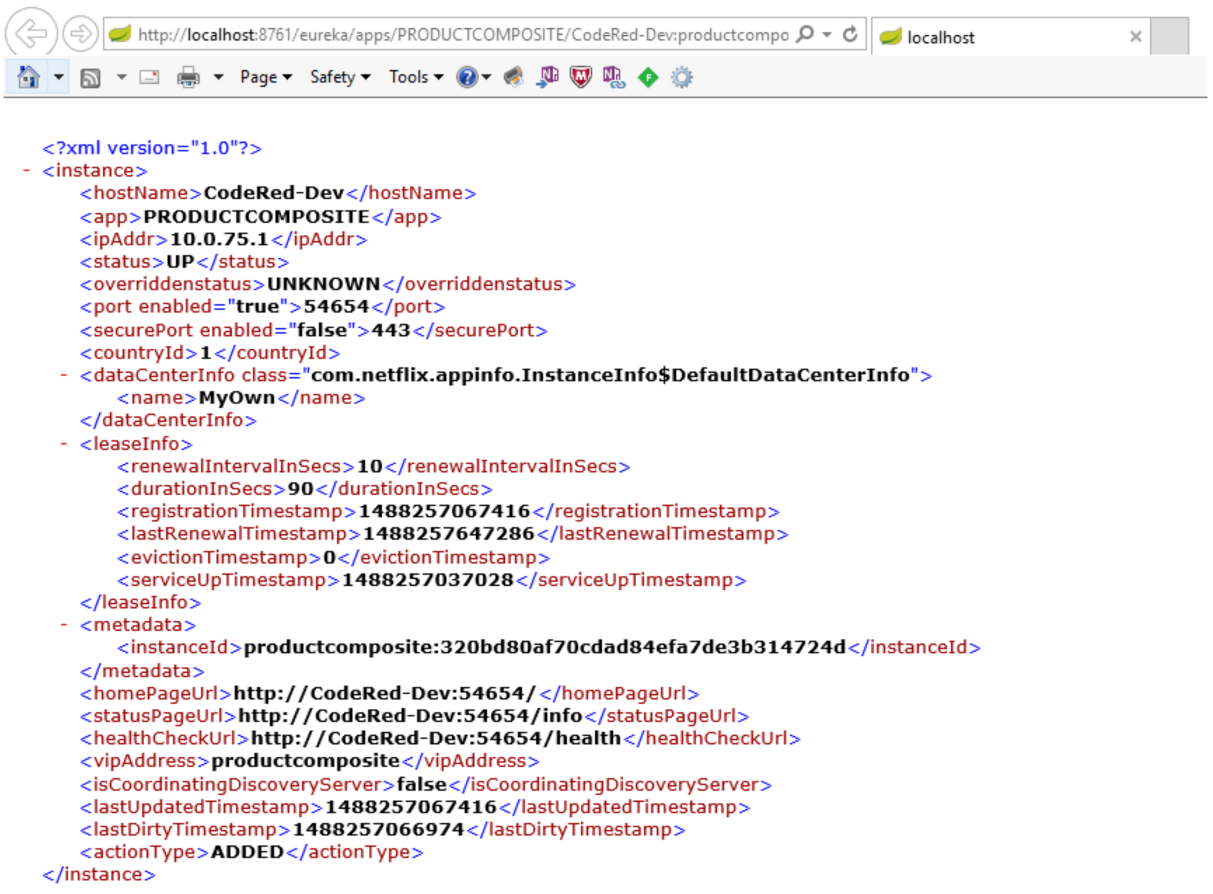
The captured packets are analysed using Wireshark as illustrated below.


NetflixAttack.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
http

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 331 | 3.402831 | 127.0.0.1 | 127.0.0.1 | HTTP | 287 | HTTP/1.1 200 OK (application/json) |
| 391 | 4.389964 | 127.0.0.1 | 127.0.0.1 | HTTP | 411 | PUT /eureka/apps/REVIEW/CodeRed-Dev:review:2dc611bdbeb18261a72e6f4fc8deb8bb?st |
| 392 | 4.389964 | 127.0.0.1 | 127.0.0.1 | HTTP | 431 | PUT /eureka/apps/PRODUCTCOMPOSITE/CodeRed-Dev:productcomposite:69a782ef925fce5 |
| 396 | 4.390964 | 127.0.0.1 | 127.0.0.1 | HTTP | 184 | HTTP/1.1 200 OK |
| 399 | 4.390964 | 127.0.0.1 | 127.0.0.1 | HTTP | 184 | HTTP/1.1 200 OK |

*Figure 3.4-3 Wireshark Packet capturing*

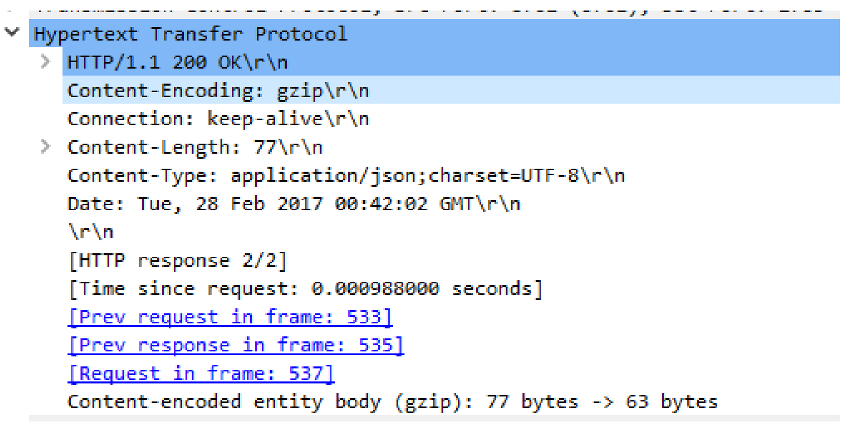
The Wireshark tool is used to analyse the the captured traffic details in-between Microservices. The http filter is used in the Wireshark to filter out required HTTP packets. The above diagram depicts the identification of PUT request  by RabbitMQ messaging service.


http://localhost:8761/eureka/apps/PRODUCTCOMPOSITE/CodeRed-Dev:productcompo
localhost
Page Safety Tools

```
<?xml version="1.0"?>
- <instance>
     <hostName>CodeRed-Dev</hostName>
     <app>PRODUCTCOMPOSITE</app>
     <ipAddr>10.0.75.1</ipAddr>
     <status>UP</status>
     <overriddenstatus>UNKNOWN</overriddenstatus>
     <port enabled="true">54654</port>
     <securePort enabled="false">443</securePort>
     <countryId>1</countryId>
   - <dataCenterInfo class="com.netflix.appinfo.InstanceInfo$DefaultDataCenterInfo">
         <name>MyOwn</name>
     </dataCenterInfo>
   - <leaseInfo>
         <renewalIntervalInSecs>10</renewalIntervalInSecs>
         <durationInSecs>90</durationInSecs>
         <registrationTimestamp>1488257067416</registrationTimestamp>
         <lastRenewalTimestamp>1488257647286</lastRenewalTimestamp>
         <evictionTimestamp>0</evictionTimestamp>
         <serviceUpTimestamp>1488257037028</serviceUpTimestamp>
     </leaseInfo>
   - <metadata>
         <instanceId>productcomposite:320bd80af70cdad84efa7de3b314724d</instanceId>
     </metadata>
     <homePageUrl>http://CodeRed-Dev:54654/</homePageUrl>
     <statusPageUrl>http://CodeRed-Dev:54654/info</statusPageUrl>
     <healthCheckUrl>http://CodeRed-Dev:54654/health</healthCheckUrl>
     <vipAddress>productcomposite</vipAddress>
     <isCoordinatingDiscoveryServer>false</isCoordinatingDiscoveryServer>
     <lastUpdatedTimestamp>1488257067416</lastUpdatedTimestamp>
     <lastDirtyTimestamp>1488257066974</lastDirtyTimestamp>
     <actionType>ADDED</actionType>
</instance>
```

*Figure 3.4-4 Product Composite Request*

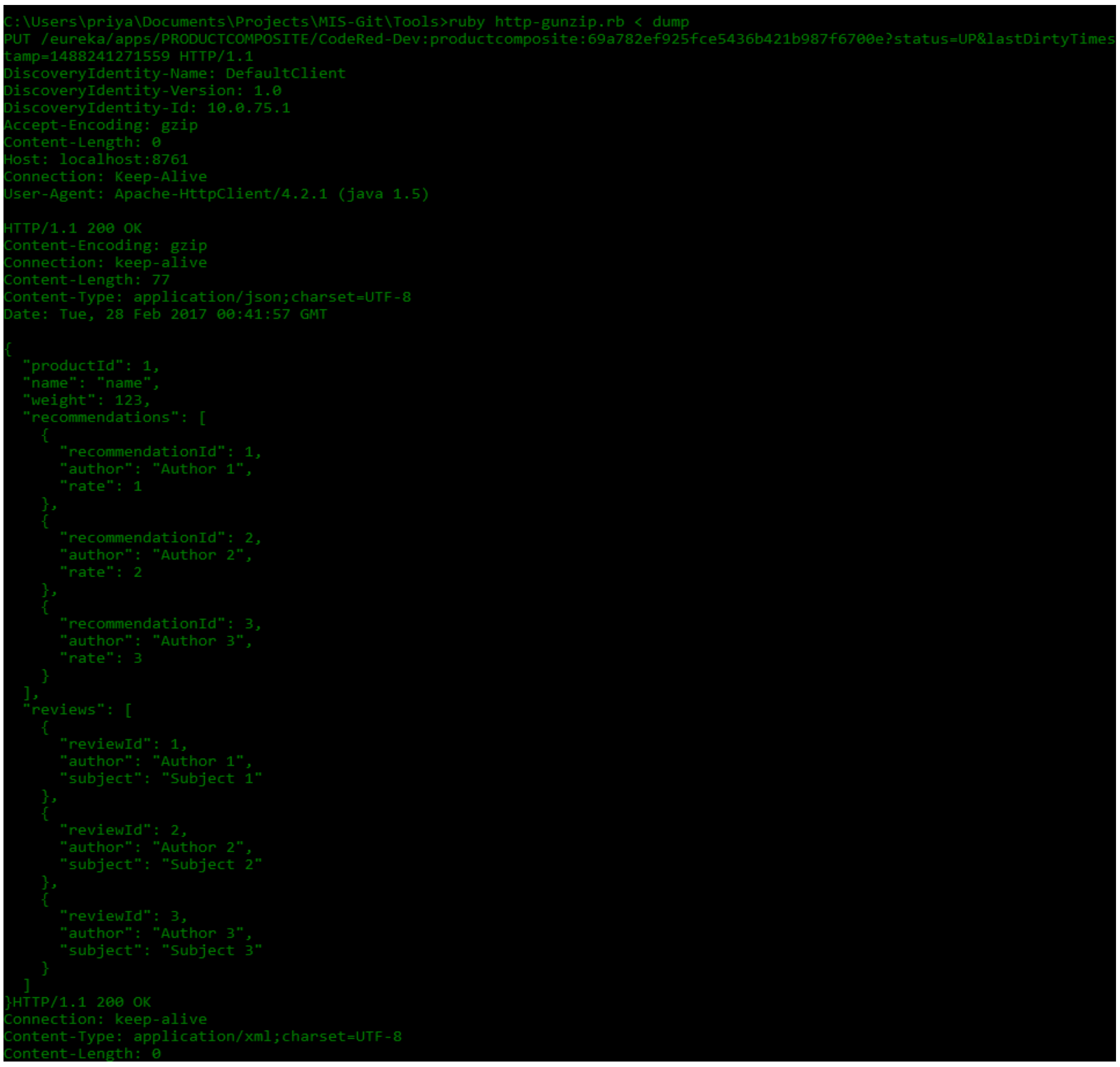
The request to the Product composite service can be identified as in the figure 3.4-3.  The most important values in this analyse is identification of the IP and the Port that  the respective service  is running.



```
Hypertext Transfer Protocol
  > HTTP/1.1 200 OK\r\n
    Content-Encoding: gzip\r\n
    Connection: keep-alive\r\n
  > Content-Length: 77\r\n
    Content-Type: application/json;charset=UTF-8\r\n
    Date: Tue, 28 Feb 2017 00:42:02 GMT\r\n
    \r\n
    [HTTP response 2/2]
    [Time since request: 0.000988000 seconds]
    [Prev request in frame: 533]
    [Prev response in frame: 535]
    [Request in frame: 537]
    Content-encoded entity body (gzip): 77 bytes -> 63 bytes
```

*Figure 3.4-5 Json Response body in Wireshark*

The Json content in the Response body is encoded using Gzip as depeicted in the figure 3.4-5 (in line 3: content encoding). The Wireshark do not have the capabilities to encode the Gzip encoding. Hence the decoding tool HTTP Gunzip [17] is used to decode the response Json content.



```
C:\Users\priya\Documents\Projects\MIS-Git\Tools>ruby http-gunzip.rb < dump
PUT /eureka/apps/PRODUCTCOMPOSITE/CodeRed-Dev:productcomposite:69a782ef925fce5436b421b987f6700e?status=UP&lastDirtyTimes
tamp=1488241271559 HTTP/1.1
DiscoveryIdentity-Name: DefaultClient
DiscoveryIdentity-Version: 1.0
DiscoveryIdentity-Id: 10.0.75.1
Accept-Encoding: gzip
Content-Length: 0
Host: localhost:8761
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.2.1 (java 1.5)

HTTP/1.1 200 OK
Content-Encoding: gzip
Connection: keep-alive
Content-Length: 77
Content-Type: application/json;charset=UTF-8
Date: Tue, 28 Feb 2017 00:41:57 GMT

{
  "productId": 1,
  "name": "name",
  "weight": 123,
  "recommendations": [
    {
      "recommendationId": 1,
      "author": "Author 1",
      "rate": 1
    },
    {
      "recommendationId": 2,
      "author": "Author 2",
      "rate": 2
    },
    {
      "recommendationId": 3,
      "author": "Author 3",
      "rate": 3
    }
  ],
  "reviews": [
    {
      "reviewId": 1,
      "author": "Author 1",
      "subject": "Subject 1"
    },
    {
      "reviewId": 2,
      "author": "Author 2",
      "subject": "Subject 2"
    },
    {
      "reviewId": 3,
      "author": "Author 3",
      "subject": "Subject 3"
    }
  ]
}HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: application/xml;charset=UTF-8
Content-Length: 0
```

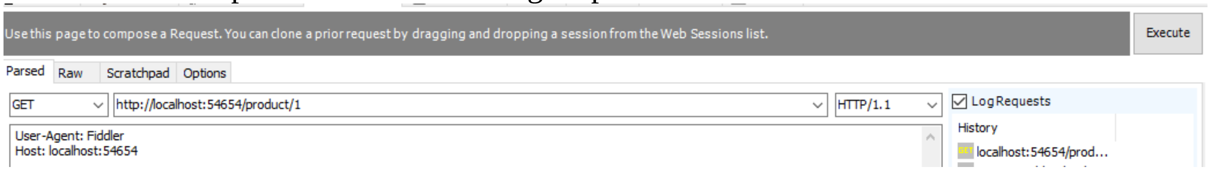
*Figure 3.4-6 Eavesdropping Response content*

An eavesdropping attacker can listen to the internal Microservice communication using this method. The figure 3.4-6 illustrates that how can an attacker see the response from the Composite service Microservice.

### 3.4.2 Attack 2 : Confused Deputy Attack

The confused deputy attack can be conducted using Fiddler tool. The Attacker can call the internal Product Composite service directly bypassing the Edge server as illustrated below.

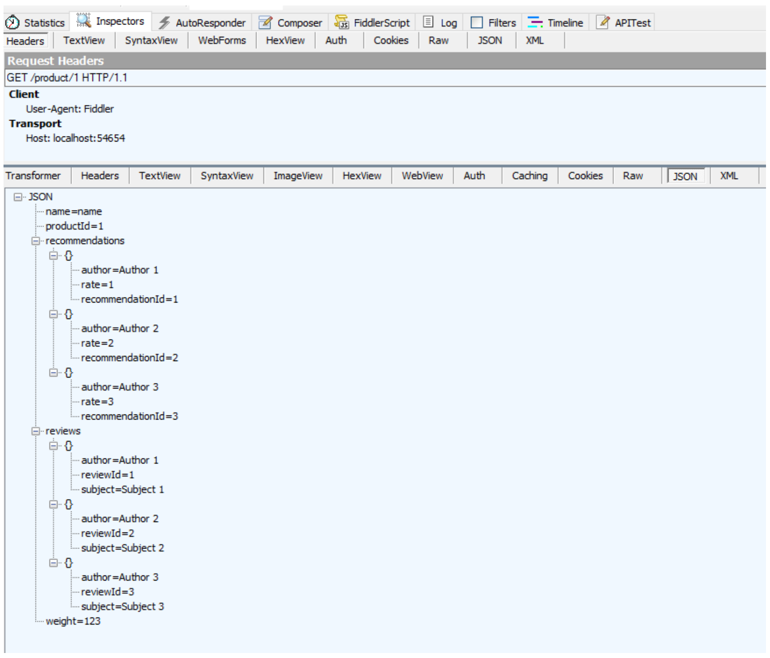
The Product Composite service is running on port 54654.


Use this page to compose a Request. You can clone a prior request by dragging and dropping a session from the Web Sessions list.
Execute
Parsed
Raw
Scratchpad
Options
Log Requests
History

```
GET    http://localhost:54654/product/1    HTTP/1.1
User-Agent: Fiddler
Host: localhost:54654
```

Figure 3.4-7 Fiddler Request to Microservice

The response from the attacked Microservice can be analysed using Fiddler. As depicts in the Figure 3.4-7, the called Microservice do not know who made the request to the Microservice API because it do not contain any Authentication or Authorization mechanism. Hence, the called Microservice responses the required output to any caller.



```
GET /product/1 HTTP/1.1
Client
    User-Agent: Fiddler
Transport
    Host: localhost:54654
```

```
JSON
  name=name
  productId=1
  recommendations
    {}
      author=Author 1
      rate=1
      recommendationId=1
    {}
      author=Author 2
      rate=2
      recommendationId=2
    {}
      author=Author 3
      rate=3
      recommendationId=3
  reviews
    {}
      author=Author 1
      reviewId=1
      subject=Subject 1
    {}
      author=Author 2
      reviewId=2
      subject=Subject 2
    {}
      author=Author 3
      reviewId=3
      subject=Subject 3
  weight=123
```

Figure 3.4-8 Fiddler Response analysing

### 3.4.3 Attack 3 : Man in the Middle Attack

An attacker can create a counterfeit Microservice and spawn in to the Microservice environment. The Adversary can alter the response and perform request / response logging. This can be achieved because there is no Authentication mechanism for internal Microservice calls.

The code in Appendix A1 illustrates creation of forged Composite service. The code is to create a malicious Microservice and send request to Product composite service. The called Microservice do not have any mechanism to Authenticate this malicious Microservice and the called service responding as normally.

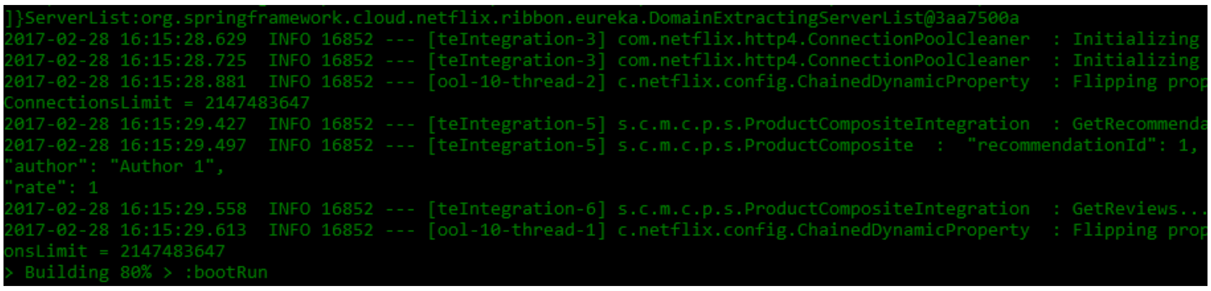
The malicious Microservice code also depicts how to log the incoming traffic and how to alter the incoming payload and respond back to the upper level Microservice which calling the malicious Microservice.

This counterfeit service cab Spawned to the Microservices ecosystem using following command.

```
start /D microservices\composite\ CounterfeitCompositeService    gradlew bootRun
```

The log results can be viewed as follows.

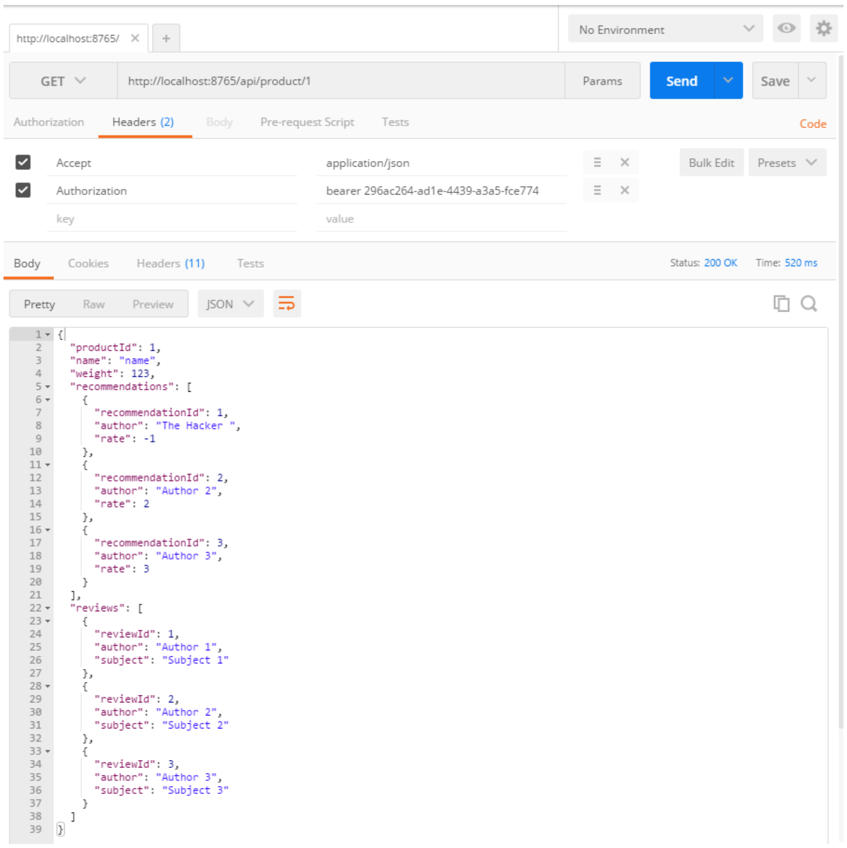
+



```
]}ServerList:org.springframework.cloud.netflix.ribbon.eureka.DomainExtractingServerList@3aa7500a
2017-02-28 16:15:28.629  INFO 16852 --- [teIntegration-3] com.netflix.http4.ConnectionPoolCleaner   : Initializing
2017-02-28 16:15:28.725  INFO 16852 --- [teIntegration-3] com.netflix.http4.ConnectionPoolCleaner   : Initializing
2017-02-28 16:15:28.881  INFO 16852 --- [ool-10-thread-2] c.netflix.config.ChainedDynamicProperty   : Flipping prop
ConnectionsLimit = 2147483647
2017-02-28 16:15:29.427  INFO 16852 --- [teIntegration-5] s.c.m.c.p.s.ProductCompositeIntegration   : GetRecommenda
2017-02-28 16:15:29.497  INFO 16852 --- [teIntegration-5] s.c.m.c.p.s.ProductComposite  :  "recommendationId": 1,
"author": "Author 1",
"rate": 1
2017-02-28 16:15:29.558  INFO 16852 --- [teIntegration-6] s.c.m.c.p.s.ProductCompositeIntegration   : GetReviews...
2017-02-28 16:15:29.613  INFO 16852 --- [ool-10-thread-1] c.netflix.config.ChainedDynamicProperty   : Flipping prop
onsLimit = 2147483647
> Building 80% > :bootRun
```

*Figure 3.4-9 Man in the Middle attack : Log results*

The Altered result can be viewed as follows



GET http://localhost:8765/api/product/1

| | |
|---|---|
| Accept | application/json |
| Authorization | bearer 296ac264-ad1e-4439-a3a5-fce774 |
| key | value |

Status: 200 OK Time: 520 ms

```
{
  "productId": 1,
  "name": "name",
  "weight": 123,
  "recommendations": [
    {
      "recommendationId": 1,
      "author": "The Hacker ",
      "rate": -1
    },
    {
      "recommendationId": 2,
      "author": "Author 2",
      "rate": 2
    },
    {
      "recommendationId": 3,
      "author": "Author 3",
      "rate": 3
    }
  ],
  "reviews": [
    {
      "reviewId": 1,
      "author": "Author 1",
      "subject": "Subject 1"
    },
    {
      "reviewId": 2,
      "author": "Author 2",
      "subject": "Subject 2"
    },
    {
      "reviewId": 3,
      "author": "Author 3",
      "subject": "Subject 3"
    }
  ]
}
```

*Figure 3.4-10 Man in The Middle Attack : Resource Results*

# 4 Design and Implementation of Solution

This chapter illustrates design and implementation of selected methods in-order to protect Netflix OSS Microservices from internal attacks.

## 4.1 Extending the Edge server’s Authorization OAuth 2 JWT Token to Internal Micro Services

The Edge server is Authorized using an OAuth 2 JWT access token. It is possible to secure internal micro services using the same incoming OAuth 2 token. Token relay pattern is using to perform this operation[18]. In Token Relay pattern,the OAuth2 consumer acts as a Client and forwards the incoming token to outgoing resource requests. The Internal Micro Services are communicating with the OAuth identity server to perform Authorization of the incoming JWT. The End user would be authorized against each Micro Service call.  The edge server would not terminate the JWT but propagate through the internal service structure using the token replay pattern.

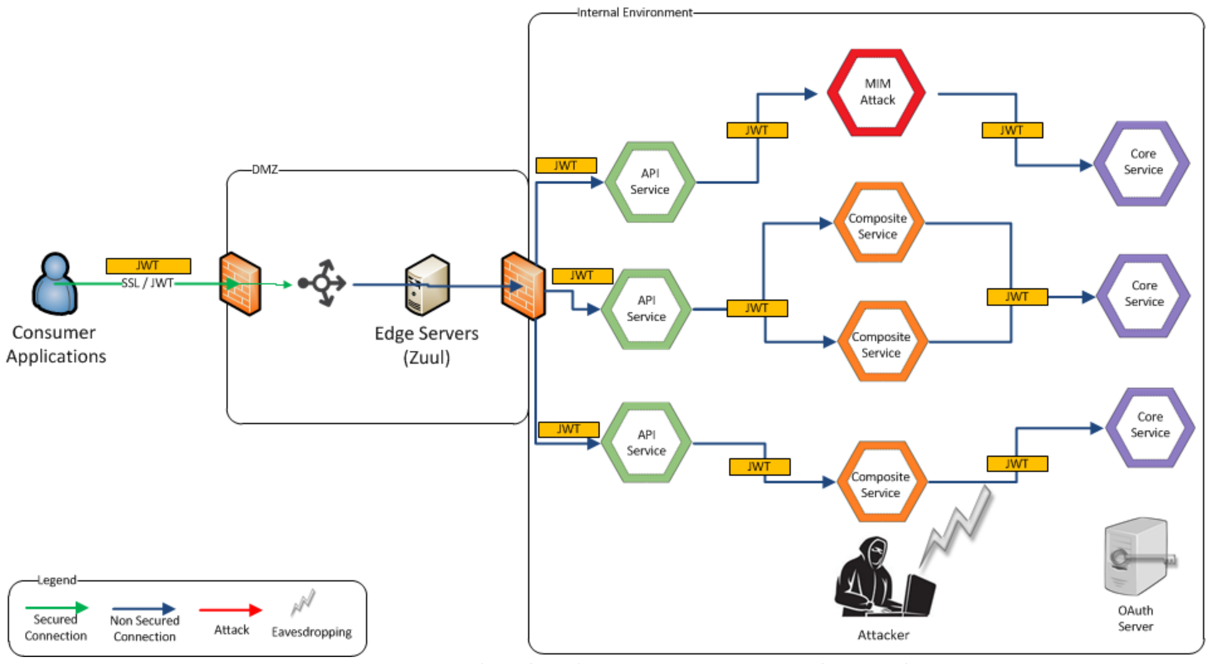
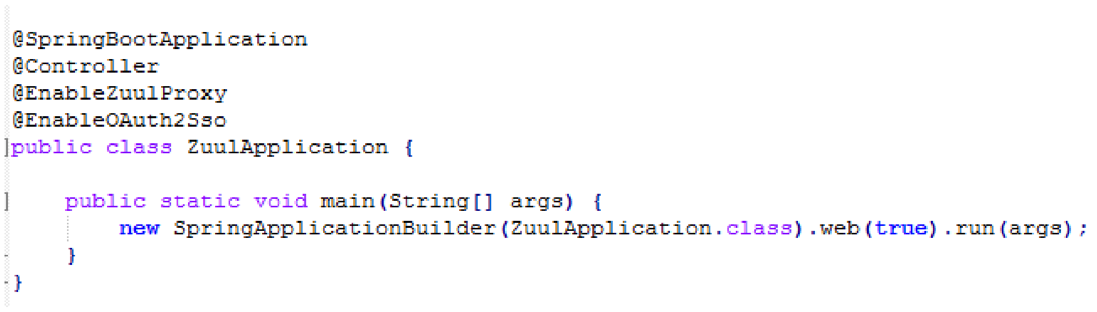


*Figure 4.1-1 Extending the Edge Server JWT to Internal Network*

The @EnableZuulProxy and @EnableOAuth2Resource annotation will enable this functionality in the calling(client) and called(Server) Micro Services respectively.



```
@SpringBootApplication
@Controller
@EnableZuulProxy
@EnableOAuth2Sso
public class ZuulApplication {

    public static void main(String[] args) {
        new SpringApplicationBuilder(ZuulApplication.class).web(true).run(args);
    }
}
```

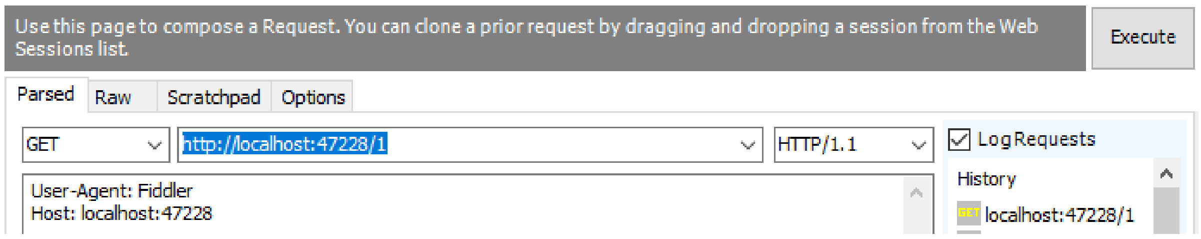
*Figure 4.1-2 Enable the Zuul server with Token Relay*

The @EnableOAuth2Sso annotation shipped in the spring-cloud-starter-security package. The spring-cloud-starter-security package triggers some autoconfiguration for a ZuulFilter. The filter just extracts an access token from the currently authenticated user, and puts it in a request header for the downstream requests.

The Product API service accepts the incoming relayed token from the Edge server using following code segment depicted in the appendix A3.

If an Adversary is trying to call the Product API service (running on port 47228), bypassing the edge server, the API service returns the 401-Unauthorized HTTP message as follows.

The Adversary is trying to access the Product API service using Fiddler.



Use this page to compose a Request. You can clone a prior request by dragging and dropping a session from the Web Sessions list.

Execute

Parsed | Raw | Scratchpad | Options

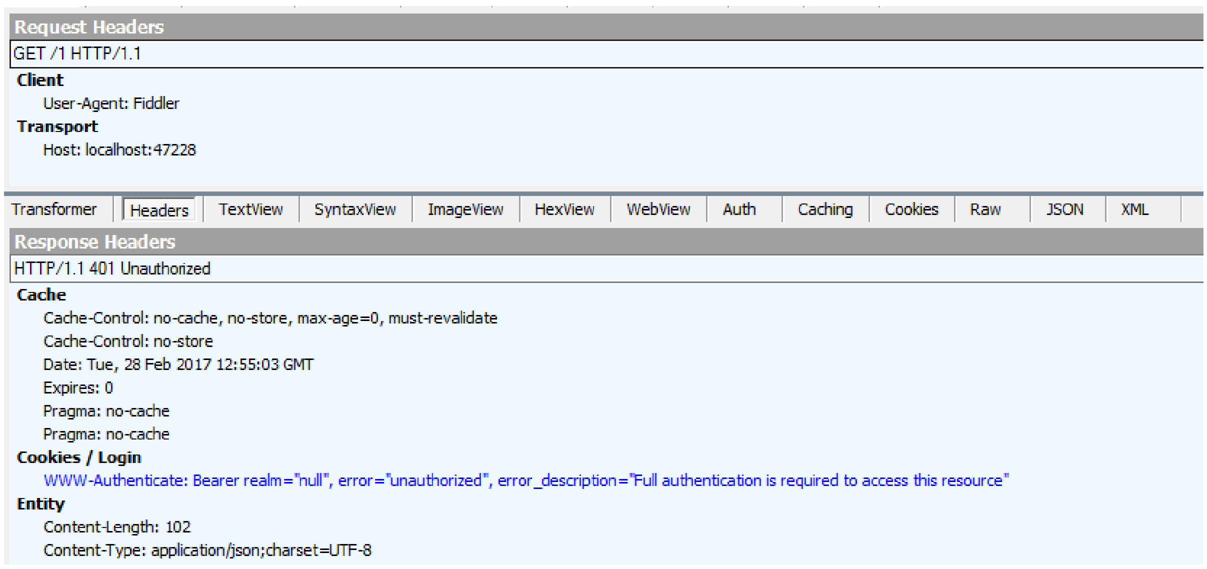
GET | http://localhost:47228/1 | HTTP/1.1 | ☑ Log Requests

```
User-Agent: Fiddler
Host: localhost:47228
```

History
GET localhost:47228/1

*Figure 4.1-3 Fiddler Request*

The response returns the 401- Unauthorized error as in the figure 4.1-4. The Fiddler tool is used to analyse the response from the Microservice.



```
Request Headers
GET /1 HTTP/1.1
Client
    User-Agent: Fiddler
Transport
    Host: localhost:47228
```

Transformer | Headers | TextView | SyntaxView | ImageView | HexView | WebView | Auth | Caching | Cookies | Raw | JSON | XML

```
Response Headers
HTTP/1.1 401 Unauthorized
Cache
    Cache-Control: no-cache, no-store, max-age=0, must-revalidate
    Cache-Control: no-store
    Date: Tue, 28 Feb 2017 12:55:03 GMT
    Expires: 0
    Pragma: no-cache
    Pragma: no-cache
Cookies / Login
    WWW-Authenticate: Bearer realm="null", error="unauthorized", error_description="Full authentication is required to access this resource"
Entity
    Content-Length: 102
    Content-Type: application/json;charset=UTF-8
```

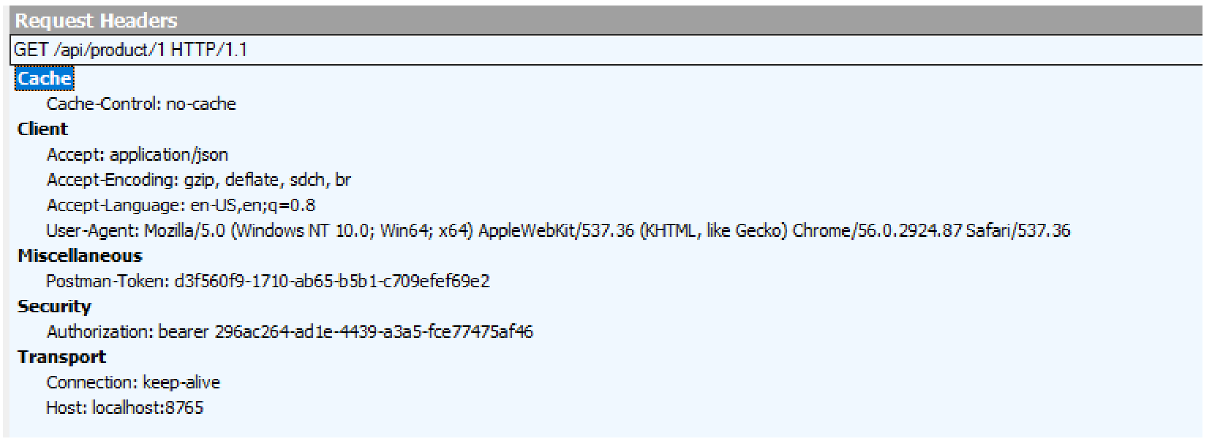
*Figure 4.1-4 401-Unauthorized Response*

The main advantage of this method is ease of Deployment. It required only to implement the Authorization mechanism in each service to validate the incoming JWT token. This method enforces less stress on internal micro services because no need of many external calls and validations.
The Token relay pattern is vulnerable to many attack vectors as follows.

1. Attacker can capture a token and propagate forward with own payload
2. No mechanism to identify which service is calling the destination service
3. No security of payload , hence can easily eavesdrop the payload
4. Cannot prevent MIM attack

### 4.1.1 Token Relay attack : Directly access internal services using captured token

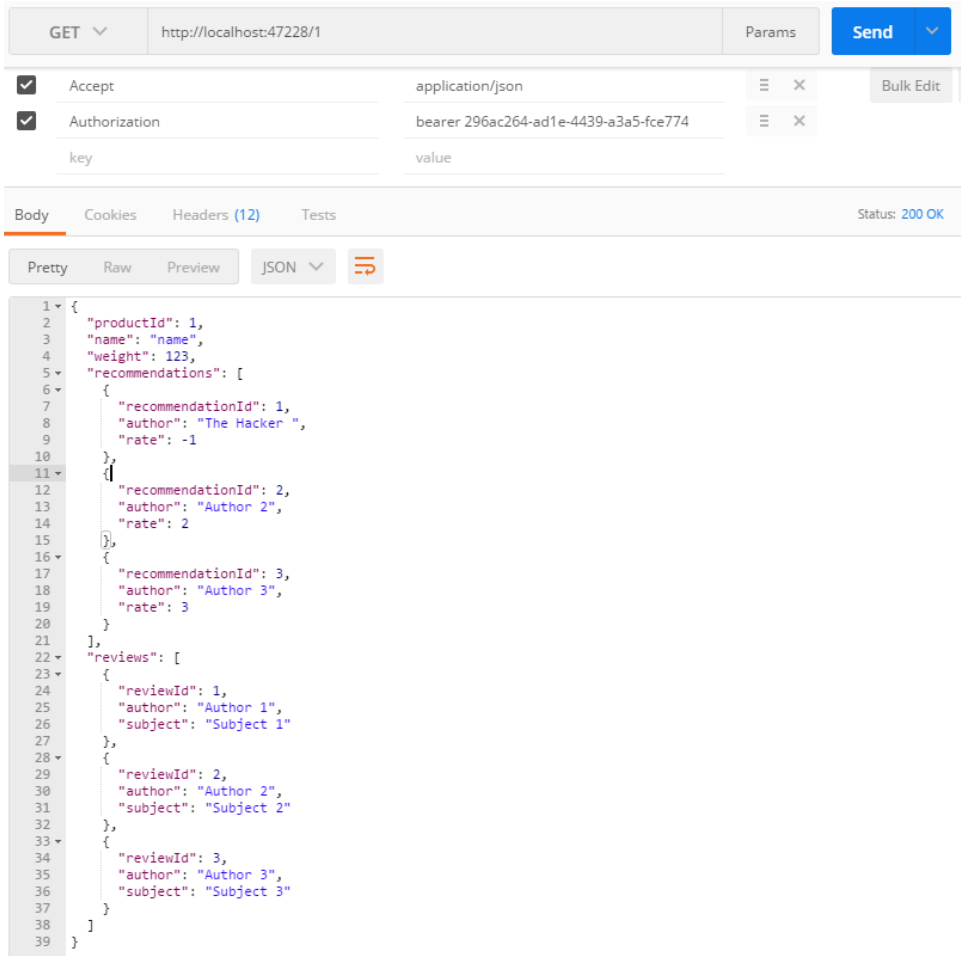
An Adversary can eavesdrop the communication in between the Microservice calls and capture the relayed token. This is possible because the channel in between service calls are not secured. The Attacker can use a proxy or use Wireshark to capture the edge server request by the user.



```
Request Headers
GET /api/product/1 HTTP/1.1
Cache
    Cache-Control: no-cache
Client
    Accept: application/json
    Accept-Encoding: gzip, deflate, sdch, br
    Accept-Language: en-US,en;q=0.8
    User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/56.0.2924.87 Safari/537.36
Miscellaneous
    Postman-Token: d3f560f9-1710-ab65-b5b1-c709efef69e2
Security
    Authorization: bearer 296ac264-ad1e-4439-a3a5-fce77475af46
Transport
    Connection: keep-alive
    Host: localhost:8765
```

*Figure 4.1-5 Token Relay attack : Response analyse*

The Attacker can read the plain text HTTP request header along with the bearer token. The adversary can use the bearer token value and perform an internal service call.



http://localhost:47228/1

| | |
|---|---|
| Accept | application/json |
| Authorization | bearer 296ac264-ad1e-4439-a3a5-fce774 |
| key | value |

Status: 200 OK

```
{
  "productId": 1,
  "name": "name",
  "weight": 123,
  "recommendations": [
    {
      "recommendationId": 1,
      "author": "The Hacker ",
      "rate": -1
    },
    {
      "recommendationId": 2,
      "author": "Author 2",
      "rate": 2
    },
    {
      "recommendationId": 3,
      "author": "Author 3",
      "rate": 3
    }
  ],
  "reviews": [
    {
      "reviewId": 1,
      "author": "Author 1",
      "subject": "Subject 1"
    },
    {
      "reviewId": 2,
      "author": "Author 2",
      "subject": "Subject 2"
    },
    {
      "reviewId": 3,
      "author": "Author 3",
      "subject": "Subject 3"
    }
  ]
}
```

*Figure 4.1-6 Token Relay Attack : Resource response analyse*

As depicted in the figure 4.1-6, the attacker can use the captured bearer token and can use a tool like Postman[33] to send a request to the corresponding Microservice. If the bearer token is still valid, the called Microservice do not have the capability to identify who called the Microservice. As far as the token is valid, the Microservice respond to the request assuming it is a legitimate request from a legitimate Microservice.

## 4.2 Design : Securing Microservices using .Net Identity server

The .Net Identity server is implemented using Web API 2.0 and Microsoft Identity framework which is using the OAuth middleware components. The persistent layer is implemented using the SQL Server 2014.

Microservice Authorization and Authentication is performed using the Identity server. The Edge server Authentication and Authorization is performed using the existing Netflix OSS Identity server.

### 4.2.1 Component Diagram

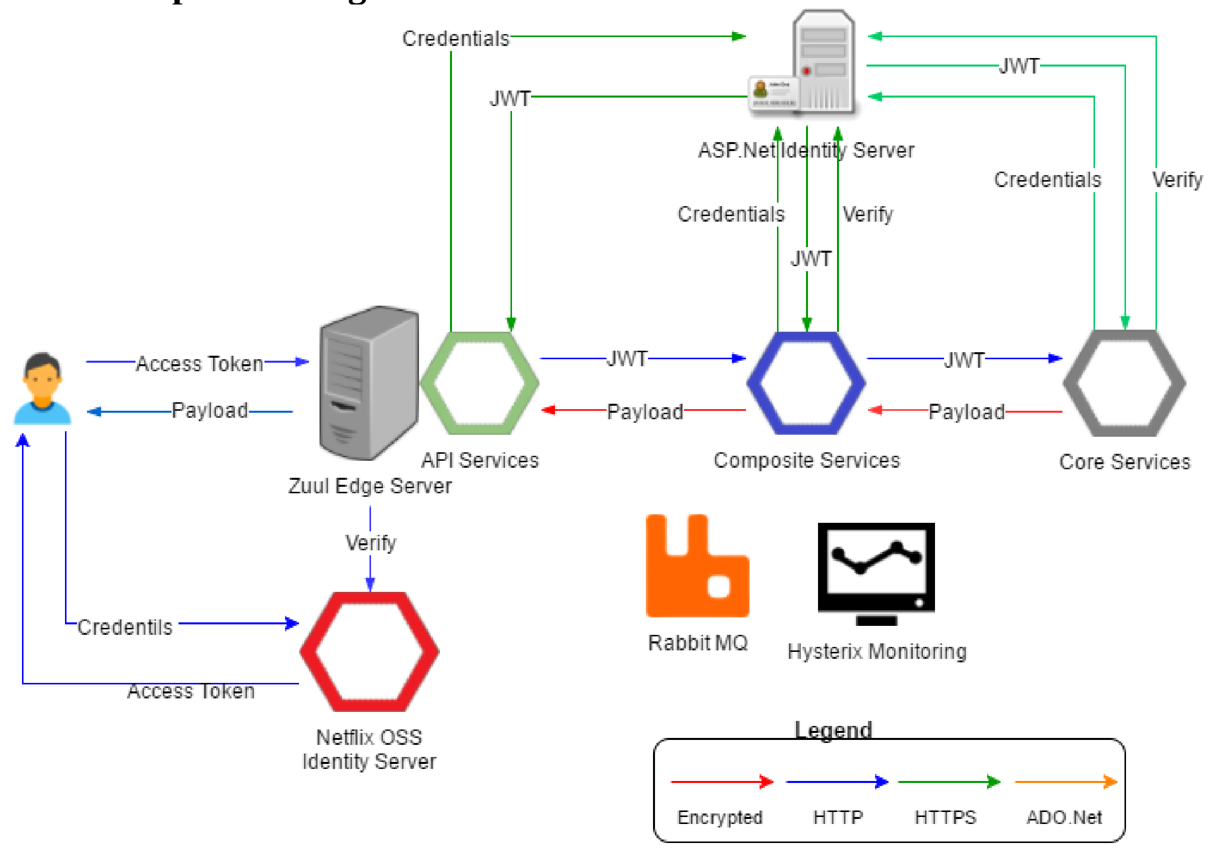


*Figure 4.2-1 Component Diagram*

The Component flow design illustrated in following section. Steps 1 to 4 are already implemented design flows in the Netflix OSS.

The Component Design flow

1. User or the consumer system enters requested credentials by the Netflix OSS Identity Server.
2. Identity server generates an Access Token.
3. User or the consumer system accessing the Netflix OSS Edge server using the Netflix OSS Access Token
4. The Netflix OSS Identity server validates the Access Token and performs required Authentication and Authorization.
5. The Microservice's user spawning the services using Microservice's credentials and credentials for respective Java Keystore.
6. Upon service request calls, the Microservice sends credentials to the .Net Identity server and the .Net Identity server performs the Authentication of the Microservice.
7. The service call is not propagating forward upon Un-Authenticated service call.
8. Upon successful Authentication, the .Net Identity server generates a Json Web Token – JWT and returns it back to the respective Microservice.
9. The calling Microservice sending the respective JWT to the inner level Microservice along with the service API HTTP request.
10. The Inner level Microservice Authorizing the incoming JWT against the .Net Identity server. This Authorization performs using Identity claims received to the
11. The service call is not propagating forward upon Un-Authorized JWT received.
12. Upon successful Authorization, the Microservice generates its own JWT by presenting its own credentials to the .Net Identity server.

13. The Microservice propagates this to inner level and capture the responses from inner level Microservices.
14. The Microservice accessing the Secured Java Keystore using the credentials given by the Spawning user.
15. The Keystore contains shared key for the Calling and Called Microservices.
16. The Called Microservice encrypt the payload and returns the HTTP response to the calling Microservice.
17. Calling Microservice decrypts the response using the Shared key stored in its own Secured Keystore.

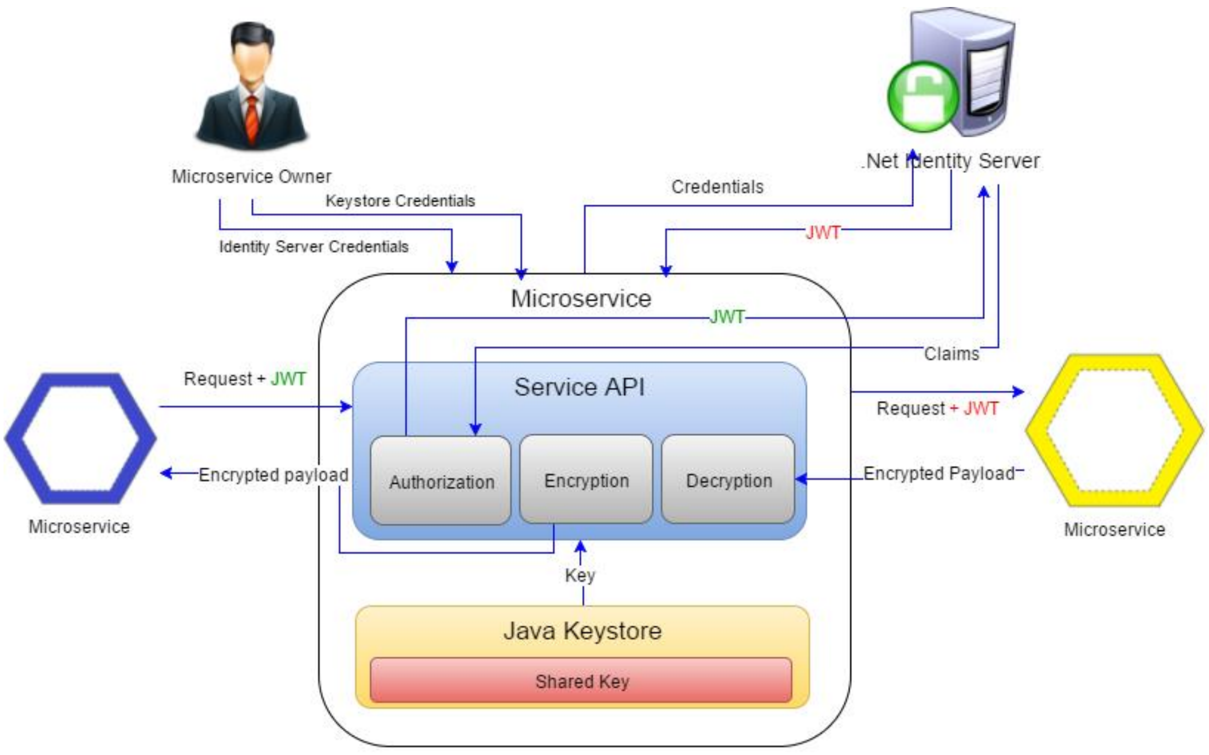


*Figure 4.2-2 Component Flow Diagram*

The component flow diagram depicts the fine-grained security process of the implemented security mechanism. The Microservice's service API contains the Authorization , Encryption and Decryption modules to support the security process. The Encryption and Decryption modules are integrated with the secured Java Keystore. The Java keystore contains the shared secret key used for the Encryption and Decryption of incoming and outgoing payloads.

The Java Keystore secures the keys using encrypted jck file. It can be decrypted only using the Keystore credentials passed during the Microservice spawning process. The appendix A10 contains the source code used to implement the secure Java keystore.

### 4.2.2 Overall Solution Layered Architecture Diagram

The Overall Solution Architecture diagram depicts solution's main components and their interactions.

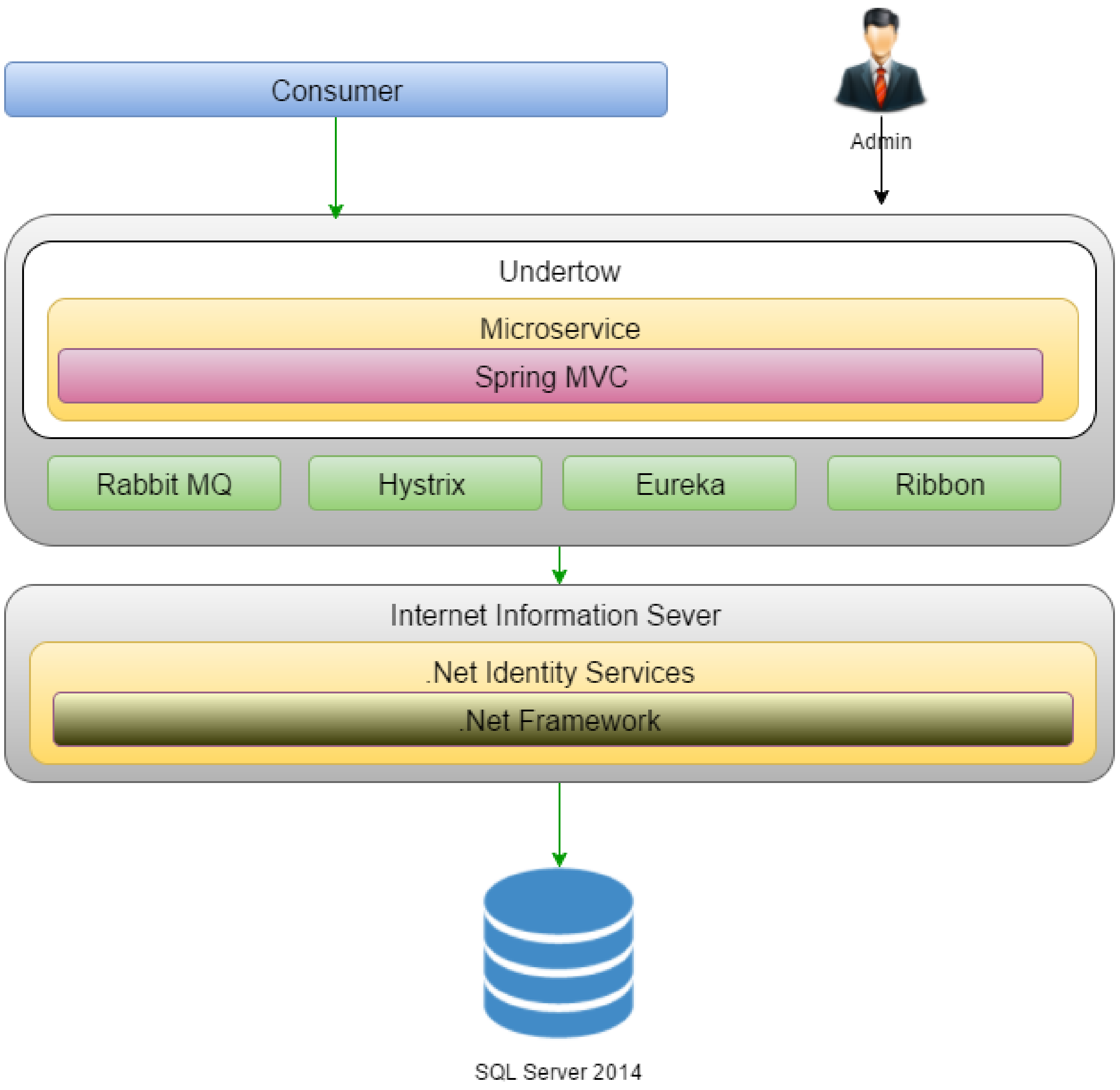


*Figure 4.2-3 Overall Solution :Layered Architecture*

The overall architecture layered diagram depicts how each layer of the application interact each other. The implemented solution contains 4 layers as the consumer layer, Undertow hosted Netflix OSS Microservices layer, IIS hosted .Net Identity server and the data persistent layer implemented using SQL Server 2014.

### 4.2.3 Layered Architecture Diagram - .Net Identity Server

The .Net Identity server is implemented using Web API 2.0 and Microsoft Identity framework which is using the OAuth middleware components. The persistent layer is implemented using the SQL Server 2014. The .Net Identity server is hosted in Internet Information Services (ISS).

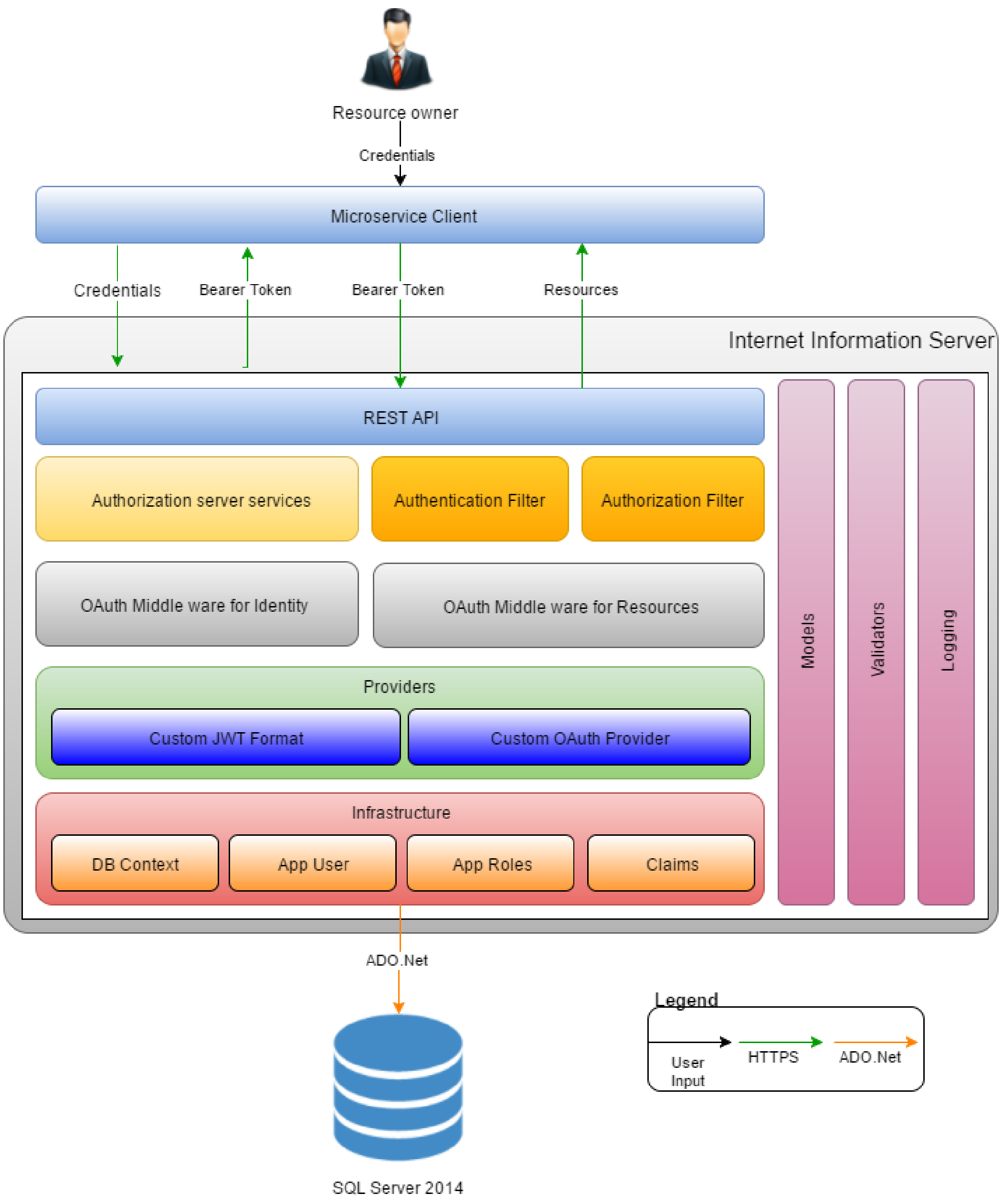


*Figure 4.2-4 .Net Identity Server : Layered Architecture*

Authentication filters are used to authenticate an HTTP request in Web API2. It allows to set an authentication scheme for individual actions or to an ASP.Net MVC Controller [26]. Authorization filters also can be applied to individual actions or to an ASP.Net MVC Controller.

The OAuth 2.0 middleware for Identity and Resources using OWIN framework and it was developed under the Katana project by Microsoft [27]. The Authorization code grant using three methods as Implicit grant, Resource owner password credentials grant and client credentials grant. This project using the, Resource owner password credentials grant in the middleware for Authentication and Authorization.

The provider layer defines the JWT format. The Infrastructure layer is using the ADO.Net along with Entity Framework Object Relational Mapper (ORM) to access SQL Server 2014 database. The Database connection is using a custom system account to access the Database.

### 4.2.4 Sequence Diagram

Sequence diagram depicts the detailed flow of securing Netflix OSS Microservices and how respective objects are behaving when communicating. The Edge server is using the access token created by the Netflix OSS Identity server. The Netflix OSS Identity server generates the access token based on the credentials given by the user.

The user who owns the Microservices are spawning the Microservices with credentials for the .Net Identity server and respective Java key store.

The .Net Identity server performs the Authentication and issue the OAuth JWT to the Microservice1. The Microservice2 performs the Authorization of Microservice1 using Claims issues by the .Net Identity server.

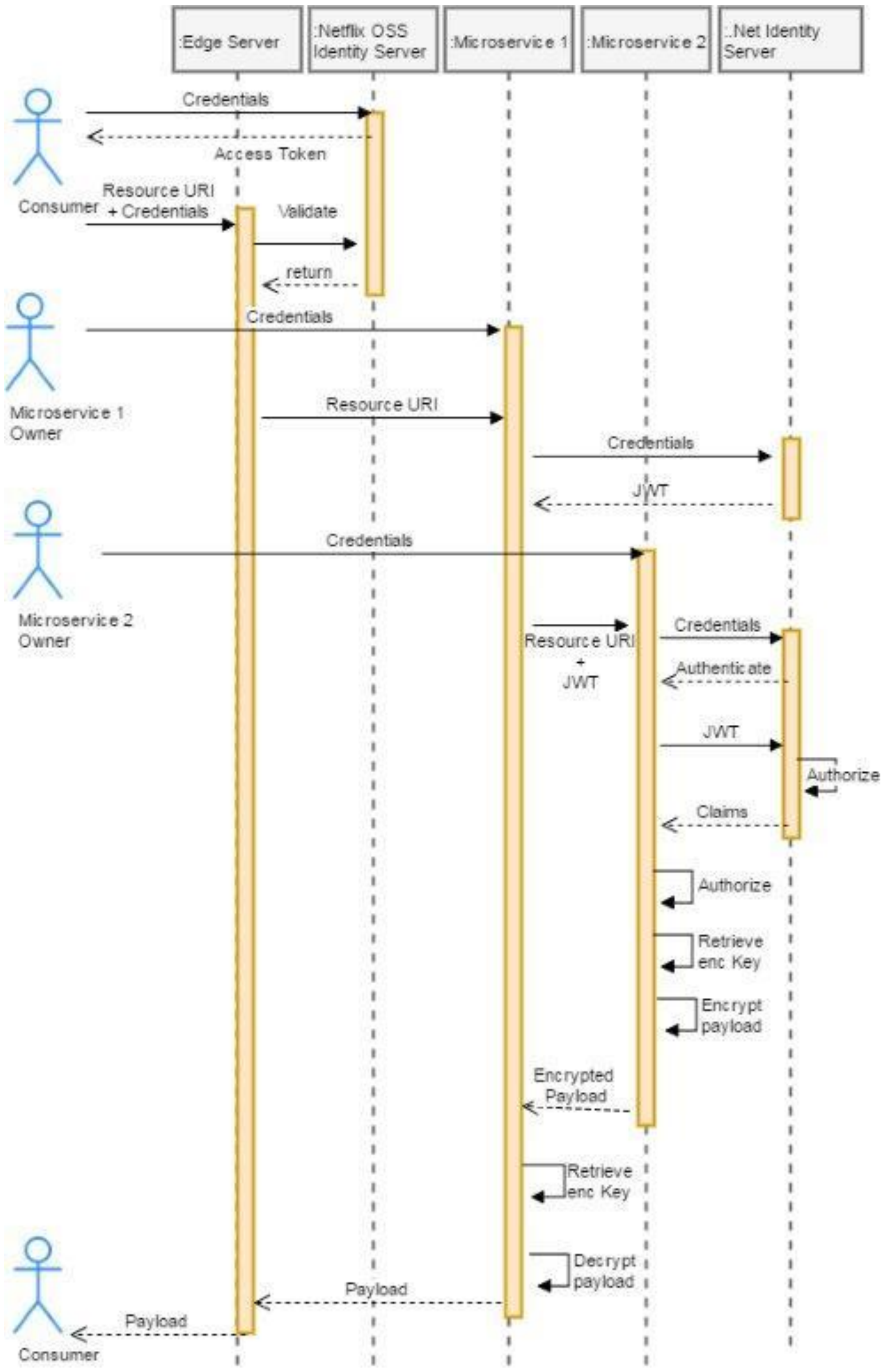


*Figure 4.2-5 Sequence Diagram*

### 4.2.5 Deployment Diagram

Deployment diagram depicts integration in-between deployed project components.

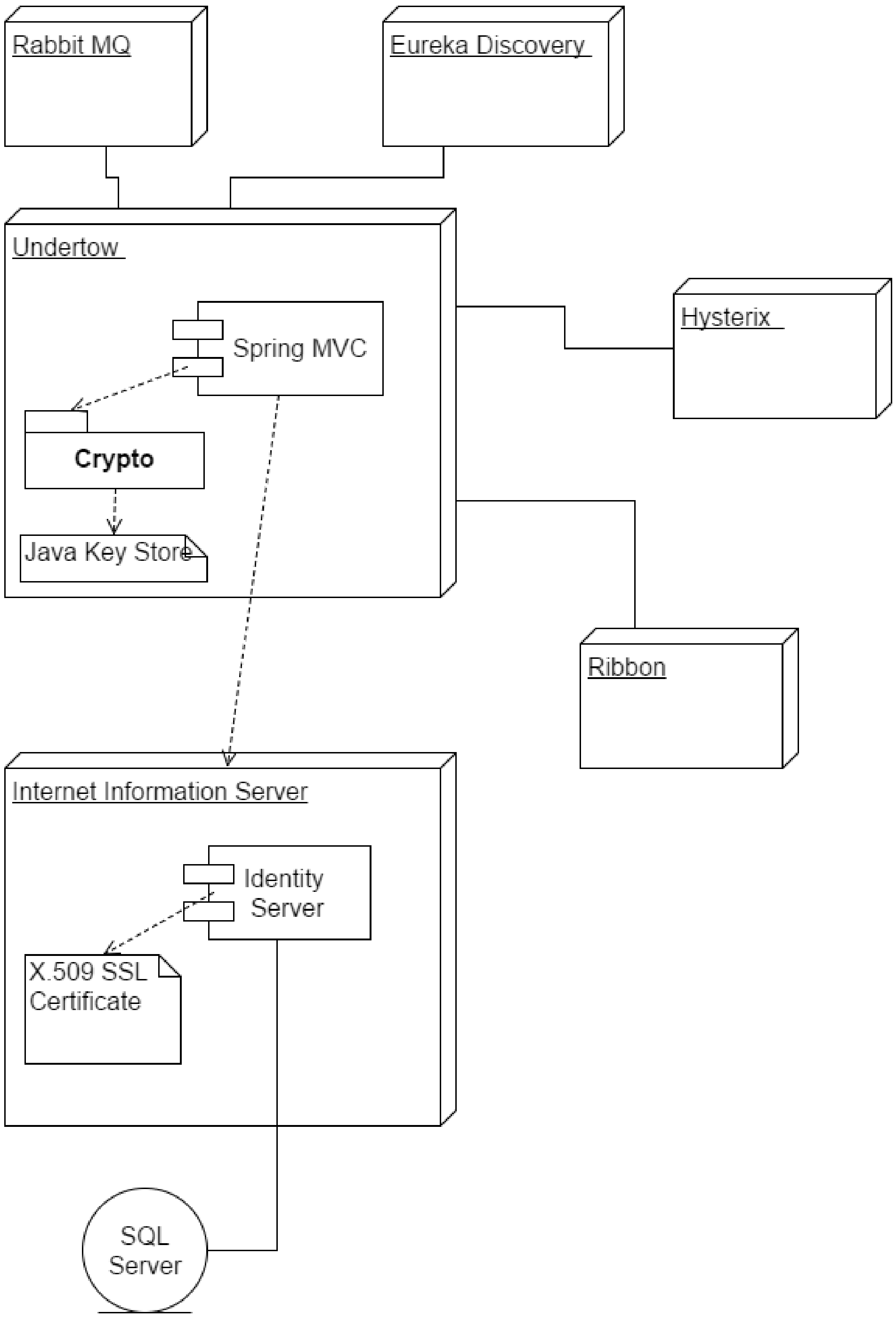


*Figure 4.2-6 Deployment Diagram*

The Netflix OSS Microservices are deployed using the Undertow containers. The Netflix OSS Microservices are hosted in the Undertow containers instead of Docker containers due to hardware limitations in the hosting computer. The hosted Microservice environment in Undertow contains the Secure Java key store and the Crypto classes are using for the encryption and decryption process.

The .Net Identity server is hosted using IIS and the Identity APIs are secured using a X.509 SSL certificate created using the openssl[34] utility . The Spring MVC packages in the Microservice environment communicates with the  IIS using HTTPS protocol.

### 4.2.6 Class Diagrams

API Controllers are the main interfaces with the external systems and all external APIs are inherited from the BaseApiController class.

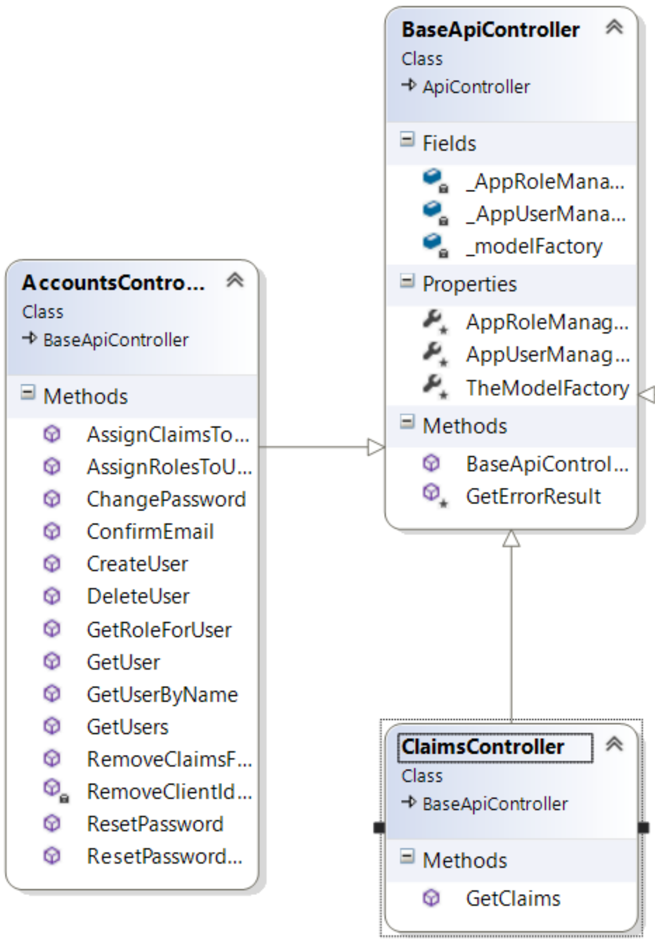


*Figure 4.2-7 Infrastructure layer class diagram*

The AccountsController class responsible for managing REST API user related functionality such as Create , Delete and Read. Assigning Roles and Claims are also managed by the same class.

The Roles functionality API managed by the RolesController class. The ManageUsersInRole method is responsible for handle particular user’s role. User can be in more than one Role in a given time.

Claims are managed by the ClaimsController class. The Getclaims methods returns respective claims for a given JWT. All API controllers and Actions are authorized using the Authorize annotation in the specific Action method in the API class. All the API classes are inherited from the BaseApiController class.

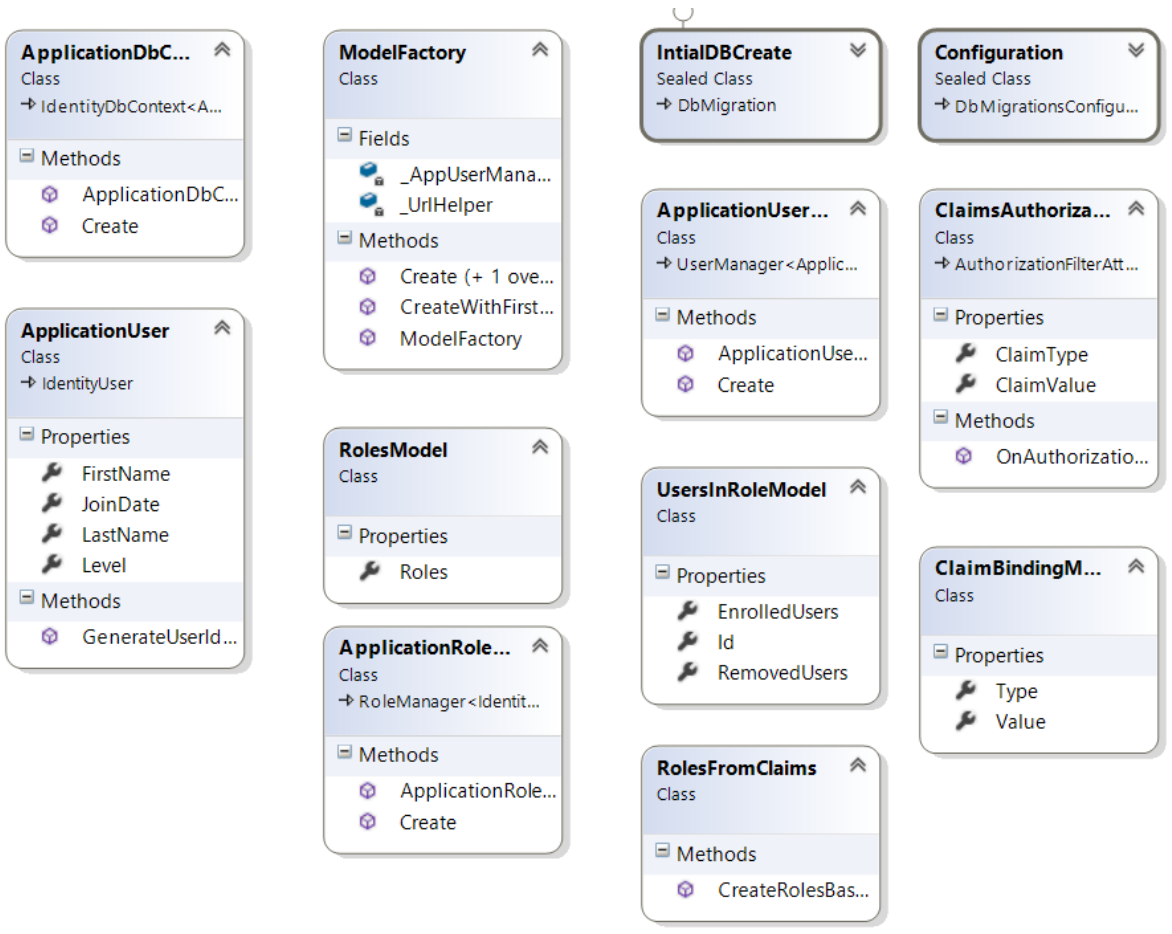


*Figure 4.2-8 Provider layer Class Diagram*

The Provider Layer classes handles internal functionality such as Database migration , Initial Database creation and binding claims to the users etc. Functional models are also featured in the Provider layer class diagram. The ModelFactory class is responsible of creation model objects to use in ORM and API responses.
The provider layer classes contain business logic required by the .Net Identity server.

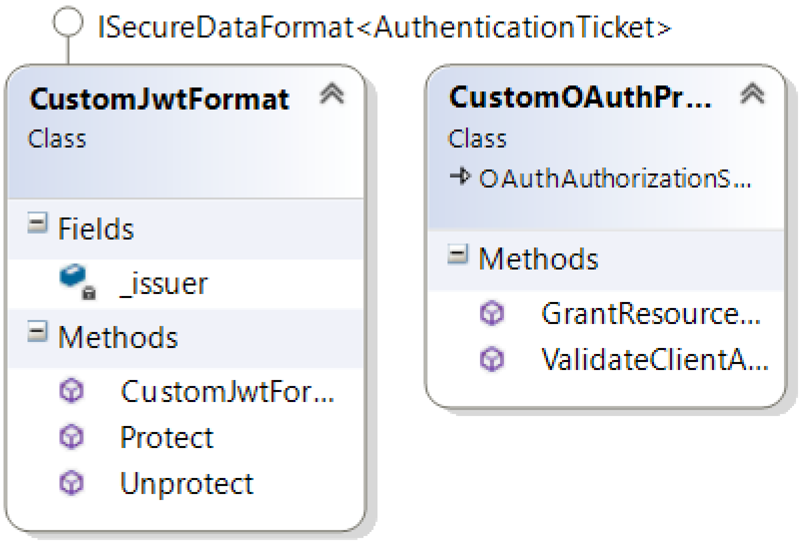


*Figure 4.2-9 Middleware layer class diagram*

The Middleware layer class diagram contains CustomJwtFormat and CustomOAuthProvider classes. The CustomJwtFormat inherited form the ISecureDataFormat interface. It is defines and assigns values to the attributes such as signingKey , issued , expired, token and handler.

The CustomOAuthProvider inherited from the OAuthAuthorizationServerProvider Class. Validate client authentication and Grant Resource Owner Credentials are the main two functionalities of this class.

## 4.3 Implementation: Securing Microservices using .Net Identity Server

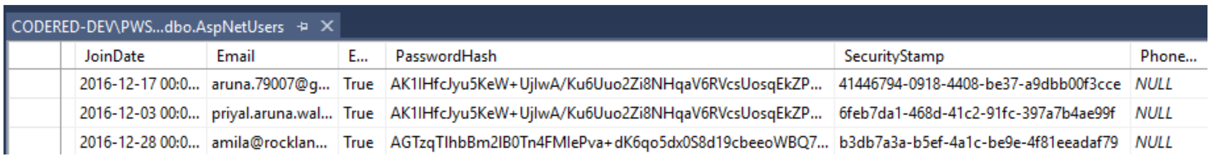
This section illustrates implementation of the selected design.

### 4.3.1 Creating a new Identity user

Creating a new Identity user is managed by the Accounts API which is implemented in the Accounts Controller.  The Microsoft Identity framework persist the user details in the dbo.AspNetUsers table. Password is hashed using the PBKDF2 algorithm and saved in the data table.

The CreateUser method is responsible for creating a user and it is accepting a CreateUserBindingModel data model. The CreateUser is a HTTP Post method ( decorated with the HTTPPost annotation. Only the users in the Admin group are authorized to access this method.

The AccountsController class (Depicts in Appendix A8) handles the user creation API functionality. Other functionalities such as CRUD operations in user object, assign claims, roles to the user and password reset functionalities are also handles by the same class.


CODERED-DEV\PWS...dbo.AspNetUsers

| JoinDate | Email | E... | PasswordHash | SecurityStamp | Phone... |
|---|---|---|---|---|---|
| 2016-12-17 00:0... | aruna.79007@g... | True | AK1IHfcJyu5KeW+UjlwA/Ku6Uuo2Zi8NHqaV6RVcsUosqEkZP... | 41446794-0918-4408-be37-a9dbb00f3cce | *NULL* |
| 2016-12-03 00:0... | priyal.aruna.wal... | True | AK1IHfcJyu5KeW+UjlwA/Ku6Uuo2Zi8NHqaV6RVcsUosqEkZP... | 6feb7da1-468d-41c2-91fc-397a7b4ae99f | *NULL* |
| 2016-12-28 00:0... | amila@rocklan... | True | AGTzqTIhbBm2IB0Tn4FMlePva+dK6qo5dx0S8d19cbeeoWBQ7... | b3db7a3a-b5ef-4a1c-be9e-4f81eeadaf79 | *NULL* |

*Figure 4.3-1 ASP.Net Users table*

The initial admin role need to be seeded into the database upon creation.

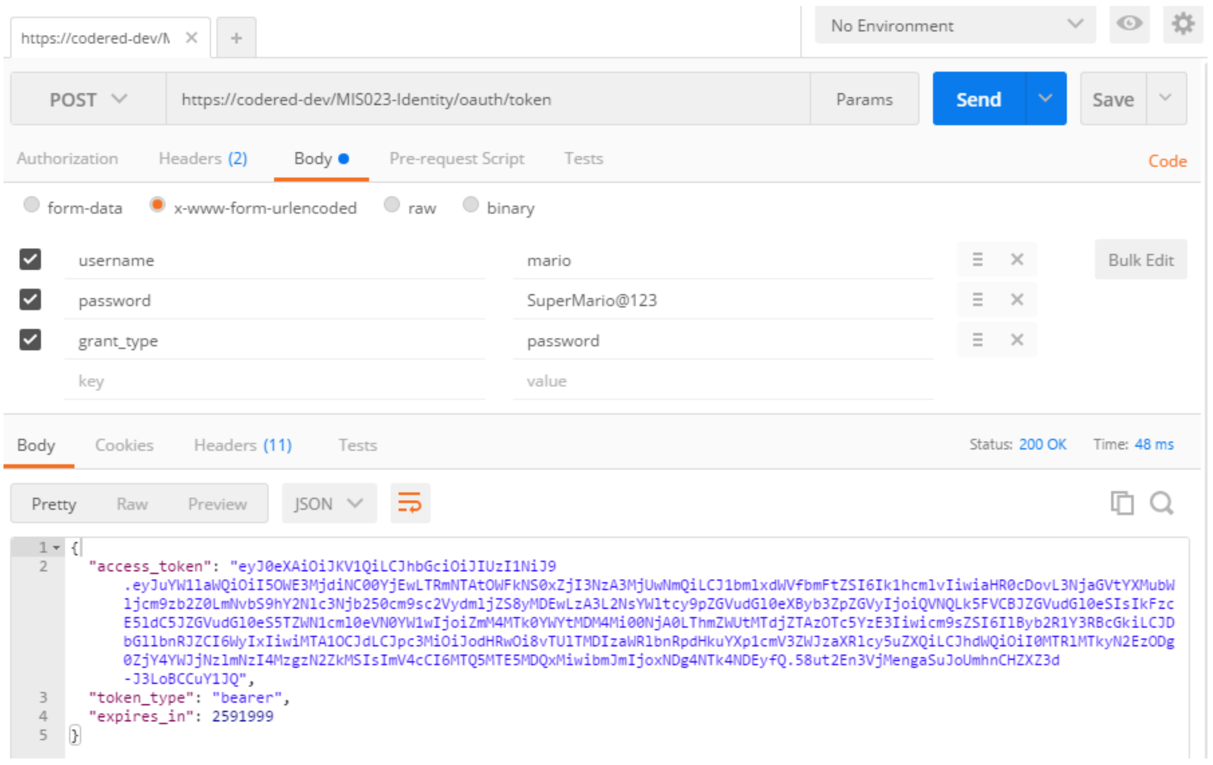
The user should be in the “SuperAdmin” role in order to create a new role. The Authorization is managed by the Authorize Attribute in the POST method defined to create a user.

The Authorize attribute is validated against the incoming JWT in the request header.  This request mapping functionality is implemented in the Startup class in the Identity server application. The startup class is depicted in the Appendix A9.

The created user can be assigned to a role which can have different permission sets. The Roles can be managed using the Roles API which is implemented in the RolesController.

### 4.3.2 Generating the Identity Token

The identity token generation is handled by the “/oauth/token” API. User need to pass in the credentials along with the grant type in the POST request body.


POST https://codered-dev/MIS023-Identity/oauth/token

Body: x-www-form-urlencoded

| key | value |
|---|---|
| username | mario |
| password | SuperMario@123 |
| grant_type | password |

Status: 200 OK Time: 48 ms

```
{
  "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
      .eyJuYW1laWQiOiI5OWE3MjdiNC00YjEwLTRmNTAtOWFkNS0xZjI3NzA3MjUwNmQiLCJ1bmlxdWVfbmFtZSI6Ik1hcmlvIiwiaHR0cDovL3NjaGVtYXMubW
      ljcm9zb2Z0LmNvbS9hY2Nlc3Njb250cm9sc2VydmljZS8yMDEwLzA3L2NsYWltcy9pZGVudGl0eXByb3ZpZGVyIjoiQVNQLk5FVCBJZGVudGl0eSIsIkFzc
      E5ldC5JZGVudGl0eS5TZWN1cml0eVN0YW1wIjoiZmM4MTk0YWYtMDM4Mi00NjA0LThmZWUtMTdjZTAzOTc5YzE3Iiwicm9sZSI6IlByb2R1Y3RBcGkiLCJD
      bGllbnRJZCI6WyIxIiwiMTA1OCJdLCJpc3MiOiJodHRwOi8vTU1TMDIzaWR1bnRpdHkuYXp1cmV3ZWJzaXRlcy5uZXQiLCJhdWQiOiI0MTR1MTkyN2EzODg
      0ZjY4YWJjNzlmNzI4MzgzN2ZkMSIsImV4cCI6MTQ5MTE5MDQxMiwibmJmIjoxNDg4NTk4NDEyfQ.58ut2En3VjMengaSuJoUmhnCHZXZ3d
      -J3LoBCCuY1JQ",
  "token_type": "bearer",
  "expires_in": 2591999
}
```

Figure 4.3-2 HTTPS Request to Identity Server

The token generation accepts only the HTTPS requests because the credentials are passed through the POST request in clear text by the client. Generated token is valid only for two seconds. Each of the inter Micro Services call requires a fresh token be generated and short-lived token guarantees to prevent any misuse of a captured Token by an adversary. This token can be used only one time. If some Microservice sends a token to the Identity server for verification, it will be expired from that point onwards.

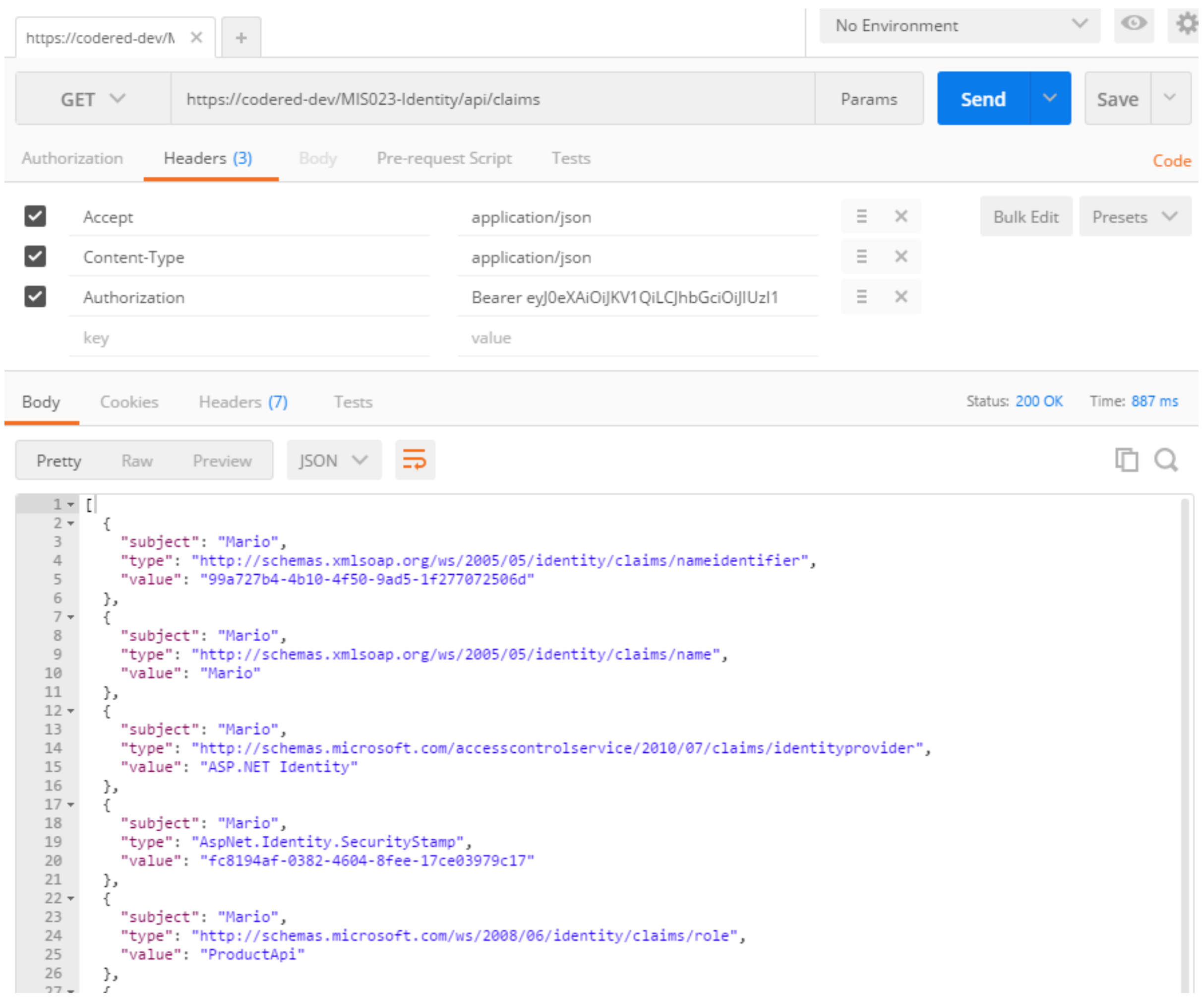
### 4.3.3 User Claims

The Microsoft Identity framework uses the Claims to retrieve Token owner's information such as Roles. The Identity framework allows to create any customized claims if the application required. The client needs to pass in the generated token via the request header in order to retrieve claims. The claims are managed by the "api/claims" API which is implemented in the Claims controller.

The Appendix A4 contains the code segments of the Claims Controller API.

The Authorize annotation make sure that all the requests to the Claims API service are Authorized and no anonymous users can make call to the API.

Claims API call result is as follows.



https://codered-dev/MIS023-Identity/api/claims

| | |
|---|---|
| Accept | application/json |
| Content-Type | application/json |
| Authorization | Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1 |
| key | value |

Status: 200 OK Time: 887 ms

```
[
  {
    "subject": "Mario",
    "type": "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier",
    "value": "99a727b4-4b10-4f50-9ad5-1f277072506d"
  },
  {
    "subject": "Mario",
    "type": "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name",
    "value": "Mario"
  },
  {
    "subject": "Mario",
    "type": "http://schemas.microsoft.com/accesscontrolservice/2010/07/claims/identityprovider",
    "value": "ASP.NET Identity"
  },
  {
    "subject": "Mario",
    "type": "AspNet.Identity.SecurityStamp",
    "value": "fc8194af-0382-4604-8fee-17ce03979c17"
  },
  {
    "subject": "Mario",
    "type": "http://schemas.microsoft.com/ws/2008/06/identity/claims/role",
    "value": "ProductApi"
  },
  {
```

*Figure 4.3-3 HTTPS Request to Claims API*

The Claims API consumer needs to pass in the bearer token in the HTTP request header. Upon successful authorization of the bearer token (JWT) , the Claims API responses with claims values. Claim values are including Security stamp , user Role, token validity period , user name etc..

The Identity Server's APIs are used to Authorize and Authenticate spring service calls. Each of the Microservice needs to spawn using credentials. Microservices need to Authenticate against the Identity server before communicating with deeper level Microservices. Following sequence diagram depicts the Authentication and Authorization process.

### 4.3.4 Securing the .Net Identity Server

The .Net Identity server is secured using Authentication and Authorization mechanisms explained in the previous sections. Apart from this, the IIS server is secured using the Transport Layer Security (TLS) as well.

The IIS deployment of the .Net identity server is secured using a self-signed X.509 certificate.

### 4.3.5 Spawning Microservices

The Spring Microservices are hosted in Gradle containers using Undertow as the web host. Microservices need to spawn using credentials for the Identity server.

```
start /D microservices\api\product-api-service  gradlew bootRun -D<username> -D<password>
```

The Micro Service trying to Authenticate against the Identity server using these injected credentials. The code handles the token generation process depicts in the Appendix A5.

The Token class is depicted in the Appendix A6. Token class's main properties are access_token, token_type and expire_in. The token type denotes the bearer token and expires_in property denotes the expiry time in seconds from the time token is generated.

If the token generation process is successful , the Spring Microservice would call inner level Micro service with the bearer token in the request url's query string parameter.

Following table depicts Micro Service Role Based Authorization privileges.

| Role / Micro Service | NetflixIdentityRole | ProductService | CompositeService |
|---|---|---|---|
| **Edge Server** | Approved | Deny | Deny |
| **Product API** | Approved | Deny | Deny |
| **Composite API** | Deny | Approved | Deny |
| **Review Core Service** | Deny | Deny | Approved |
| **Product Core Service** | Deny | Deny | Approved |
| **Recommendation Service** | Deny | Deny | Approved |

*Table 4.3-1 Identity Server Roles and Authorization*

The called Micro service acquire the token from the query string and call the .Net Identity service to verify the Token's role using claims. The Microservices are Authorized using Role Based Authorization.

The Role authentication in the called Service is performed using the claims API in the .Net Identity server. The Claims API source code depicts in the Appendix A5. Appendix A7 depicts the Java class which handles the incoming claims from the .Net Identity server.

### 4.3.6 Encryption and Decryption of Microservice payloads

The called Micro Service decrypts the payload using the Crypto Java class. The symmetric encryption key is stored securely using the Java Keystore and the method GetKeyStoreKey access the Keystore and returns the shared secret symmetric key. The GetKeyStoreKey method accepts the Keystore alias Key password and the Keystore password as parameters.

The  Encrypt and Decrypt methods handles the encryption and decryption methods respectively. Encryption process using the AES encryption algorithm.

The Appendix A10 depicts the Crypto class implemented using Java. This class is using a key which is expired in one months period. Hence , it is required to update the jck file once a month with a new key. A new password needs to pass-in to the Microservice upon spawning,

# 5 Testing and Evaluation

The implemented security mechanisms need to be tested for performance and security. This chapter elaborates about performance and security testing and evaluation of other Industry solutions to similar problems.

## 5.1 Performance Testing

The Performance testing needs to be performed in the securely hardened Netflix OSS Microservice eco system and needs to compare with the original Netflix OSS Microservices eco system. The Load Complete [18] tool without any customizations is used to perform the performance testing.

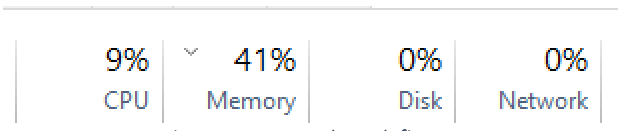
The performance testing is carried out in following scenarios. Output results from the Load Complete tool is evaluated to determine the performance impact in the application. The parameters that are changing for this test are as following.

1. Number of users
2. Plain Microservice API and Security Hardened Microservice API

All the other variables are remains same.

Scenario 1 : Netflix OSS Microservices with 1 user
Scenario 2 : Netflix OSS Microservices with 5 concurrent users
Scenario 3 : Netflix OSS Microservices with 10 concurrent users
Scenario 4 : Security hardened Netflix OSS Microservices with 1 user
Scenario 5 : Security hardened Netflix OSS Microservices with 5 concurrent users
Scenario 6 : Security hardened Netflix OSS Microservices with 10 concurrent users

All the tests are conducted using same hardware benchmark as in figure 5.1-1.



| 9% | 41% | 0% | 0% |
|---|---|---|---|
| CPU | Memory | Disk | Network |

*Figure 5.1-1 Benchmark figures*

### 5.1.1 Test Results

#### 5.1.1.1 Scenario 1 : Netflix OSS Microservices with 1 user

Parameter Description :

**URL :** URL of the Zuul Edger Server. The Edge server is using http protocol in its demo version.
**Netflix Identity Token Value :** The Token generated by the Netflix OSS Identity server. This token is used to Authenticate the Zuul Edger service. ( The token is using changed due to timeouts of the session)
**Number of concurrent users :** Number of users taken into the account during testing. This is a parameter set in the Load complete tool.

**Number of Microservices :** Number of Microservices that are spawned per each type
**Response time :** Response time to execute the request issued to the Edge service

**Testing Parameters**

**URL :** http://localhost:8765/api/product/1
**Netflix Identity Token Value :** 8e24b8d4-63bf-421e-8868-744ede23502b
**Number of concurrent users :** 1
**Number of Microservices :** One from each service
**90% Response time :** 340 ms
The Test result reports are referenced in Appendix B1

#### 5.1.1.2 Scenario 2 : Netflix OSS Microservices with 5 users

**Testing Parameters**

**URL** : http://localhost:8765/api/product/1
**Netflix Identity Token Value** : 8e24b8d4-63bf-421e-8868-744ede23502b
**Number of concurrent users :** 5
**Number of Microservices :** One from each service
**90% Response time :** 403 ms
The Test result reports are referenced in Appendix B2

#### 5.1.1.3 Scenario 3 : Netflix OSS Microservices with 10 users

**Testing Parameters**

**URL :** http://localhost:8765/api/product/1
**Netflix Identity Token Value :** 8e24b8d4-63bf-421e-8868-744ede23502b77
**Number of concurrent users :** 10
**Number of Microservices :** One from each service
**90% Response time :** 461 ms
The Test result reports are referenced in Appendix B3

#### 5.1.1.4 Test Result Analysis

According to the outcome results, it is observed that the response time is increasing when number of concurrent users are increasing. This is due to the increasing resource utilization in the server side. It is also noticed that proportion of latency is decreasing when number of users are increasing.

Most of the request and transfer speeds are decreasing against especially when number of users are increasing. This is due to the response caching capabilities of the Edge server.

#### 5.1.1.5 Scenario 4 : Security hardened Netflix OSS Microservices with 1 user

**Testing Parameters**

**URL :** http://localhost:8765/api/product/1
**Netflix Identity Token Value :** a169572d-fc61-429d-8b60-6dac8193ba54
**Number of concurrent users :** 1
**Number of Microservices :** One from each service
**90% Response time :** 461 ms
The Test result reports are referenced in Appendix B4

#### 5.1.1.6 Scenario 5 : Security hardened Netflix OSS Microservices with 5 concurrent users

**Testing Parameters**

**URL :** http://localhost:8765/api/product/1
**Netflix Identity Token Value :** a169572d-fc61-429d-8b60-6dac8193ba54
**Number of concurrent users :** 5
**Number of Microservices :** One from each service
**90% Response time :** 515 ms
The Test result reports are referenced in Appendix B5

#### 5.1.1.7 Scenario 6 : Security hardened Netflix OSS Microservices with 10 concurrent users

**Testing Parameters**
**URL :** http://localhost:8765/api/product/1
**Token Value :** a169572d-fc61-429d-8b60-6dac8193ba54
**Netflix Identity Number of concurrent users :** 10
**Number of Microservices :** One from each service
**90% Response time :** 945 ms
The Test result reports are referenced in Appendix B6

#### 5.1.1.8 Test Result Analysis

The test results are similar to the section 5.1.1.4 except overall response time been increased after securing the internal Netflix OSS Microservice.

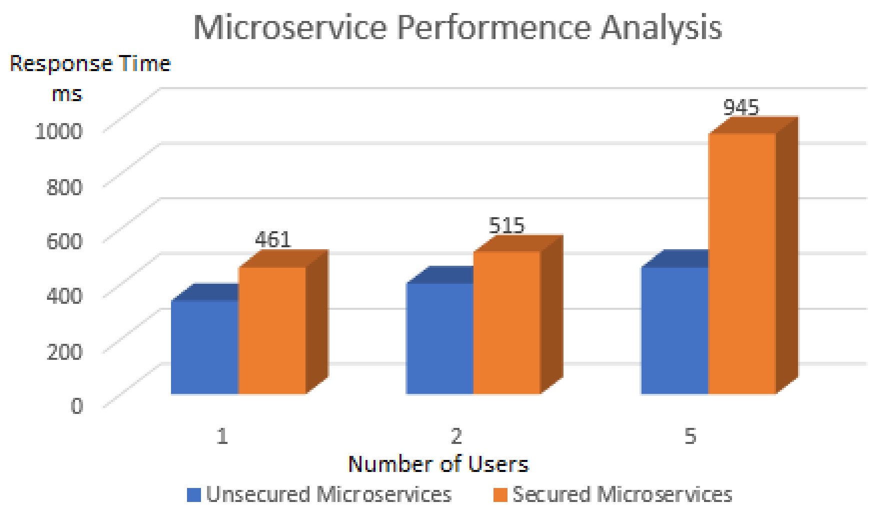


*Figure 5.1-2 Microservice Performance Analysis*

Figure 5.1-2 depicts analysis of 90% response time against number of concurrent users.

The composite analysis of the performance test results depicts that there is a performance degradation occurred due to the security hardening of Netflix OSS internal microservice calls. This is acceptable because of the encryption, decryption and extra authentication service calls that the Microservices need to perform.

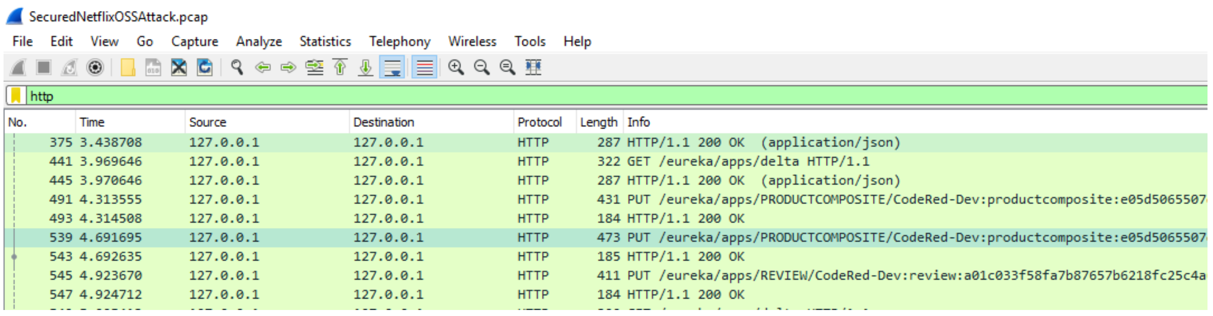
## 5.2 Security Testing

This section illustrates how implemented security methods are securing internal Netflix OSS microservice calls from identified vulnerabilities.

### 5.2.1 Attack 1 : Eavesdropping

The attack is conducted against the Composite service Microservice. The Composite service Microservice is spawned in the port 53661 during the testing attack phase.

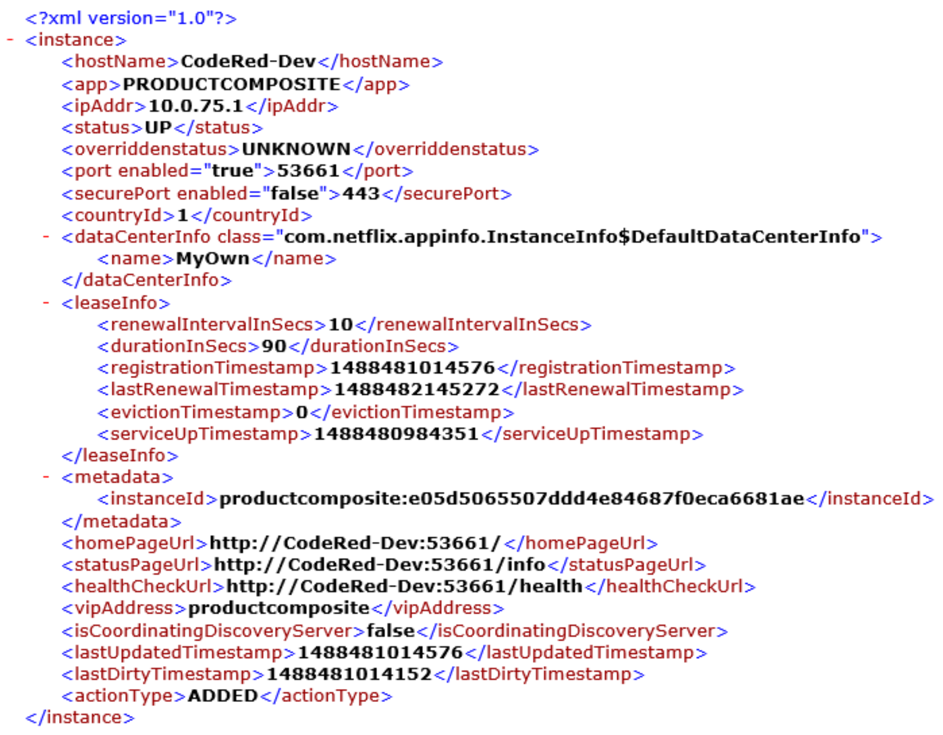
The captured packets are analysed using Wireshark as illustrated below.


SecuredNetflixOSSAttack.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
http

```
No.    Time       Source      Destination  Protocol  Length Info
   375 3.438708   127.0.0.1   127.0.0.1    HTTP      287 HTTP/1.1 200 OK  (application/json)
   441 3.969646   127.0.0.1   127.0.0.1    HTTP      322 GET /eureka/apps/delta HTTP/1.1
   445 3.970646   127.0.0.1   127.0.0.1    HTTP      287 HTTP/1.1 200 OK  (application/json)
   491 4.313555   127.0.0.1   127.0.0.1    HTTP      431 PUT /eureka/apps/PRODUCTCOMPOSITE/CodeRed-Dev:productcomposite:e05d5065507
   493 4.314508   127.0.0.1   127.0.0.1    HTTP      184 HTTP/1.1 200 OK
   539 4.691695   127.0.0.1   127.0.0.1    HTTP      473 PUT /eureka/apps/PRODUCTCOMPOSITE/CodeRed-Dev:productcomposite:e05d5065507
   543 4.692635   127.0.0.1   127.0.0.1    HTTP      185 HTTP/1.1 200 OK
   545 4.923670   127.0.0.1   127.0.0.1    HTTP      411 PUT /eureka/apps/REVIEW/CodeRed-Dev:review:a01c033f58fa7b87657b6218fc25c4a
   547 4.924712   127.0.0.1   127.0.0.1    HTTP      184 HTTP/1.1 200 OK
```

*Figure 5.2-1 Eavesdropping : Wireshark Packet analyse*

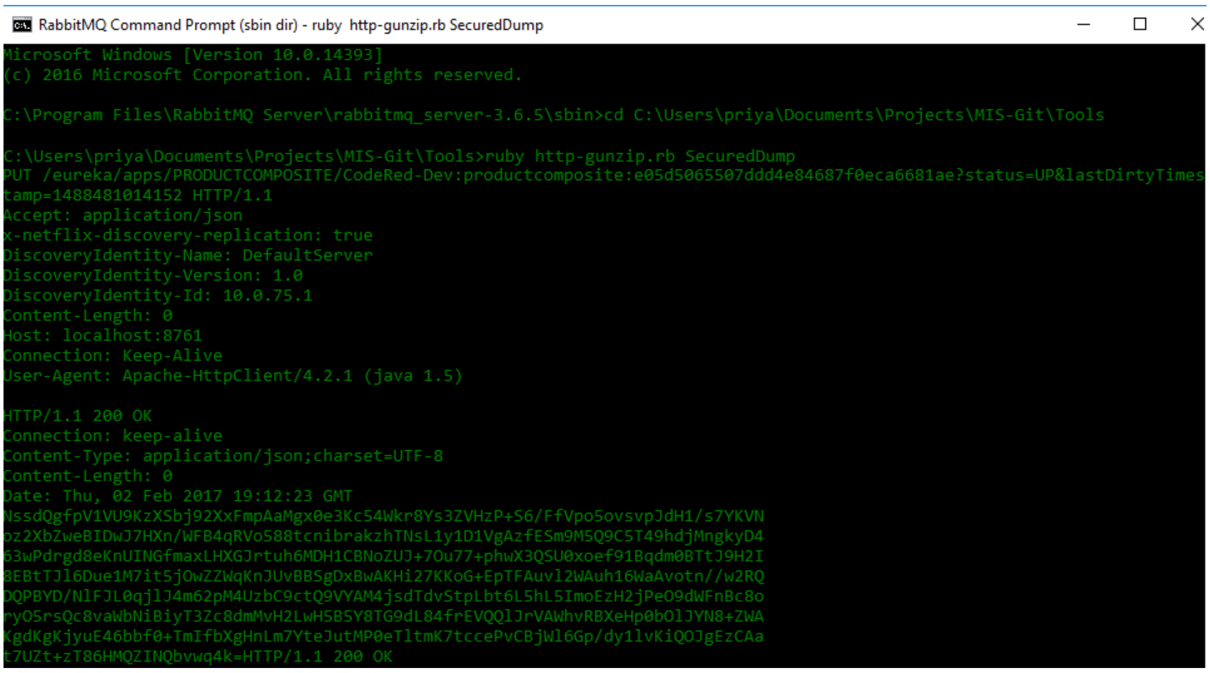
.



```
  <?xml version="1.0"?>
- <instance>
      <hostName>CodeRed-Dev</hostName>
      <app>PRODUCTCOMPOSITE</app>
      <ipAddr>10.0.75.1</ipAddr>
      <status>UP</status>
      <overriddenstatus>UNKNOWN</overriddenstatus>
      <port enabled="true">53661</port>
      <securePort enabled="false">443</securePort>
      <countryId>1</countryId>
    - <dataCenterInfo class="com.netflix.appinfo.InstanceInfo$DefaultDataCenterInfo">
          <name>MyOwn</name>
      </dataCenterInfo>
    - <leaseInfo>
          <renewalIntervalInSecs>10</renewalIntervalInSecs>
          <durationInSecs>90</durationInSecs>
          <registrationTimestamp>1488481014576</registrationTimestamp>
          <lastRenewalTimestamp>1488482145272</lastRenewalTimestamp>
          <evictionTimestamp>0</evictionTimestamp>
          <serviceUpTimestamp>1488480984351</serviceUpTimestamp>
      </leaseInfo>
    - <metadata>
          <instanceId>productcomposite:e05d5065507ddd4e84687f0eca6681ae</instanceId>
      </metadata>
      <homePageUrl>http://CodeRed-Dev:53661/</homePageUrl>
      <statusPageUrl>http://CodeRed-Dev:53661/info</statusPageUrl>
      <healthCheckUrl>http://CodeRed-Dev:53661/health</healthCheckUrl>
      <vipAddress>productcomposite</vipAddress>
      <isCoordinatingDiscoveryServer>false</isCoordinatingDiscoveryServer>
      <lastUpdatedTimestamp>1488481014576</lastUpdatedTimestamp>
      <lastDirtyTimestamp>1488481014152</lastDirtyTimestamp>
      <actionType>ADDED</actionType>
  </instance>
```

*Figure 5.2-2 Request to the Product Composite service*

The Wireshark do not have the capabilities to encode the Gzip encoding. Hence the decoding tool HTTP Gunzip [17] is used to decode the response Json content.



RabbitMQ Command Prompt (sbin dir) - ruby http-gunzip.rb SecuredDump

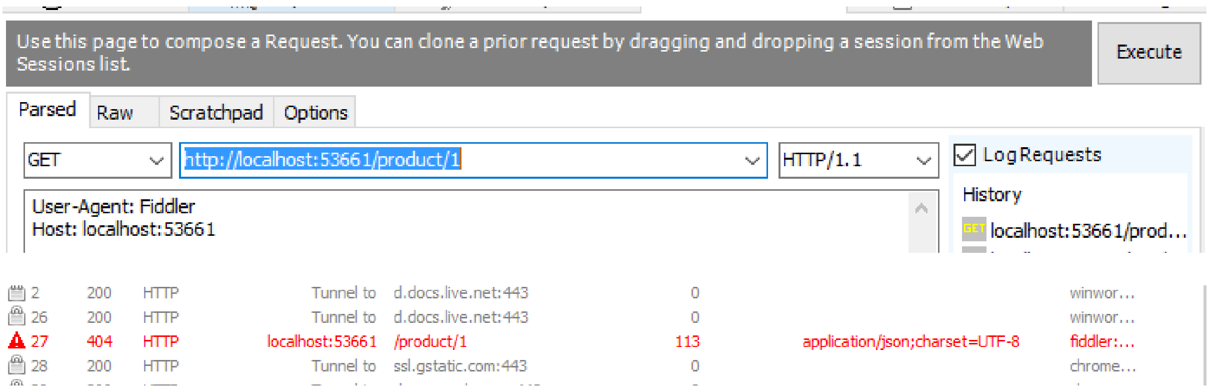
```
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Program Files\RabbitMQ Server\rabbitmq_server-3.6.5\sbin>cd C:\Users\priya\Documents\Projects\MIS-Git\Tools

C:\Users\priya\Documents\Projects\MIS-Git\Tools>ruby http-gunzip.rb SecuredDump
PUT /eureka/apps/PRODUCTCOMPOSITE/CodeRed-Dev:productcomposite:e05d5065507ddd4e84687f0eca6681ae?status=UP&lastDirtyTimes
tamp=1488481014152 HTTP/1.1
Accept: application/json
x-netflix-discovery-replication: true
DiscoveryIdentity-Name: DefaultServer
DiscoveryIdentity-Version: 1.0
DiscoveryIdentity-Id: 10.0.75.1
Content-Length: 0
Host: localhost:8761
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.2.1 (java 1.5)

HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: application/json;charset=UTF-8
Content-Length: 0
Date: Thu, 02 Feb 2017 19:12:23 GMT
NssdQgfpV1VU9KzXSbj92XxFmpAaMgx0e3Kc54Wkr8Ys3ZVHzP+S6/FfVpo5ovsvpJdH1/s7YKVN
oz2XbZweBIDwJ7HXn/WFB4qRVo588tcnibrakzhTNsL1y1D1VgAzfESm9M5Q9C5T49hdjMngkyD4
63wPdrgd8eKnUINGfmaxLHXGJrtuh6MDH1CBNoZUJ+7Ou77+phwX3QSU0xoef91Bqdm0BTtJ9H2I
8EBtTJl6Due1M7it5jOwZZWqKnJUvBBSgDxBwAKHi27KKoG+EpTFAuvl2WAuh16WaAvotn//w2RQ
DQPBYD/NlFJL0qjlJ4m62pM4UzbC9ctQ9VYAM4jsdTdvStpLbt6L5hL5ImoEzH2jPeO9dWFnBc8o
ryO5rsQc8vaWbNiBiyT3Zc8dmMvH2LwH5B5Y8TG9dL84frEVQQlJrVAWhvRBXeHp0bOlJYN8+ZWA
KgdKgKjyuE46bbf0+TmIfbXgHnLm7YteJutMP0eTltmK7tccePvCBjWl6Gp/dy1lvKiQOJgEzCAa
t7UZt+zT86HMQZINQbvwq4k=HTTP/1.1 200 OK
```

*Figure 5.2-3 Decoded response : Json content*

The Attacker can visualize only the encrypted payload. Hence the implemented method successfully mitigates the Eavesdropping attack against Netflix OSS internal microservice calls.

### 5.2.2 Attack 2 : Confused Deputy Attack

The confused deputy attack can be conducted using Fiddler tool. The Attacker can call the internal Product Composite service directly bypassing the Edge server as illustrated below. The Product Composite service is running in port 53661.


Use this page to compose a Request. You can clone a prior request by dragging and dropping a session from the Web Sessions list.
Execute
Parsed
Raw
Scratchpad
Options
Log Requests
History

```
GET    http://localhost:53661/product/1    HTTP/1.1
User-Agent: Fiddler
Host: localhost:53661
```

localhost:53661/prod...

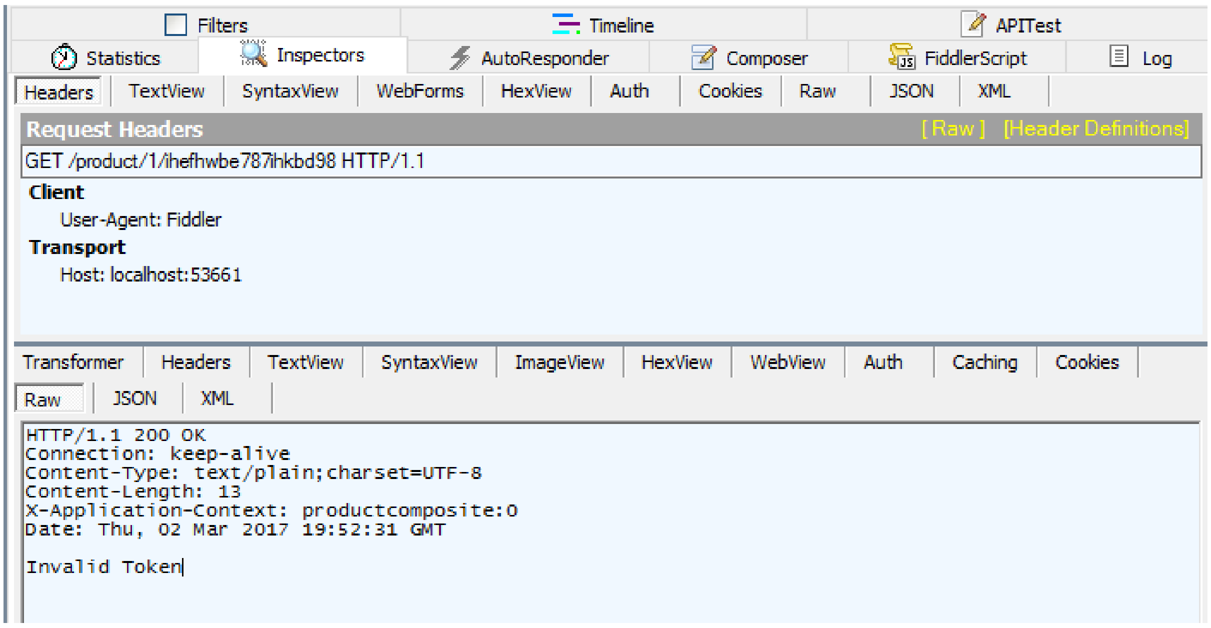
```
2    200  HTTP  Tunnel to        d.docs.live.net:443   0                                     winwor...
26   200  HTTP  Tunnel to        d.docs.live.net:443   0                                     winwor...
27   404  HTTP  localhost:53661  /product/1            113  application/json;charset=UTF-8  fiddler:...
28   200  HTTP  Tunnel to        ssl.gstatic.com:443   0                                     chrome...
```

*Figure 5.2-4 Confused Deputy Attack : Composite Service 404 Error*

This is resulted with HTTP 404-Not Found error. Because the secured Product Composite service required following URI format.

http://localhost:53661/product/{ProductId}/{JWT}

The attacker can enumerate possible URIs and identify the required pattern and send a request with random JWT value. The Product Composite service returns HTTP 200 -Ok code. But the Attacker do not pose a valid JWT to authenticate himself against the Product Composite microservice. The Product Composite microservice returns the error message "Invalid Token"



```
GET /product/1/ihefhwbe787ihkbd98 HTTP/1.1
Client
    User-Agent: Fiddler
Transport
    Host: localhost:53661
```

```
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: text/plain;charset=UTF-8
Content-Length: 13
X-Application-Context: productcomposite:0
Date: Thu, 02 Mar 2017 19:52:31 GMT

Invalid Token
```

*Figure 5.2-5 Confused Deputy Attack : Invalid Token*

Hence the implemented method successfully mitigates the confused deputy attack against Netflix OSS internal microservice calls.

### 5.2.3 Attack 3: Man in the Middle Attack

The attacker is not being able to perform a MiTM attack because of following reasons.

1. Attacker do not know the required URI format to call the internal microservices after introducing the JWT token.
2. Even though attacker find the required URI format, attacker cannot authenticate against the internal microservice because the attacker do not possessing a valid JWT token generated by the .Net Identity server.
3. Attacker cannot change the returning payload because the attacker do not poses a valid key to encrypt the payload. The Secured Netflix OSS microservices can accepts only encrypted payloads which are encrypted using legitimate key.

Hence the implemented method successfully mitigates the MiTM attack against Netflix OSS internal microservice calls.

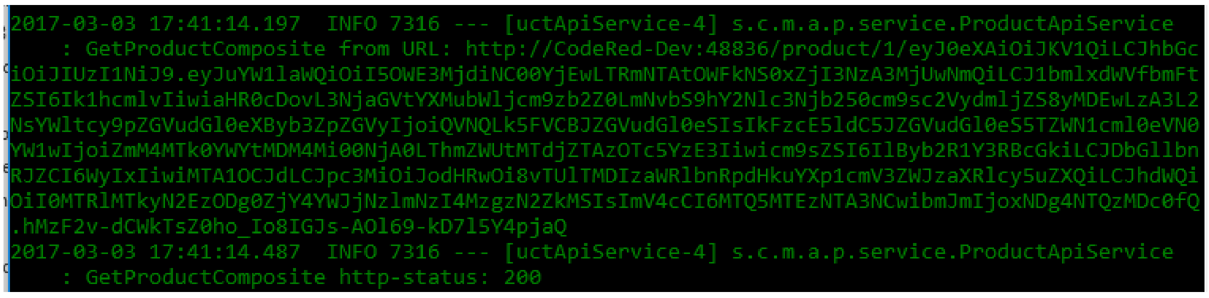
### 5.2.4 Attack 4: Eavesdropping attack against the .Net Identity server to capture credentials

Eavesdropping attack against the .Net Identity server is not possible because it is being secured using the Transport Layer Security (TLS). Hence the communication in between secured Microservice and the .Net Identity server is encrypted.

### 5.2.5 Attack 5: Replay  attack with captured JWT token

The Request from one Microservice to another is not encrypted or not using TLS. Hence it is possible to attacker to gain access to the Request header and capture the Authentication Token. An Attacker can try access inner level Microservice using this captured token.

Capturing the Authentication (JWT) token



```
2017-03-03 17:41:14.197  INFO 7316 --- [uctApiService-4] s.c.m.a.p.service.ProductApiService
    : GetProductComposite from URL: http://CodeRed-Dev:48836/product/1/eyJ0eXAiOiJKV1QiLCJhbGc
iOiJIUzI1NiJ9.eyJuYW1laWQiOiI5OWE3MjdiNC00YjEwLTRmNTAtOWFkNS0xZjI3NzA3MjUwNmQiLCJ1bmlxdWVfbmFt
ZSI6Ik1hcmlvIiwiaHR0cDovL3NjaGVtYXMubWljcm9zb2Z0LmNvbS9hY2Nlc3Njb250cm9sc2VydmljZS8yMDEwLzA3L2
NsYWltcy9pZGVudGl0eXByb3ZpZGVyIjoiQVNQLk5FVCBJZGVudGl0eSIsIkFzcE5ldC5JZGVudGl0eS5TZWN1cml0eVN0
YW1wIjoiZmM4MTk0YWYtMDM4Mi00NjA0LThmZWUtMTdjZTAzOTc5YzE3Iiwicm9sZSI6IlByb2R1Y3RBcGkiLCJDbGllbn
RJZCI6WyIxIiwiMTA1OCJdLCJpc3MiOiJodHRwOi8vTUlTMDIzaWRlbnRpdHkuYXp1cmV3ZWJzaXRlcy5uZXQiLCJhdWQi
OiI0MTRlMTkyN2EzODg0ZjY4YWJjNzlmNzI4MzgzN2ZkMSIsImV4cCI6MTQ5MTEzNTA3NCwibmJmIjoxNDg4NTQzMDc0fQ
.hMzF2v-dCWkTsZ0ho_Io8IGJs-AOl69-kD7l5Y4pjaQ
2017-03-03 17:41:14.487  INFO 7316 --- [uctApiService-4] s.c.m.a.p.service.ProductApiService
    : GetProductComposite http-status: 200
```

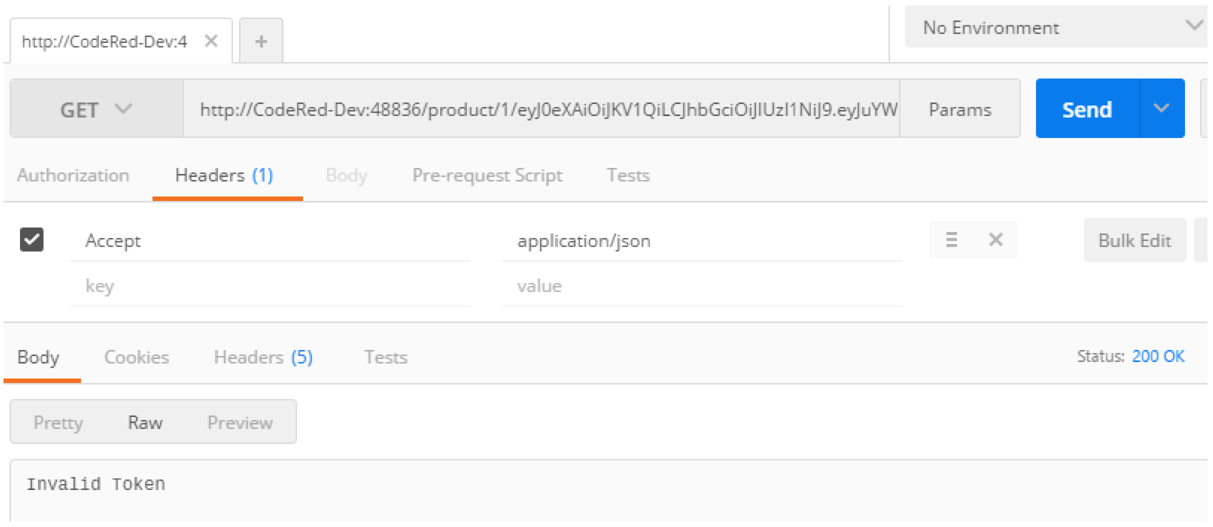
*Figure 5.2-6 Captured Token*

Captured Token by the Attacker :

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJuYW1laWQiOiI5OWE3MjdiNC00YjEwLTRmNTAtOWFkNS0xZjI3NzA3
MjUwNmQiLCJ1bmlxdWVfbmFtZSI6Ik1hcmlvIiwiaHR0cDovL3NjaGVtYXMubWljcm9zb2Z0LmNvbS9hY2Nlc3Njb
250cm9sc2VydmljZS8yMDEwLzA3L2NsYWltcy9pZGVudGl0eXByb3ZpZGVyIjoiQVNQLk5FVCBJZGVudGl0eSIsIkFz
cE5ldC5JZGVudGl0eS5TZWN1cml0eVN0YW1wIjoiZmM4MTk0YWYtMDM4Mi00NjA0LThmZWUtMTdjZTAzOTc5
YzE3Iiwicm9sZSI6IlByb2R1Y3RBcGkiLCJDbGllbnRJZCI6WyIxIiwiMTA1OCJdLCJpc3MiOiJodHRwOi8vTUlTMDIzaW
RlbnRpdHkuYXp1cmV3ZWJzaXRlcy5uZXQiLCJhdWQiOiI0MTRlMTkyN2EzODg0ZjY4YWJjNzlmNzI4MzgzN2ZkMSIs

ImV4cCI6MTQ5MTEzNTA3NCwibmJmIjoxNDg4NTQzMDc0fQ.hMzF2v-dCWkTsZ0ho_Io8IGJs-AOl69-kD7l5Y4pjaQ

Attacker can use a tool like Postman to access internal Microservice directly using this JWT.



```
Invalid Token
```

*Figure 5.2-7 Failed Replay Attack*

The attack attempt is a failure because the Token is already consumed by the internal Microservice (Product Composite Service ). Hence, the token is not valid anymore to use by the Attacker.

### 5.2.6 General Security Test Cases

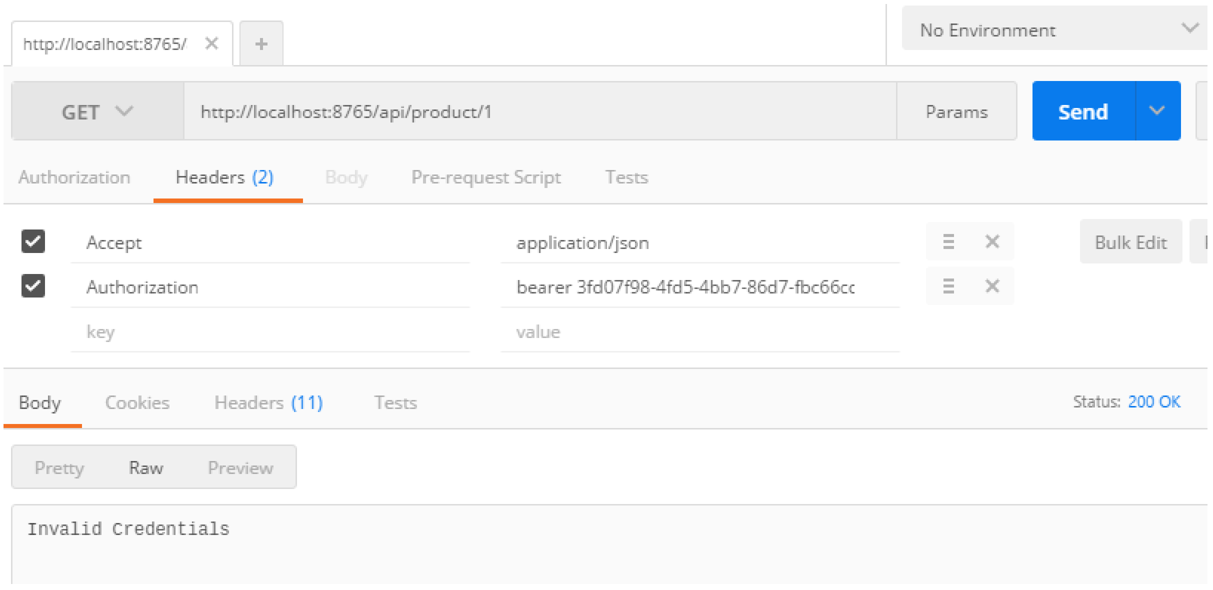
Several test cases are executed to validate user scenarios in the secured Netflix OSS Microservices

| TC # | Test Case | Expected Results | Actual Results | Pass/Fail |
|---|---|---|---|---|
| 1 | Spawning Microservice with Invalid Credentials | Microservice should be Unauthorized | "Invalid Credentials" message Returned with HTTP-200 | Pass |

**Approach :** Spawn a Microservice with invalid .Net Identity credentials

start /D microservices\core\product-service gradlew bootRun -DUserName=invaliduser -DInvalidPassword -DKeyStoreCredentials-M1so@#123

**Result :**



GET http://localhost:8765/api/product/1

| Key | Value |
|---|---|
| Accept | application/json |
| Authorization | bearer 3fd07f98-4fd5-4bb7-86d7-fbc66cc |

Status: 200 OK

```
Invalid Credentials
```

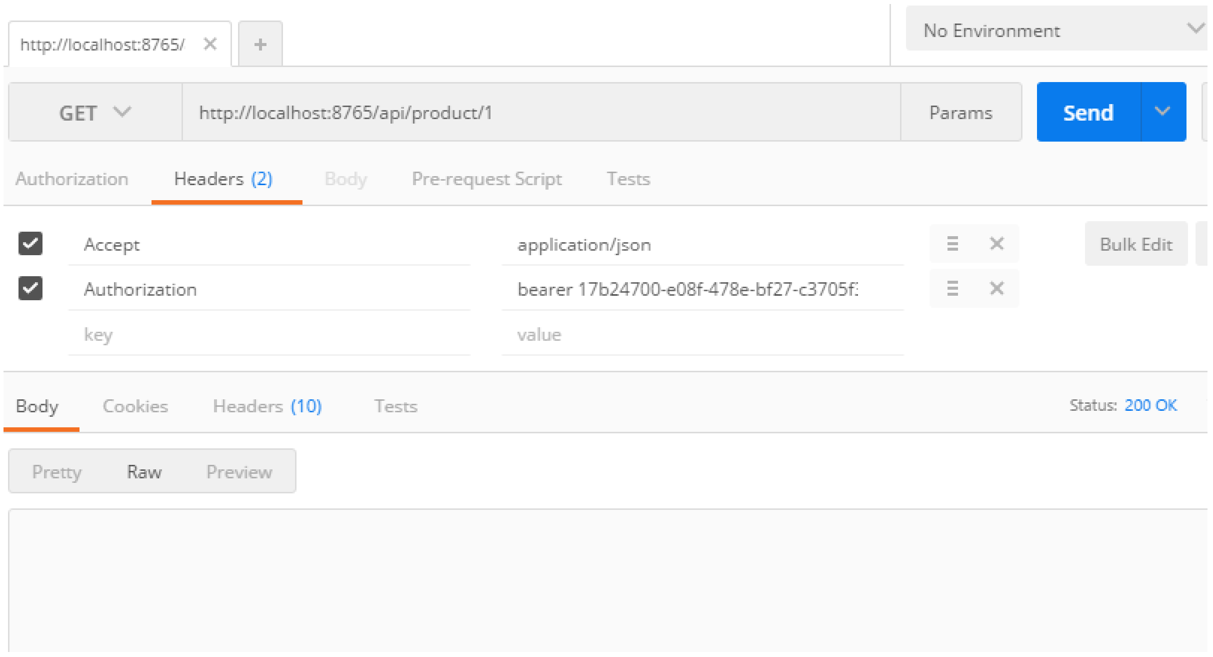
*Figure 5.2-8 Test Case : Spawning Microservice with Invalid Credentials*

| TC # | Test Case | Expected Results | Actual Results | Pass/Fail |
|---|---|---|---|---|
| 2 | Spawning Microservice with Invalid Credentials to Java Key store | Null payload should return | Null payload | Pass |

**Approach :** Spawn a Microservice with invalid Java Keystore credentials

start /D microservices\core\product-service gradlew bootRun -DUserName=mario -D SuperMario@123 -DKeyStoreCredentials-gfhgfhgfhgf

**Result :**



GET http://localhost:8765/api/product/1

| Key | Value |
|---|---|
| Accept | application/json |
| Authorization | bearer 17b24700-e08f-478e-bf27-c3705f |

Status: 200 OK

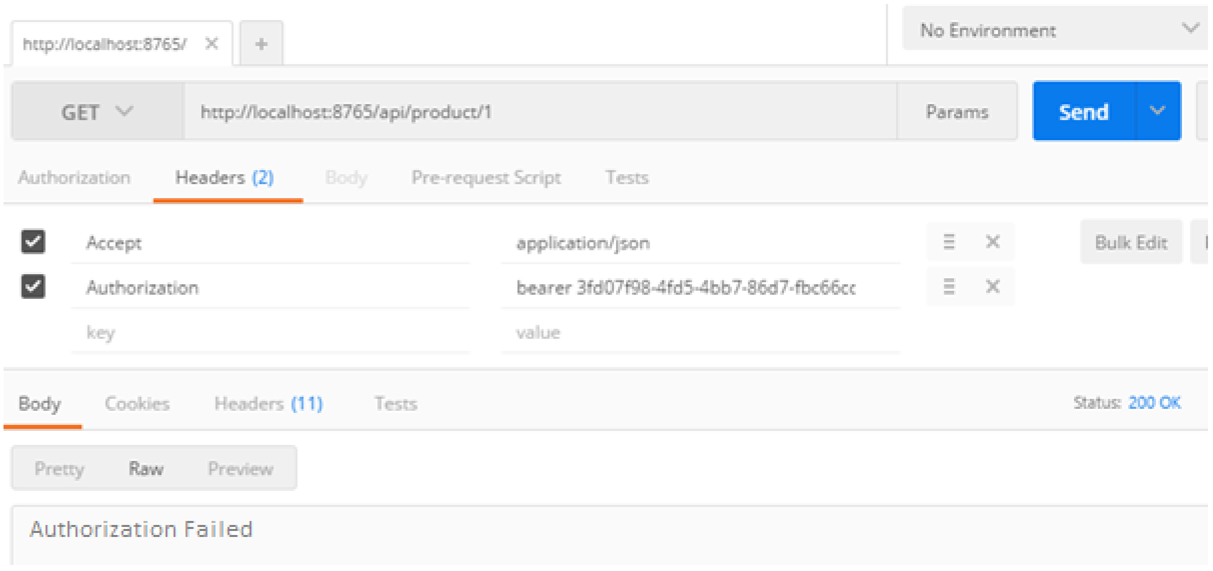
*Figure 5.2-9 Test Case : Spawning Microservice with Invalid Credentials to Java Key store*

| TC # | Test Case | Expected Results | Actual Results | Pass/Fail |
|---|---|---|---|---|
| 3 | Spawning Microservice with Valid Credentials but invalid application role | Service call should be Unauthorized | "Authorization Failed" message returned with HTTP -200 | Pass |

**Approach :** Spawn a Microservice with invalid .Net Identity credentials

```
start /D microservices\core\product-service   gradlew bootRun -DUserName=priyalw -DPriyal@123 -DKeyStoreCredentials-M1so@#123
```

**Result :**



GET http://localhost:8765/api/product/1

| | |
|---|---|
| Accept | application/json |
| Authorization | bearer 3fd07f98-4fd5-4bb7-86d7-fbc66cc |
| key | value |

Status: 200 OK

```
Authorization Failed
```

*Figure 5.2-10 Test Case : Spawn a Microservice with invalid .Net Identity credentials*

## 5.3 Study of Similar industrial solutions for Microservice security

This sections elaborates examples of how industrial solutions tried to solve the security issues in Microservices.

### 5.3.1 DZone : The Interceptor pattern

The Interceptor pattern capturing inbound and outbound HTTP traffic flow in a Microservice. The capturing information are HTTP URI, URL, and the credentials provided by the HTTP agent[20].

Following picture depicts the Interceptor pattern in its general implementation.

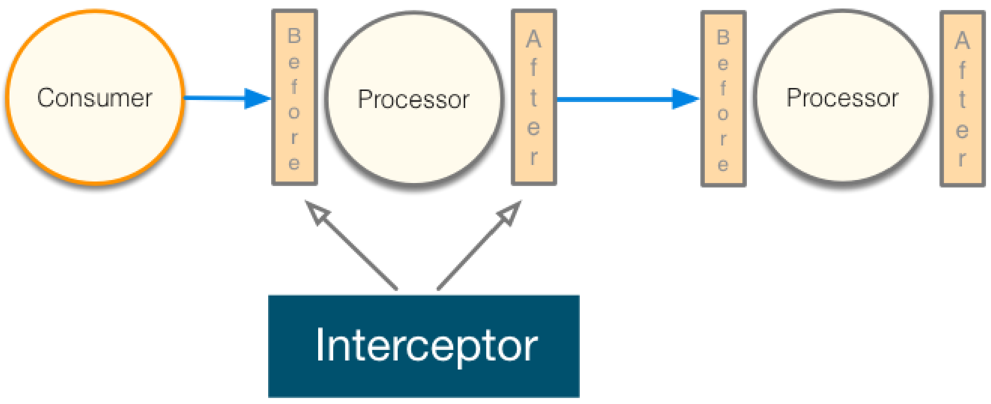


*Figure 5.3-1 The Interceptor Pattern[20]*

The interceptor acts as a Security guardian for the Microservice implementation. The Interceptor can use existing security frameworks such as Apache Shiro, Spring Security, and Apache WS4J in order to perform the Authentication and Authorization.

The Interceptor pattern being enriched by using the features offered by the Jetty/Netty HTTP Server by,

1. Enable Secure transmission using TLS between HTTP Agent and the Server
2. Two-way SSL /TSL
3. Restrict Web resource access using SecurityConstraint associated with a user's role.
4. The Web HTTP container controls the user Authentication.
5. The Interceptor control the Authorization.

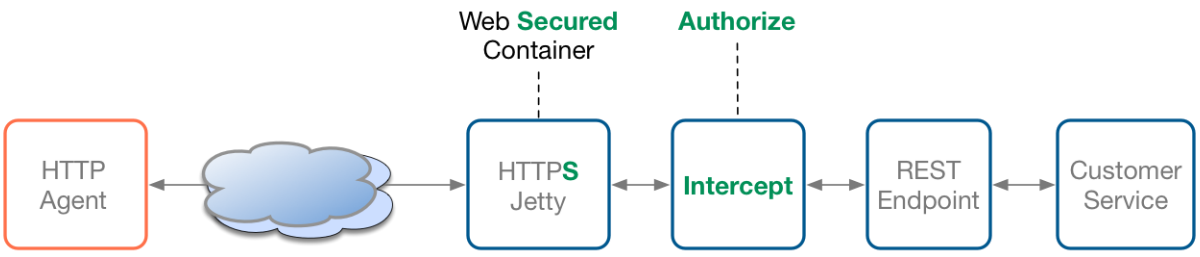


*Figure 5.3-2 The Interceptor Pattern with Secured Web container[20]*

The interceptor pattern is flexible and easy to handle. But, interceptor pattern needs to implement using custom security coding which is not a standard practice. Management of the Microservices is a burden using the interceptor pattern because each and every Microservice need to be managed independently. This is decreasing the quick commissioning and decommission capabilities of Microservices. Managing certificates with two-way TLS is also problematic with Microservices architecture.

### 5.3.2 Nordic API : Use JWT Token to secure Microservices

The Nordic API application framework implements an Identity server to perform Authentication for each Microservice. A JWT is generated using OpenID Connect Authorization server. The resource owner creates a session and persist the generated JWT in the particular user session.

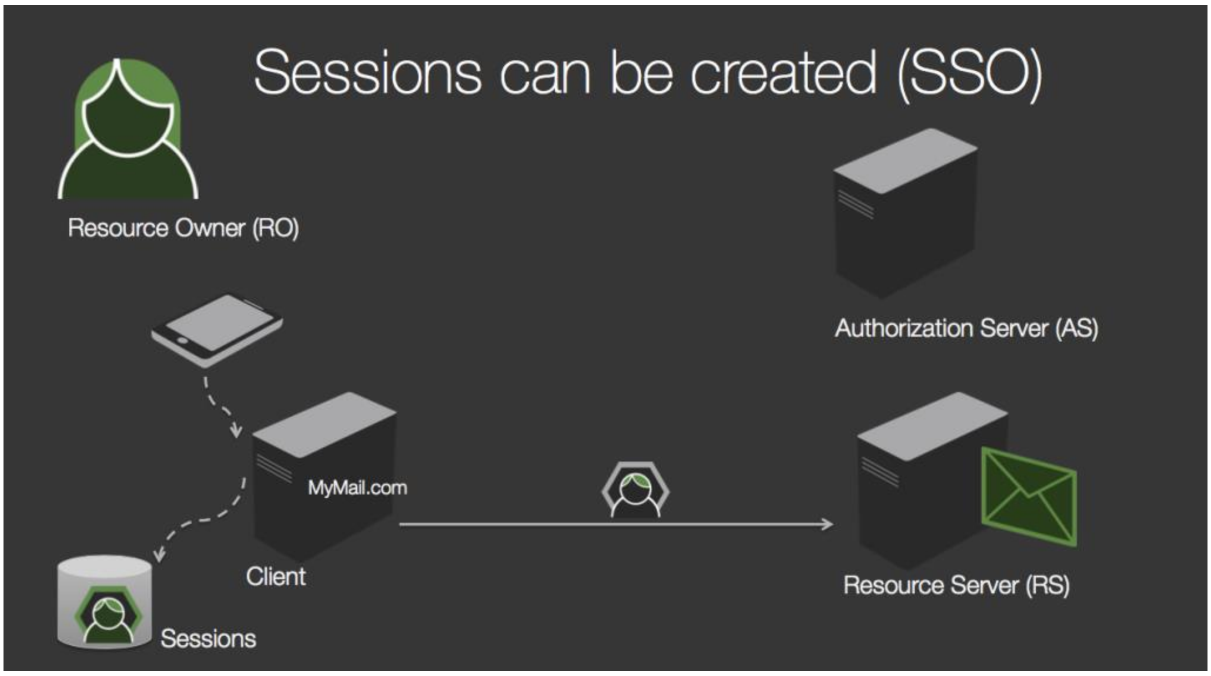


*Figure 5.3-3 Nordic API : SSO Architecture[21]*

The user is authenticated using information contained in the ID token. Creating a user session in the client side can be considered as an overhead in this approach for an open ended clients such as Netflix consumers.

The Nordic API Tokens are generated once and flowing through from one service to another. If an Adversary manage to attack the user’s session and aquire a valid JWT, the Adversary can access resources from any internal Microservice as well.

### 5.3.3 JHispter UAA for Microservice Security

JHipster UAA (User Account and Authorization ) is an authorizing service for securing microservices using the OAuth2 authorization protocol.

The JHipster defines 6 important claims to clarify a solid security solution

1. Central Authentication
2. Statelessness : this is to maintain the scalability of the Microservices architecture
3. User/Machine access distinction
4. Fine-grained access control
5. Safe from attacks
6. Scalability

The JHipster using Feign clients to secure inter-service communication within the Microservices internal calls.

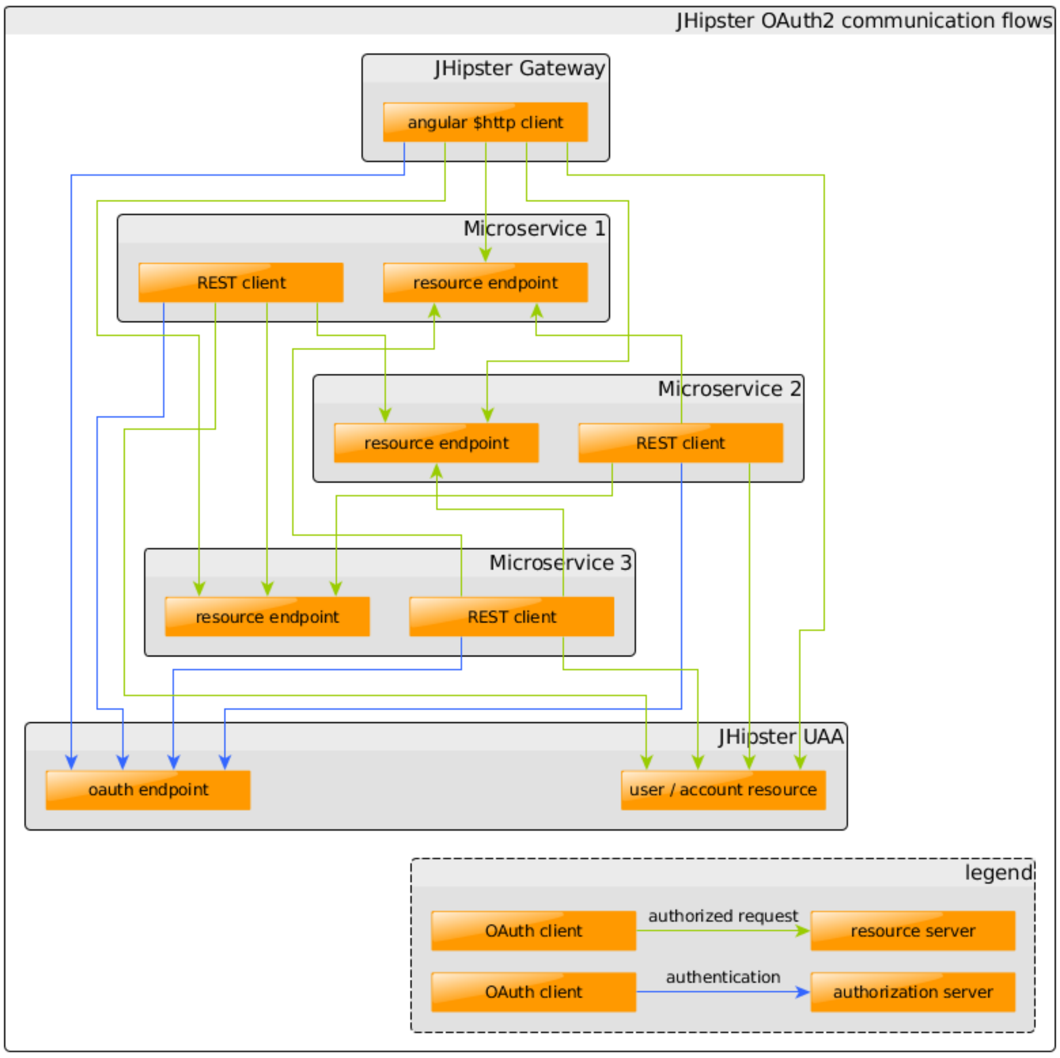


*Figure 5.3-4 JHipster OAuth Architecture[22]*

The main components of the JHipster UAA are as follows.

1. JHipster UAA server
2. At least one other microservice using UAA authentication
3. A JHipster gateway using UAA authentication

The Ribbon load balanced REST clients for endpoints registered in Eureka can be written using Feign. Feign provides with fallback implementations controlled using Hystrix, using nothing more than  Java interfaces with some annotations.

Feign clients are using to define an interface when one REST service to access another service or resource. The Interface is defined as follows.

```
@FeignClient(name = "other-service")
interface OtherServiceClient {
  @RequestMapping(value = "/api/other-resources")
  List<OtherResource> getResourcesFromOtherService();
}
@Service
class SomeService {
  private OtherServiceClient otherServiceClient;
  @Inject
  public SomeService(OtherServiceClient otherServiceClient) {
    this.otherServiceClient = otherServiceClient;
  }
}
```

The @AuthorizedFeignClients annotation enables the Authorization of the calling Microservice.

```
@AuthorizedFeignClient(name = "other-service")
interface OtherServiceClient {
  @RequestMapping(value = "/api/other-resources")
  List<OtherResource> getResourcesFromOtherService();
}
```

The JHipster UAA is a comprehensive framework to secure internal Netflix OSS Microservice infrastructure because , it is directly supporting Netflix OSS components such as Hystrix, Ribbon and Eureka . The main drawback of the JHipster UAA is it is still in its beta version. It is not an idustry best practice to use beta versioned components  in a production environment.

# 6 Conclusion

## 6.1 Summary

The Netflix OSS Microservices inherently contains vulnerabilities inside the perimeter network where an internal attacker can exploit. The major vulnerabilities are identified as Eavesdropping internal service to service communication, Confused deputy attack, Man in the middle attack and Replay attack.

The main objective of this project is to secure Netflix OSS Microservice's internal service calls from said vulnerabilities. Preserving the Microservices' characteristics such as Scalability, Performance, and Automation while securely hardening the internal service calls was also a major objective of the project.

During the literature review, various technologies were studied to find the best and optimal technique to secure Netflix OSS Microservices. The knowledge gained from those studies helped to finalize the best possible technique which satisfies project objectives.

The Netflix OSS Microservice eco system is being protected from external attacks using an OAuth Identity server implemented using Java Spring MVC. This Identity server is performing authentication of external service calls to the Edge server. An attempt was made to secure Netflix OSS Microservices from internal attacks by relaying the Edge server's authentication token into internal Microservices. But this attempt failed and it was proven that it is vulnerable to a token replay attack. Service to Service authorization also cannot be solved by using the token relaying technique.

Another attempt was made to secure the Netflix OSS Microservices' internal service to service communication by implementing .Net Identity server. The .Net Identity server issues a JWT upon successful Authentication issued by the Microservice. The same JWT is used by the called Microservice to Authorize the calling microservice. Requests from one Microservice to another was performed using HTTP. Difficulty of managing server side and client side HTTPS certificates in a dynamic environment like Microservices prevented using HTTPS communication in between service calls. But the JWT was secured by expiring the JWT upon one Authorize request and providing limited lifespan. It was proven that the token replay attacks, Man in the middle attacks and Confused deputy attacks can be avoided using the said JWT security mechanisms.

The Eavesdropping attack was prevented by encrypting the service responses from one Microservice to another. AES encryption algorithm was used along with a shared key to perform the encryption. The shared keys were protected using Java keystore.

It was proven during the testing and evaluation phase, that there were bit of a performance impact to the Netflix OSS Microservices after introducing discussed security implementations. This is caused by encryption, decryption and identity verification API calls that needs to perform because of the security enhancement.

## 6.2 Limitations

The Defense In-depth security framework for Netflix OSS Micro Services project encountered following limitations during the research and implementations phases.

- The major obstacle of this project was to find academic research papers and other reference material about the security of internal Microservice communication. The Microservice architectural pattern is relatively new concept and because of that, it was hard to find academic research papers about Microservice internal security.

- Failed to host Microservices in an industrial level containers such as Docker.
  This limitation occurred due to lack of hardware resources in the testing computer. Because of this, the Microservices had to host using Undertow containers using different ports in the localhost environment.

- Use self-signed certificate in the IIS server to secure .Net Identity Server communication.
  Using self-signed certificates is not an industrial level recommended practice. SSL certificates should be validated using a Certificate Authority according to the industrial standards. Since this is a research project, a self-signed certificate is used to secure IIS server.

## 6.3 Future Enhancements

Based on the conducted testing and evaluation, the Defence In-depth security framework for Netflix OSS Micro Services project managed to successfully secure Internal Netflix OSS Microservices. However, following areas are left open to future research and development.

- Access the .Net Identity server through Eureka server using Rabbit MQ message queue.
  In the Current implementation, Microservices are accessing the .Net identity server directly in-order to perform Authentication and Authorizations. It is a good practice to integrate the .Net identity server to the same Microservice eco system.

- Auto shutdown a Microservice upon invalid credentials provided to the .Net Identity server or Java Keystore.
  This feature makes sure that no unauthorized or malicious Microservices are not being able to keep alive in the Netflix OSS Microservice eco system.

- Auto shutdown a Microservice upon identification of malicious payload returned from inner level Microservice.
  This feature makes sure that no unauthorized or malicious Microservices are not being able to keep alive in the Netflix OSS Microservice eco system.

- After security hardening of Netflix OSS internal Microservice calls, a significant performance decrement was monitored in the Microservices ecosystem. Further research needs to be performed about increasing the performance while maintaining the high security implementation of internal Microservice ecosystems.

## References

[1] Netflix Supported Devices | Watch Netflix on your phone, TV or favorite device. 2016. Netflix Supported Devices | Watch Netflix on your phone, TV or favorite device. [ONLINE] Available at:https://devices.netflix.com/en/. [Accessed 03 May 2016]

[2] Statista. 2016. • Netflix subscribers, users 2016. [ONLINE] Available at:https://www.statista.com/statistics/250934/quarterly-number-of-netflix-streaming-subscribers-worldwide/. [Accessed 07 May 2016]

[3] Sam Newman, 2015. Building Microservices. 1 Edition. O'Reilly Media.

[4] Vinh D. Le, M. M. (2012). Micro service-based Architecture for the NRDC. International Journal of Open Information Technologies , 45.

[5] Netflix Open Source Software Center. 2016. Netflix Open Source Software Center. [ONLINE] Available at: https://netflix.github.io/. [Accessed 26 August 2016].

[6] SnapLogic Blog. 2016. Two-way SSL with SnapLogic’s REST Snap | SnapLogic Blog. [ONLINE] Available at: http://www.snaplogic.com/blog/two-way-ssl-with-snaplogics-rest-snap/. [Accessed 26 August 2016].

[7] The Open Universe: One Way and Two Way SSL and TLS. 2016. The Open Universe: One Way and Two Way SSL and TLS. [ONLINE] Available at: http://www.ossmentor.com/2015/03/one-way-and-two-way-ssl-and-tls.html. [Accessed 26 August 2016].

[8] DigitalOcean. 2016. An Introduction to OAuth 2 | DigitalOcean. [ONLINE] Available at:https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2. [Accessed 30 August 2016].

[9] Thread Safe: The problem with OAuth for Authentication.. 2016. Thread Safe: The problem with OAuth for Authentication.. [ONLINE] Available at: http://www.thread-safe.com/2012/01/problem-with-oauth-for-authentication.html. [Accessed 19 July 2016].

[10] draft-ietf-oauth-proof-of-possession-11 - Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs). 2016. draft-ietf-oauth-proof-of-possession-11 - Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs). [ONLINE] Available at: https://tools.ietf.org/html/draft-ietf-oauth-proof-of-possession-11. [Accessed 27 July 2016

[11] Netflix Open Source Software Center. 2016. Netflix Open Source Software Center. [ONLINE] Available at: https://netflix.github.io/. [Accessed 27 June 2016]

[12] Stuart Douglas. 2016. Undertow · JBoss Community. [ONLINE] Available at: http://undertow.io/. [Accessed 27 September 2016].

[13] Netflix, Inc Netflix Open Source Platform. [ONLINE] Available at: https://github.com/Netflix
[Accessed 27 September 2016].

[14]Richards, M, 2015. Microservices vs. Service- Oriented Architecture. 1st ed. United States of America: O'Reilly.

[15] auth0.com. 2017. JSON Web Token Introduction - jwt.io. [ONLINE] Available at: https://jwt.io/introduction/. [Accessed 04 October 2016]

[16] Using HMAC to authenticate Web service requests – rc3.org. 2017. Using HMAC to authenticate Web service requests – rc3.org. [ONLINE] Available at: http://rc3.org/2011/12/02/using-hmac-to-authenticate-web-service-requests/. [Accessed 14 October 2016].

[17] GitHub - kizzx2/wireshark-http-gunzip: Make Wireshark's "Follow TCP Stream" support Content-Encoding: gzip. 2017. GitHub - kizzx2/wireshark-http-gunzip: Make Wireshark's "Follow TCP Stream" support Content-Encoding: gzip. [ONLINE] Available at: https://github.com/kizzx2/wireshark-http-gunzip. [Accessed 28 January 2017].

[18] Token relay pattern: service secured with oauth to call another oauth secured service · Issue #45 · spring-cloud/spring-cloud-security · GitHub. 2017. Token relay pattern: service secured with oauth to call another oauth secured service · Issue #45 · spring-cloud/spring-cloud-security · GitHub. [ONLINE] Available at: https://github.com/spring-cloud/spring-cloud-security/issues/45. [Accessed 21 January 2017].

[19] SmartBear. 2017. Load Tester. [ONLINE] Available at: https://smartbear.com/lp/loadui-org/loadcomplete-free-tool/. [Accessed 28 February 2017].

[20] Dzone. 2017. Security enforcement of the microservices - Dzone Integration . [ONLINE] Available at: https://dzone.com/articles/security-enforcement-of-the-microservices. [Accessed 13 February 2017].

[21] Nordic API. 2017. How to Control User Ids Within Microservices . [ONLINE] Available at: http://nordicapis.com/how-to-control-user-identity-within-microservices/. [Accessed 15 February 2017].

[22] Jhipster. 2017. Using UAA for Microservice Security . [ONLINE] Available at: https://jhipster.github.io/using-uaa/. [Accessed 22 February 2017].

[23] RFC 5246 - The Transport Layer Security (TLS) Protocol Version 1.2. 2017. RFC 5246 - The Transport Layer Security (TLS) Protocol Version 1.2. [ONLINE] Available at: https://tools.ietf.org/html/rfc5246?as_url_id=AAAAAAXTxRCKJXm4F2HWdjCh4Jxj1h RuOCJYqyzGnd9MPhztwQSso2IIm22QlXESwpr2rmXMwttJ33VZo2pBnBHDKnuQ. [Accessed 04 March 2017]

[24] NGINX. 2017. Introduction to Microservices | NGINX. [ONLINE] Available at: https://www.nginx.com/blog/introduction-to-microservices/. [Accessed 07 March 2017].

[25] Uber Engineering Blog. 2017. The Uber Engineering Tech Stack, Part I: The Foundation - Uber Engineering Blog. [ONLINE] Available at: https://eng.uber.com/tech-stack-part-one/. [Accessed 07 March 2017].

[26] MikeWasson. 2017. Authentication Filters in ASP.NET Web API 2 | Microsoft Docs. [ONLINE] Available at: https://docs.microsoft.com/en-us/aspnet/web-api/overview/security/authentication-filters. [Accessed 08 March 2017].

[27] hongyes. 2017. OWIN OAuth 2.0 Authorization Server | Microsoft Docs. [ONLINE] Available at: https://docs.microsoft.com/en-us/aspnet/aspnet/overview/owin-and-katana/owin-oauth-20-authorization-server. [Accessed 08 March 2017].

[28] Uber's Business Model. 2017. Uber's Business Model. [ONLINE] Available at: https://www.slideshare.net/funk97/ubers-business-model. [Accessed 08 March 2017].

[29] Ebay history and architecture - High Scalability - . 2017. Ebay history and architecture - High Scalability - . [ONLINE] Available at: http://highscalability.com/blog/2009/3/31/ebay-history-and-architecture.html. [Accessed 08 March 2017].

[30] Eric Knorr. 2017. What eBay looks like under the hood. [ONLINE] Available at: http://www.infoworld.com/article/3041064/application-development/what-ebay-looks-like-under-the-hood.html. [Accessed 8 March 2017].

[31] Front-end Renaissance at eBay | eBay Tech Blog. 2017. Front-end Renaissance at eBay | eBay Tech Blog. [ONLINE] Available at: http://www.ebaytechblog.com/2014/01/13/front-end-renaissance-at-ebay/. [Accessed 08 March 2017].

[32] Microservices at eBay. 2017. Microservices at eBay. [ONLINE] Available at: https://de.slideshare.net/kasun04/microservices-at-ebay. [Accessed 08 March 2017]

[33] Postman - Chrome Web Store. 2017. Postman - Chrome Web Store. [ONLINE] Available at: https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjbdggehcddcbncdddomop. [Accessed 04 June 2017]

[34] OpenSSL Foundation, Inc.. 2017. /index.html . [ONLINE] Available at: https://www.openssl.org/. [Accessed 04 June 2017].

# Appendix A : Source Code

## Appendix A1

```
import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

import se.callista.microservices.composite.product.model.ProductAggregated;

import se.callista.microservises.core.product.model.Product;

import se.callista.microservises.core.recommendation.model.Recommendation;

import se.callista.microservises.core.review.model.Review;


import java.util.Date;

import java.util.List;


/**
 * Created by The Hacker
 */
@RestController
public class CounterfeitCompositeService {


    private static final Logger LOG = LoggerFactory.getLogger(CounterfeitCompositeService.class);
    @Autowired
    ProductCompositeIntegration integration;
    @Autowired
    Util util;


   @RequestMapping("/")
    public String getProduct() {
        return "{\"timestamp\":\"" + new Date() + "\",\"content\":\"I am a legitimate Microservice HA HA ... \"}";
    }
    @RequestMapping("/product/{productId}")
    public ResponseEntity<ProductAggregated> getProduct(@PathVariable int productId) {
        //Get the product id from the incoming request and relay to the internal services
        ResponseEntity<Product> productResult = integration.getProduct(productId);


        // 2. Get optional recommendations
        List<Recommendation> recommendations = null;
        try {
            ResponseEntity<List<Recommendation>> recommendationResult = 
integration.getRecommendations(productId);
            recommendations = recommendationResult.getBody();
                                        // I am a bad person and I need to see whats  coming  to my bad service
                                        // So I do log the response from internal service
                                         LOG.debug(recommendations);


                                         // I need to change the original content and need my content to return
                                         String myBadRecomendation = "{\"recommendationId\":\"1\" ,\"author\":\"The 
Hacker \" ,\"rate\":\"-1 \"}"


        } catch (Throwable t) {
            LOG.error("Somethig wrong here  ", t);
            throw t;
        }
        }
// Returning my bad content
        return util.createOkResponse(new ProductAggregated(productResult.getBody(), recommendations, 
reviews));
    }
}
```

## Appendix A2

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.oauth2.config.annotation.configurers.ClientDetailsServiceConfigurer;
import org.springframework.security.oauth2.config.annotation.web.configuration.AuthorizationServerConfigurerAdapter;
import org.springframework.security.oauth2.config.annotation.web.configuration.EnableAuthorizationServer;
import org.springframework.security.oauth2.config.annotation.web.configuration.EnableResourceServer;
import org.springframework.security.oauth2.config.annotation.web.configurers.AuthorizationServerEndpointsConfigurer;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.security.Principal;

@SpringBootApplication
@RestController
@EnableResourceServer
public class AuthserverApplication {

	@RequestMapping("/user")
	public Principal user(Principal user) {
		return user;
	}

	public static void main(String[] args) {
		SpringApplication.run(AuthserverApplication.class, args);
	}

	@Configuration
	@EnableAuthorizationServer
	protected static class OAuth2Config extends AuthorizationServerConfigurerAdapter {

		@Autowired
		private AuthenticationManager authenticationManager;

		@Override
		public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {
			endpoints.authenticationManager(authenticationManager);
		}

		@Override
		public void configure(ClientDetailsServiceConfigurer clients) throws Exception {
			clients.inMemory()
				.withClient("acme")
				.secret("acmesecret")
```

```
                                        .authorizedGrantTypes("authorization_code", "refresh_token", "implicit",
"password", "client_credentials")

                                        .scopes("webshop");

                    }

          }

}
```

## Appendix A3

```
@Autowired

   private LoadBalancerClient loadBalancer;

   @RequestMapping("/{productId}")

   @HystrixCommand(fallbackMethod = "defaultProductComposite")

   public ResponseEntity<String> getProductComposite(

      @PathVariable int productId,

      @RequestHeader(value="Authorization") String authorizationHeader,

      Principal currentUser) {

      URI uri = loadBalancer.choose("productcomposite").getUri();

      String url = uri.toString() + "/product/" + productId;

      ResponseEntity<String> result = restTemplate.getForEntity(url, String.class);

      return result;

   }
```

## Appendix A4

```
namespace MIS023.IdentityServer.Controllers
{
    [RoutePrefix("api/claims")]
    public class ClaimsController : BaseApiController
    {
        [Authorize]
        [Route("")]
        public IHttpActionResult GetClaims()
        {
            var identity = User.Identity as ClaimsIdentity;

            var claims = from c in identity.Claims
                         select new
                         {
                             subject = c.Subject.Name,
                             type = c.Type,
                             value = c.Value
                         };
```

```
                return Ok(claims);
            }

        }
}
```

## Appendix A5

```
private String Authenticate(String userName, String password)
  {
     try (CloseableHttpClient httpClient = HttpClientBuilder.create().build()) {
        HttpPost request = new HttpPost("http://localhost:55471/oauth/" + "token");
        request.addHeader("content-type", "application/json");
        request.addHeader("Accept", "application/json");

        List < NameValuePair > params = new ArrayList<NameValuePair>(2);
        params.add(new BasicNameValuePair("username", userName));
        params.add(new BasicNameValuePair("password", password ));
        params.add(new BasicNameValuePair("grant_type", "password"));
        request.setEntity(new UrlEncodedFormEntity(params, "UTF-8"));

        HttpResponse result = httpClient.execute(request);
        String json = EntityUtils.toString(result.getEntity(), "UTF-8");

        if(json.contains("invalid_grant"))
           return "Access Denied";
        com.google.gson.Gson gson = new com.google.gson.Gson();
        Token token = gson.fromJson(json, Token.class);
        LOG.info(token.getAccess_token());
        return token.getAccess_token
  } catch (Exception ex) {
     LOG.info("exception :" + ex.getMessage());
                return "Token Exception";
  }
  }
```

## Appendix A6

```
class Token
{
    private String access_token;
    private String token_type;
    private String expires_in;

   public String getAccess_token() {
      return access_token;
   }
```

```
    public void setAccess_token(String access_token) {
        this.access_token = access_token;
    }

    public String getToken_type() {
        return token_type;
    }
    public void setToken_type(String token_type) {
        this.token_type = token_type;
    }
    public String getExpires_in() {
        return expires_in;
    }
    public void setExpires_in(String expires_in) {
        this.expires_in = expires_in;
    }
}
```

## Appendix A7

```
public class Claim{
    private String subject;
    private String type;
    private String value;

    public String getSubject() {
        return subject;
    }

    public void setSubject(String subject) {
        this.subject = subject;
    }
    public String getType() {
        return type;
    }
    public void setType(String type) {
        this.type = type;
    }
    public String getValue() {
        return value;
    }
    public void setValue(String value) {
        this.value = value;
    }
    public String getRole(String token){
    String role=null;
    try (CloseableHttpClient httpClient = HttpClientBuilder.create().build()) {
        HttpGet claimsRequest = new HttpGet("https://localhost:55471/api/"+"claims");
        claimsRequest.addHeader("Content-Type", "application/json");
        claimsRequest.addHeader("Accept", "application/json");
        claimsRequest.addHeader("Authorization", "Bearer " + token);

        HttpResponse claimsResult = httpClient.execute(claimsRequest);
        String jsonClaims = EntityUtils.toString(claimsResult.getEntity(), "UTF-8");

        com.google.gson.Gson gson = new com.google.gson.Gson();
        if(jsonClaims.equals("{\"message\":\"Authorization has been denied for this
request.\"}"))
            return "403";
        Claim[] claimResponse = gson.fromJson(jsonClaims, Claim[].class);

        for (Claim item : claimResponse) {

if("http://schemas.microsoft.com/ws/2008/06/identity/claims/role".equals(item.type))
            {
                role = item.value; break;
            }
        }
    } catch (Exception ex) {
        return null;
    }
    return role;
    }
}
```

## Appendix A8

```
namespace MIS023.IdentityServer.Controllers
{
    [RoutePrefix("api/accounts")]
    public class AccountsController : BaseApiController
    {

        [Authorize(Roles= "SuperAdmin")]
        [Route("users")]
        public IHttpActionResult GetUsers()
        {
            //Only SuperAdmin or Admin can delete users (Later when implement roles)
            var identity = User.Identity as System.Security.Claims.ClaimsIdentity;

            return Ok(this.AppUserManager.Users.ToList().Select(u =>
this.TheModelFactory.Create(u)));
        }

        [Route("RoleByUser/{id:guid}", Name = "RoleByUser")]
        public async Task<IHttpActionResult> GetRoleForUser(string Id)
        {
            IList<string> roles = this.AppUserManager.GetRolesAsync(Id).Result;

            if (roles.Count > 0)
            {
               return Ok(roles[0]);
            }

            return NotFound();

        }

        [Authorize(Roles = "SuperAdmin")]
        [Route("users/{id:guid}", Name = "GetUserById")]
        public async Task<IHttpActionResult> GetUser(string Id)
        {
            //Only SuperAdmin or Admin can delete users (Later when implement roles)
            var user = await this.AppUserManager.FindByIdAsync(Id);

            if (user != null)
            {
               return Ok(this.TheModelFactory.CreateWithFirstLastName(user));
            }

            return NotFound();

        }

        [Authorize(Roles = "SuperAdmin")]
        [Route("users/{username}")]
        public async Task<IHttpActionResult> GetUserByName(string username)
        {
            //Only SuperAdmin or Admin can delete users (Later when implement roles)
            var user = await this.AppUserManager.FindByNameAsync(username);

            if (user != null)
            {
               return Ok(this.TheModelFactory.Create(user));
            }

            return NotFound();

        }
```

```
    [AllowAnonymous]
    [HttpPost]
    [Route("users")]
    [Authorize(Roles = "SuperAdmin")]
    public async Task<IHttpActionResult> CreateUser(CreateUserBindingModel
createUserModel)
    {

      if (!ModelState.IsValid)
      {
        return BadRequest(ModelState);
      }

      var user = new ApplicationUser()
      {
        UserName = createUserModel.Username,
        Email = createUserModel.Email,
        FirstName = createUserModel.FirstName,
        LastName = createUserModel.LastName,
        Level = 3,
        JoinDate = DateTime.Now.Date,
      };



      IdentityResult addUserResult = await this.AppUserManager.CreateAsync(user,
createUserModel.Password);

      if (!addUserResult.Succeeded)
      {
        return GetErrorResult(addUserResult);
      }

      string code = await
this.AppUserManager.GenerateEmailConfirmationTokenAsync(user.Id);

      var callbackUrl = new Uri(Url.Link("ConfirmEmailRoute", new { userId = user.Id, code
= code }));

      await this.AppUserManager.SendEmailAsync(user.Id,
                                  "Confirm your account",
                                  "Please confirm your account by clicking <a href=\"" +
callbackUrl + "\">here</a>");

      Uri locationHeader = new Uri(Url.Link("GetUserById", new { id = user.Id }));
```

```
            return Created(locationHeader, TheModelFactory.Create(user));

        }

        [AllowAnonymous]
        [HttpGet]
        [Route("ConfirmEmail", Name = "ConfirmEmailRoute")]
        public async Task<IHttpActionResult> ConfirmEmail(string userId = "", string code = "")
        {
            if (string.IsNullOrWhiteSpace(userId) || string.IsNullOrWhiteSpace(code))
            {
                ModelState.AddModelError("", "User Id and Code are required");
                return BadRequest(ModelState);
            }

            IdentityResult result = await this.AppUserManager.ConfirmEmailAsync(userId, code);

            if (result.Succeeded)
            {
                return Ok();
            }
            else
            {
                return GetErrorResult(result);
            }
        }

        [Authorize]
        [Route("ChangePassword")]
        public async Task<IHttpActionResult> ChangePassword(ChangePasswordBindingModel
model)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            IdentityResult result = await
this.AppUserManager.ChangePasswordAsync(User.Identity.GetUserId(),
model.OldPassword, model.NewPassword);

            if (!result.Succeeded)
            {
                return GetErrorResult(result);
            }

            return Ok();
        }



        [AllowAnonymous]
        [Route("resetpasswordmail")]
        public async Task<IHttpActionResult>
ResetPasswordMail(ResetPasswordEmailBindingModel model)
        {
            try
            {
                if (model == null)
                {
                    return BadRequest(ModelState);
                }

                string resetPasswdLink =
ConfigurationManager.AppSettings["ResetPasswordLink"];

                ApplicationUser user = await AppUserManager.FindByEmailAsync(model.Email);

                var resetToken =
this.AppUserManager.GeneratePasswordResetToken(user.Id.ToString());

                string  callbackUrl =  resetPasswdLink + string.Format( "?userid={0}&token={1}",
user.Id, resetToken);

                await this.AppUserManager.SendEmailAsync(user.Id,
                    "Reset your password",
                    "Please reset  your password by clicking <a href=\"" + callbackUrl +
"\">here</a>");

                //if (!result.Succeeded)
                //{
                //    return GetErrorResult(result);
                //}

                return Ok();
            }
            catch (Exception ex)
            {
                return InternalServerError();
            }
        }

        [AllowAnonymous]
        [Route("resetpassword")]
        public async Task<IHttpActionResult> ResetPassword(ResetPasswordBindingModel
model)
        {
            try
            {
                if (model == null)
                {
                    return BadRequest(ModelState);
                }

             var result =  await  this.AppUserManager.ResetPasswordAsync(model.UserId,
model.Token, model.Password);
                if (result.Succeeded)
                    return Ok();
                else
                    return BadRequest(result.Errors.FirstOrDefault());
            }
            catch (Exception ex)
            {
                return InternalServerError();
            }
        }

        [Authorize(Roles = "SuperAdmin")]
        [Route("users/{id:guid}")]
        public async Task<IHttpActionResult> DeleteUser(string id)
        {

            //Only SuperAdmin or Admin can delete users (Later when implement roles)

            var appUser = await this.AppUserManager.FindByIdAsync(id);

            if (appUser != null)
            {
                IdentityResult result = await this.AppUserManager.DeleteAsync(appUser);

                if (!result.Succeeded)
                {
                    return GetErrorResult(result);
                }

                return Ok();

            }

            return NotFound();
        }

        [Authorize(Roles = "SuperAdmin")]
        [Route("users/{id:guid}/roles")]
        [HttpPut]
        public async Task<IHttpActionResult> AssignRolesToUser([FromUri] string id,
[FromBody] RolesModel roles)
        {
            string[] rolesToAssign = new string[1];
            rolesToAssign[0] = roles.Roles;
            var appUser = await this.AppUserManager.FindByIdAsync(id);

            if (appUser == null)
            {
                return NotFound();
            }

            var currentRoles = await this.AppUserManager.GetRolesAsync(appUser.Id);

            var rolesNotExists = rolesToAssign.Except(this.AppRoleManager.Roles.Select(x =>
x.Name)).ToArray();

            if (rolesNotExists.Count() > 0)
            {

                ModelState.AddModelError("", string.Format("Roles '{0}' does not exixts in the
system", string.Join(",", rolesNotExists)));
                return BadRequest(ModelState);
            }

            IdentityResult removeResult = await
this.AppUserManager.RemoveFromRolesAsync(appUser.Id, currentRoles.ToArray());

            if (!removeResult.Succeeded)
            {
                ModelState.AddModelError("", "Failed to remove user roles");
                return BadRequest(ModelState);
            }

            IdentityResult addResult = await this.AppUserManager.AddToRolesAsync(appUser.Id,
rolesToAssign);

            if (!addResult.Succeeded)
            {
                ModelState.AddModelError("", "Failed to add user roles");
                return BadRequest(ModelState);
            }
```

```
          return Ok();

        }

        [Authorize(Roles = "SuperAdmin")]
        [Route("users/{id:guid}/assignclaims")]
        [HttpPut]
        public async Task<IHttpActionResult> AssignClaimsToUser([FromUri] string id,
[FromBody] List<ClaimBindingModel> claimsToAssign)
        {

            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            var appUser = await this.AppUserManager.FindByIdAsync(id);

            if (appUser == null)
            {
                return NotFound();
            }

            RemoveClientIdClaim(id);

            foreach (ClaimBindingModel claimModel in claimsToAssign)
            {
                if (appUser.Claims.Any(c => c.ClaimType == claimModel.Type))
                {

                    await this.AppUserManager.RemoveClaimAsync(id,
ExtendedClaimsProvider.CreateClaim(claimModel.Type, claimModel.Value));
                }

                await this.AppUserManager.AddClaimAsync(id,
ExtendedClaimsProvider.CreateClaim(claimModel.Type, claimModel.Value));
            }

            return Ok();
        }

        [Authorize(Roles = "SuperAdmin")]
        [Route("users/{id:guid}/removeclaims")]
        [HttpPut]
        public async Task<IHttpActionResult> RemoveClaimsFromUser([FromUri] string id,
[FromBody] List<ClaimBindingModel> claimsToRemove)
```

```
        {

            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            var appUser = await this.AppUserManager.FindByIdAsync(id);

            if (appUser == null)
            {
                return NotFound();
            }

            foreach (ClaimBindingModel claimModel in claimsToRemove)
            {
                if (appUser.Claims.Any(c => c.ClaimType == claimModel.Type))
                {
                    await this.AppUserManager.RemoveClaimAsync(id,
ExtendedClaimsProvider.CreateClaim(claimModel.Type, claimModel.Value));
                }
            }

            return Ok();
        }

        private  void RemoveClientIdClaim(string id)
        {
            var appUser =  this.AppUserManager.FindById(id);

            foreach (var claim in appUser.Claims.Where(x=> x.ClaimType == "ClientId"))
            {
                 this.AppUserManager.RemoveClaimAsync(id,
ExtendedClaimsProvider.CreateClaim(claim.ClaimType, claim.ClaimValue));
            }
        }

    }
}
```

## Appendix A9

```
[assembly: OwinStartup(typeof(MIS023.IdentityServer.Startup))]
namespace MIS023.IdentityServer
{
    public class Startup
    {
```

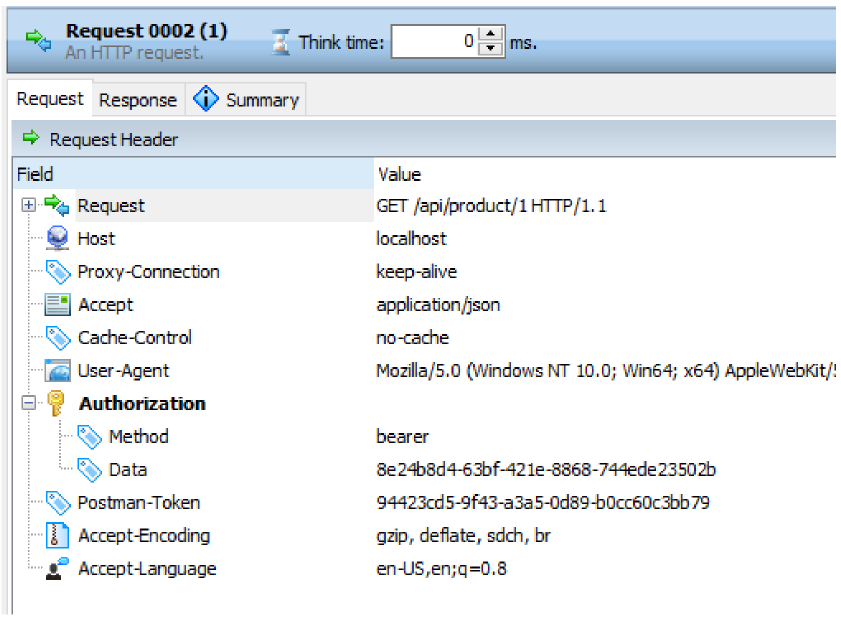
```
    public void Configuration(IAppBuilder app)
    {
        HttpConfiguration httpConfig = new HttpConfiguration();

        ConfigureOAuthTokenGeneration(app);

        ConfigureOAuthTokenConsumption(app);

        ConfigureWebApi(httpConfig);

        app.UseCors(Microsoft.Owin.Cors.CorsOptions.AllowAll);

        app.UseWebApi(httpConfig);

    }

    private void ConfigureOAuthTokenGeneration(IAppBuilder app)
    {
        // Configure the db context and user manager to use a single instance per request
        app.CreatePerOwinContext(ApplicationDbContext.Create);
        app.CreatePerOwinContext<ApplicationUserManager>(ApplicationUserManager.Create);
        app.CreatePerOwinContext<ApplicationRoleManager>(ApplicationRoleManager.Create);

        OAuthAuthorizationServerOptions OAuthServerOptions = new
OAuthAuthorizationServerOptions()
        {
            //For Dev enviroment only (on production should be AllowInsecureHttp = false)
            AllowInsecureHttp = true,
            TokenEndpointPath = new PathString("/oauth/token"),
            AccessTokenExpireTimeSpan = TimeSpan.FromDays(30),
            Provider = new CustomOAuthProvider(),
            AccessTokenFormat = new
CustomJwtFormat(ConfigurationManager.AppSettings["as:AuthServerApi"])
        };

        // OAuth 2.0 Bearer Access Token Generation
        app.UseOAuthAuthorizationServer(OAuthServerOptions);
    }

    private void ConfigureOAuthTokenConsumption(IAppBuilder app)
    {

        var issuer = ConfigurationManager.AppSettings["as:AuthServerApi"];
        string audienceId = ConfigurationManager.AppSettings["as:AudienceId"];
        byte[] audienceSecret =
TextEncodings.Base64Url.Decode(ConfigurationManager.AppSettings["as:AudienceSecret"]);

        // Api controllers with an [Authorize] attribute will be validated with JWT
        app.UseJwtBearerAuthentication(
            new JwtBearerAuthenticationOptions
            {
            AuthenticationMode = AuthenticationMode.Active,
            AllowedAudiences = new[] { audienceId },
            IssuerSecurityTokenProviders = new IIssuerSecurityTokenProvider[]
            {
                new SymmetricKeyIssuerSecurityTokenProvider(issuer, audienceSecret)
            }
            });
    }

    private void ConfigureWebApi(HttpConfiguration config)
    {
        config.MapHttpAttributeRoutes();

        var jsonFormatter = config.Formatters.OfType<JsonMediaTypeFormatter>().First();
        jsonFormatter.SerializerSettings.ContractResolver = new
CamelCasePropertyNamesContractResolver();
    }
}
}
```

## Appendix A10

```

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.io.Serializable;
import java.security.Key;
import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.UnrecoverableKeyException;
import java.security.cert.CertificateException;
import javax.crypto.SealedObject;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;
import javax.crypto.Cipher;

public class Crypto {
    public SecretKey GetKey(String encodedKey){

        try{
        byte[] decodedKey = Base64.getDecoder().decode(encodedKey);
        // rebuild key using SecretKeySpec
        SecretKey key = new SecretKeySpec(decodedKey, 0, decodedKey.length, "AES");
        return key;
        }catch(Exception ex){return null;}

    }

            public static String Encrypt(String str, SecretKey key) throws Exception {
        byte[] utf8 = str.getBytes("UTF8");
        Cipher ecipher = Cipher.getInstance("AES");
        ecipher.init(Cipher.ENCRYPT_MODE, key);
        byte[] enc = ecipher.doFinal(utf8);
        return new sun.misc.BASE64Encoder().encode(enc);
    }

            public static String Decrypt(String str,SecretKey key) throws Exception {
        byte[] dec = new sun.misc.BASE64Decoder().decodeBuffer(str);
        Cipher dcipher = Cipher.getInstance("AES");
        dcipher.init(Cipher.DECRYPT_MODE, key);
        byte[] utf8 = dcipher.doFinal(dec);
        return new String(utf8, "UTF8");
    }

    public SecretKey GetKeyStoreKey(String alias, String keyPass, String keystorePass  ) throws
FileNotFoundException, IOException, NoSuchAlgorithmException, CertificateException,
KeyStoreException, UnrecoverableKeyException
    {
        Key key = null;

        String location = "C:\\Program Files\\Java\\jre1.8.0_101\\bin\\aes-keystore.jck";
        InputStream keystoreStream = new FileInputStream(location);

        KeyStore keystore = KeyStore.getInstance("JCEKS");
        keystore.load(keystoreStream, keystorePass.toCharArray());

        if (!keystore.containsAlias(alias)) {
            throw new RuntimeException("Alias for key not found");
        }

        key = keystore.getKey(alias, keyPass.toCharArray()); int i=0;

        return (SecretKey) key;
    }
}
```

# Appendix B

## Appendix B1



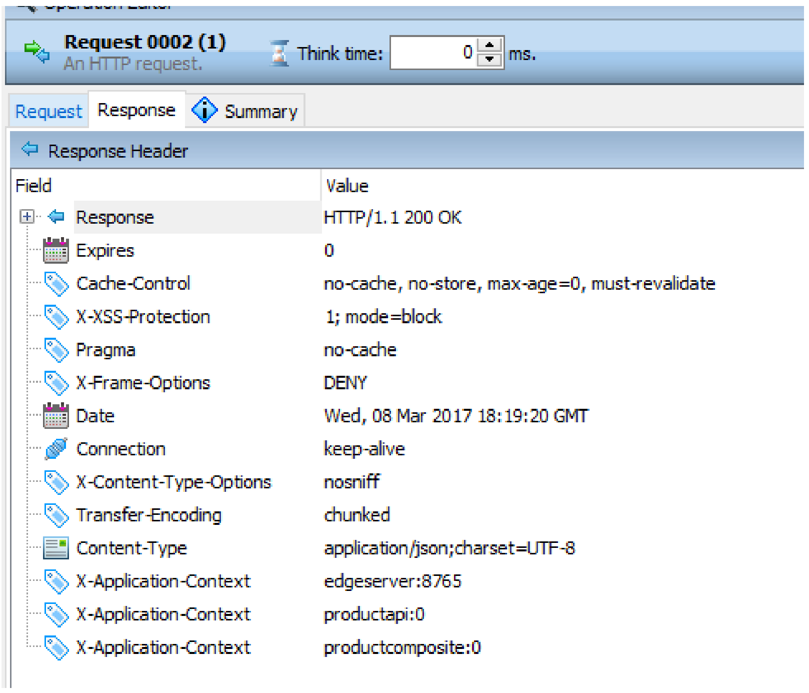
Request 0002 (1)
An HTTP request.
Think time: 0 ms.

Request | Response | Summary

Request Header

| Field | Value |
|---|---|
| Request | GET /api/product/1 HTTP/1.1 |
| Host | localhost |
| Proxy-Connection | keep-alive |
| Accept | application/json |
| Cache-Control | no-cache |
| User-Agent | Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/ |
| **Authorization** | |
| Method | bearer |
| Data | 8e24b8d4-63bf-421e-8868-744ede23502b |
| Postman-Token | 94423cd5-9f43-a3a5-0d89-b0cc60c3bb79 |
| Accept-Encoding | gzip, deflate, sdch, br |
| Accept-Language | en-US,en;q=0.8 |

Load Complete Test Request



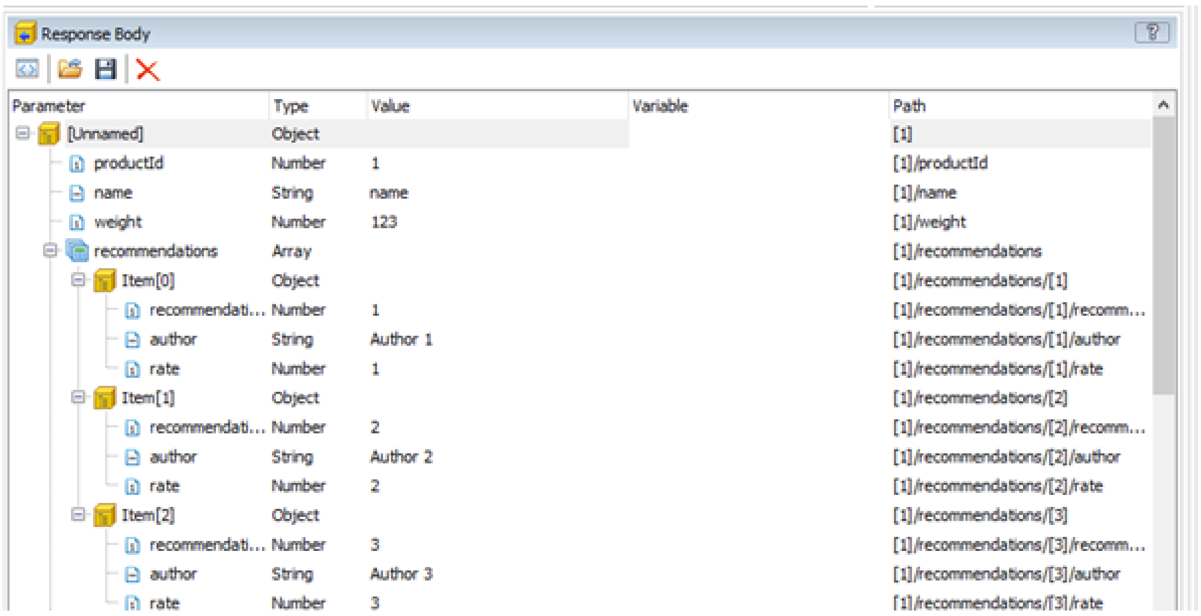
Request 0002 (1)
An HTTP request.
Think time: 0 ms.

Request | Response | Summary

Response Header

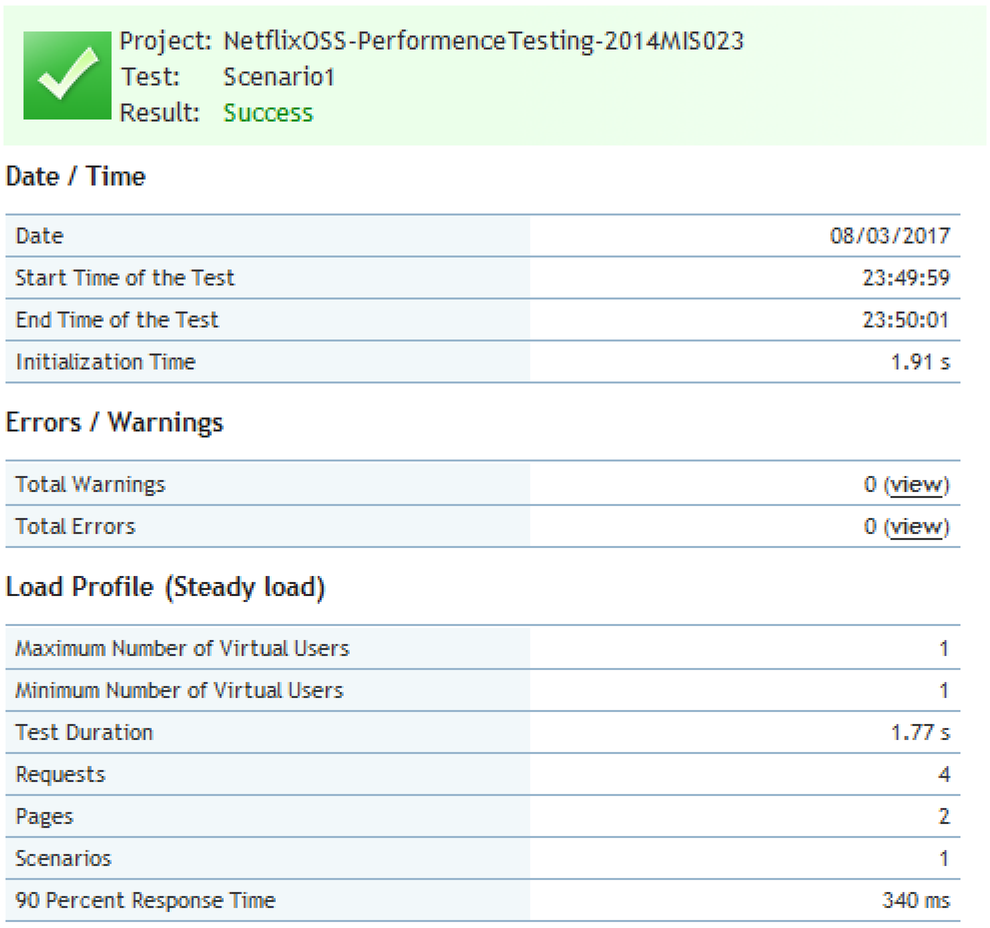
| Field | Value |
|---|---|
| Response | HTTP/1.1 200 OK |
| Expires | 0 |
| Cache-Control | no-cache, no-store, max-age=0, must-revalidate |
| X-XSS-Protection | 1; mode=block |
| Pragma | no-cache |
| X-Frame-Options | DENY |
| Date | Wed, 08 Mar 2017 18:19:20 GMT |
| Connection | keep-alive |
| X-Content-Type-Options | nosniff |
| Transfer-Encoding | chunked |
| Content-Type | application/json;charset=UTF-8 |
| X-Application-Context | edgeserver:8765 |
| X-Application-Context | productapi:0 |
| X-Application-Context | productcomposite:0 |

Load Complete Test Response



| Parameter | Type | Value | Variable | Path |
|---|---|---|---|---|
| [Unnamed] | Object | | | [1] |
| productId | Number | 1 | | [1]/productId |
| name | String | name | | [1]/name |
| weight | Number | 123 | | [1]/weight |
| recommendations | Array | | | [1]/recommendations |
| Item[0] | Object | | | [1]/recommendations/[1] |
| recommendati... | Number | 1 | | [1]/recommendations/[1]/recomm... |
| author | String | Author 1 | | [1]/recommendations/[1]/author |
| rate | Number | 1 | | [1]/recommendations/[1]/rate |
| Item[1] | Object | | | [1]/recommendations/[2] |
| recommendati... | Number | 2 | | [1]/recommendations/[2]/recomm... |
| author | String | Author 2 | | [1]/recommendations/[2]/author |
| rate | Number | 2 | | [1]/recommendations/[2]/rate |
| Item[2] | Object | | | [1]/recommendations/[3] |
| recommendati... | Number | 3 | | [1]/recommendations/[3]/recomm... |
| author | String | Author 3 | | [1]/recommendations/[3]/author |
| rate | Number | 3 | | [1]/recommendations/[3]/rate |

Load Complete Test Response Body



Project: NetflixOSS-PerformenceTesting-2014MIS023
Test: Scenario1
Result: Success

**Date / Time**

| | |
|---|---|
| Date | 08/03/2017 |
| Start Time of the Test | 23:49:59 |
| End Time of the Test | 23:50:01 |
| Initialization Time | 1.91 s |

**Errors / Warnings**

| | |
|---|---|
| Total Warnings | 0 (view) |
| Total Errors | 0 (view) |

**Load Profile (Steady load)**

| | |
|---|---|
| Maximum Number of Virtual Users | 1 |
| Minimum Number of Virtual Users | 1 |
| Test Duration | 1.77 s |
| Requests | 4 |
| Pages | 2 |
| Scenarios | 1 |
| 90 Percent Response Time | 340 ms |

### Scenario Completion Time

| | |
|---|---|
| Average | 1.77 s |
| Maximum | 1.77 s |
| Minimum | 1.77 s |

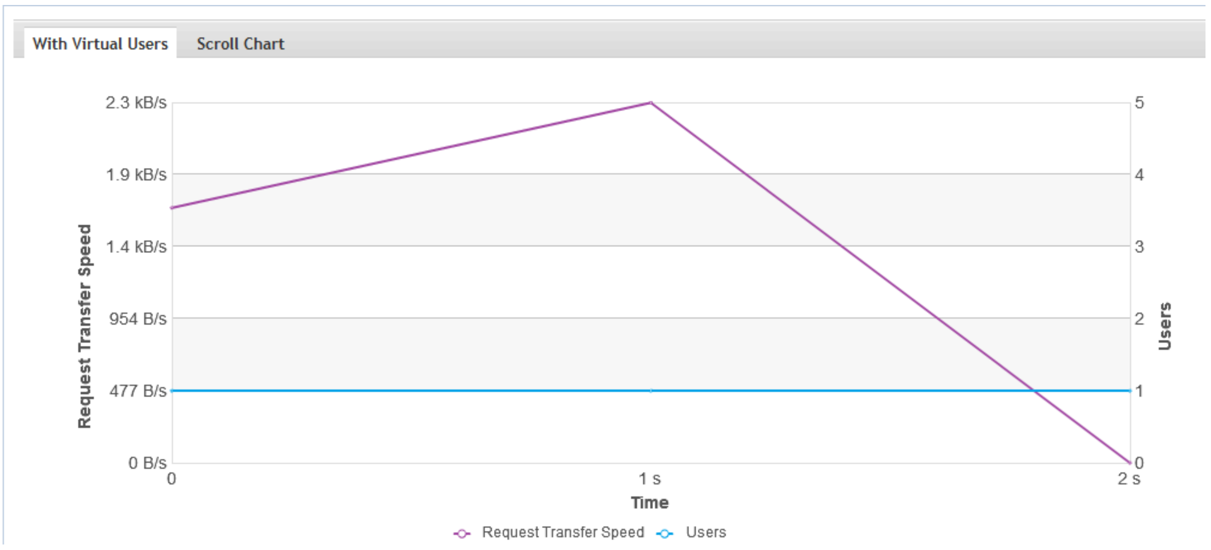
### Page Load Time

| | |
|---|---|
| Average | 886 ms |
| Maximum | 1.51 s |
| Minimum | 262 ms |

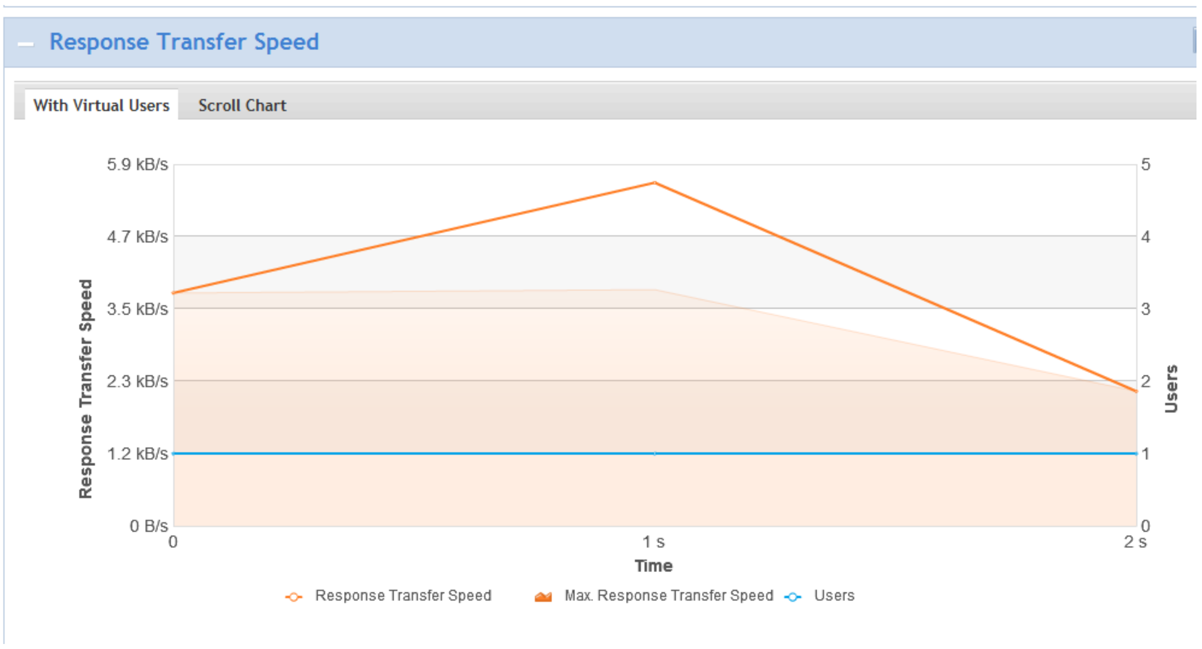
### Time to First Byte

| | |
|---|---|
| Average | 224 ms |
| Maximum | 393 ms |
| Minimum | 50 ms |

Scenario1 Test results report

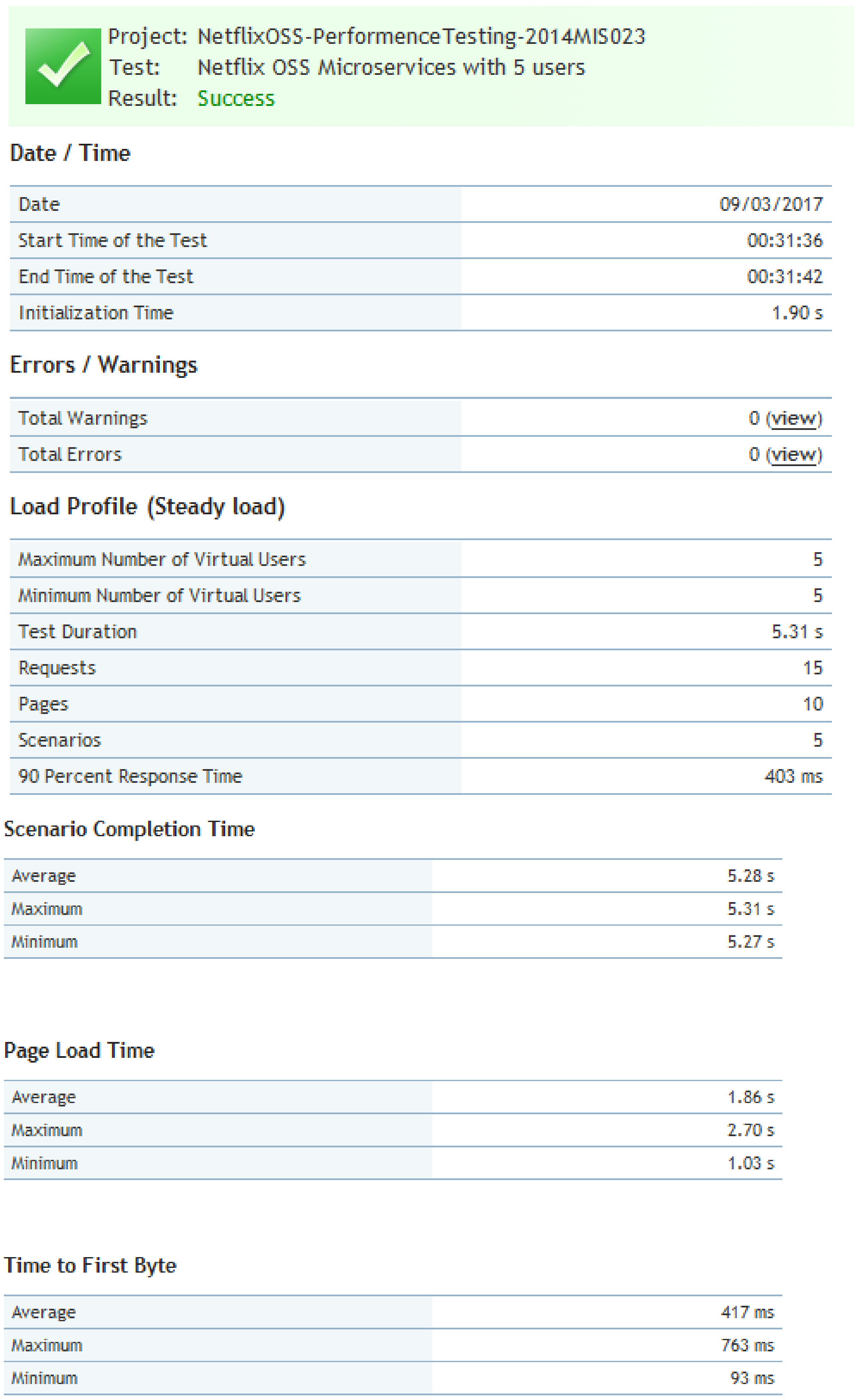


Scenario 1 Request Transfer Speed



Scenario 1 Response Transfer Speed

## Appendix B2



Project: NetflixOSS-PerformenceTesting-2014MIS023
Test: Netflix OSS Microservices with 5 users
Result: Success

**Date / Time**

| | |
|---|---|
| Date | 09/03/2017 |
| Start Time of the Test | 00:31:36 |
| End Time of the Test | 00:31:42 |
| Initialization Time | 1.90 s |

**Errors / Warnings**

| | |
|---|---|
| Total Warnings | 0 (view) |
| Total Errors | 0 (view) |

**Load Profile (Steady load)**

| | |
|---|---|
| Maximum Number of Virtual Users | 5 |
| Minimum Number of Virtual Users | 5 |
| Test Duration | 5.31 s |
| Requests | 15 |
| Pages | 10 |
| Scenarios | 5 |
| 90 Percent Response Time | 403 ms |

**Scenario Completion Time**

| | |
|---|---|
| Average | 5.28 s |
| Maximum | 5.31 s |
| Minimum | 5.27 s |

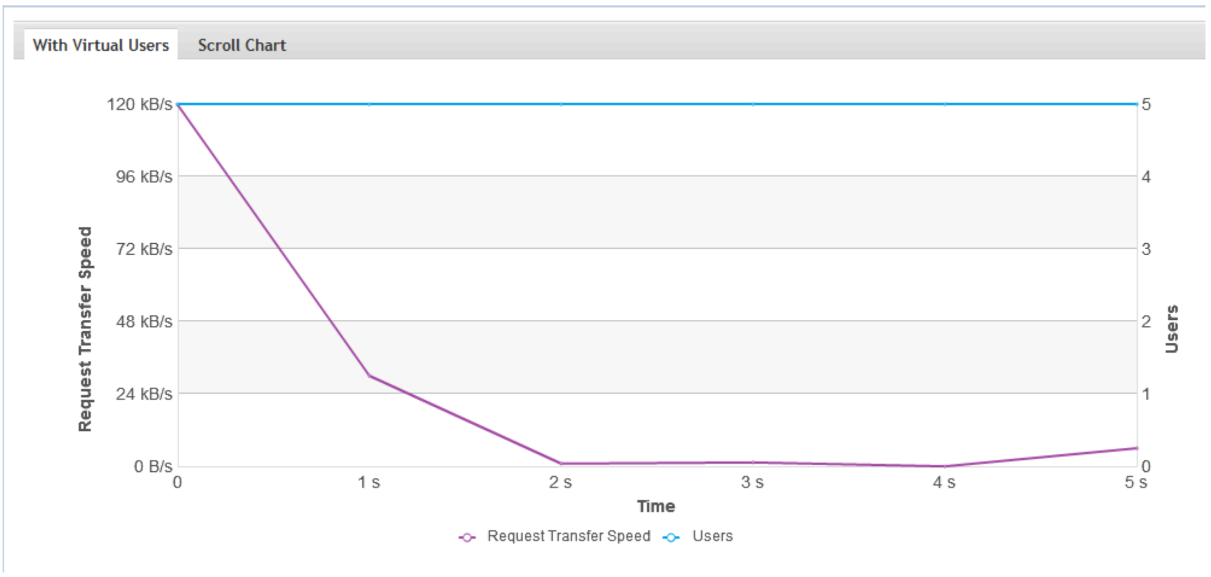
**Page Load Time**

| | |
|---|---|
| Average | 1.86 s |
| Maximum | 2.70 s |
| Minimum | 1.03 s |

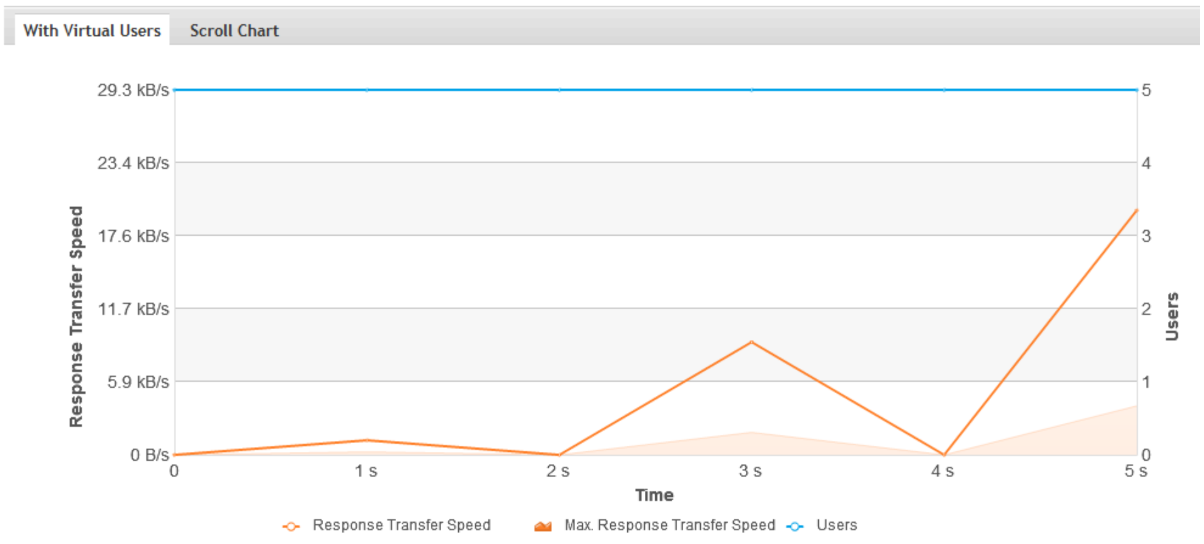
**Time to First Byte**

| | |
|---|---|
| Average | 417 ms |
| Maximum | 763 ms |
| Minimum | 93 ms |

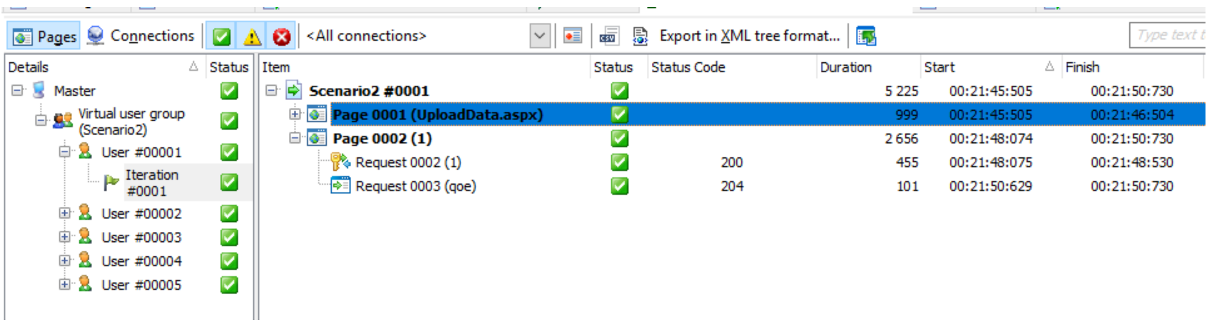
Scenario2 Test results report



Scenario 2 Request Transfer Speed



Scenario 2 Response Transfer Speed

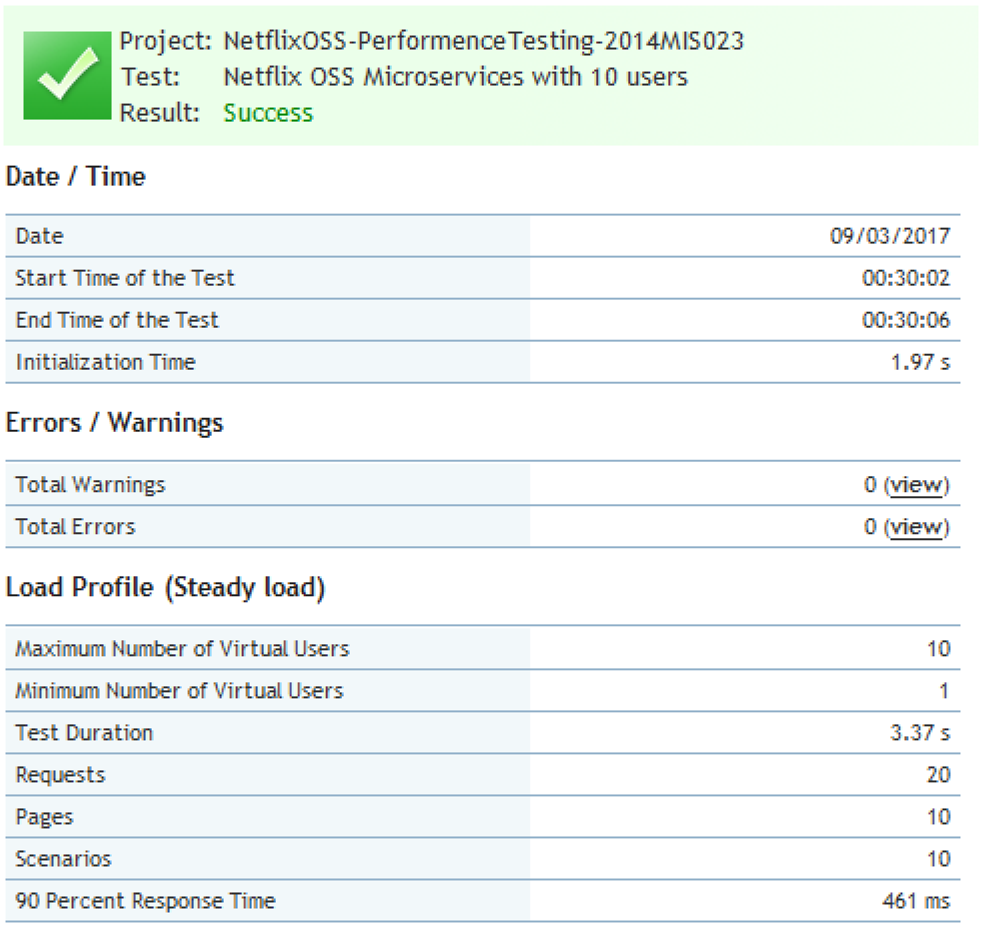


| Details | Status |
|---|---|
| Master | ✓ |
| Virtual user group (Scenario2) | ✓ |
| User #00001 | ✓ |
| Iteration #0001 | ✓ |
| User #00002 | ✓ |
| User #00003 | ✓ |
| User #00004 | ✓ |
| User #00005 | ✓ |

| Item | Status | Status Code | Duration | Start | Finish |
|---|---|---|---|---|---|
| **Scenario2 #0001** | ✓ | | 5 225 | 00:21:45:505 | 00:21:50:730 |
| **Page 0001 (UploadData.aspx)** | ✓ | | 999 | 00:21:45:505 | 00:21:46:504 |
| **Page 0002 (1)** | ✓ | | 2 656 | 00:21:48:074 | 00:21:50:730 |
| Request 0002 (1) | ✓ | 200 | 455 | 00:21:48:075 | 00:21:48:530 |
| Request 0003 (qoe) | ✓ | 204 | 101 | 00:21:50:629 | 00:21:50:730 |

Senario 2 Multiple user test

## Appendix B3



| | |
|---|---|
| Project: | NetflixOSS-PerformenceTesting-2014MIS023 |
| Test: | Netflix OSS Microservices with 10 users |
| Result: | Success |

### Date / Time

| | |
|---|---|
| Date | 09/03/2017 |
| Start Time of the Test | 00:30:02 |
| End Time of the Test | 00:30:06 |
| Initialization Time | 1.97 s |

### Errors / Warnings

| | |
|---|---|
| Total Warnings | 0 (view) |
| Total Errors | 0 (view) |

### Load Profile (Steady load)

| | |
|---|---|
| Maximum Number of Virtual Users | 10 |
| Minimum Number of Virtual Users | 1 |
| Test Duration | 3.37 s |
| Requests | 20 |
| Pages | 10 |
| Scenarios | 10 |
| 90 Percent Response Time | 461 ms |

### Scenario Completion Time

| | |
|---|---|
| Average | 2.24 s |
| Maximum | 3.37 s |
| Minimum | 2.06 s |

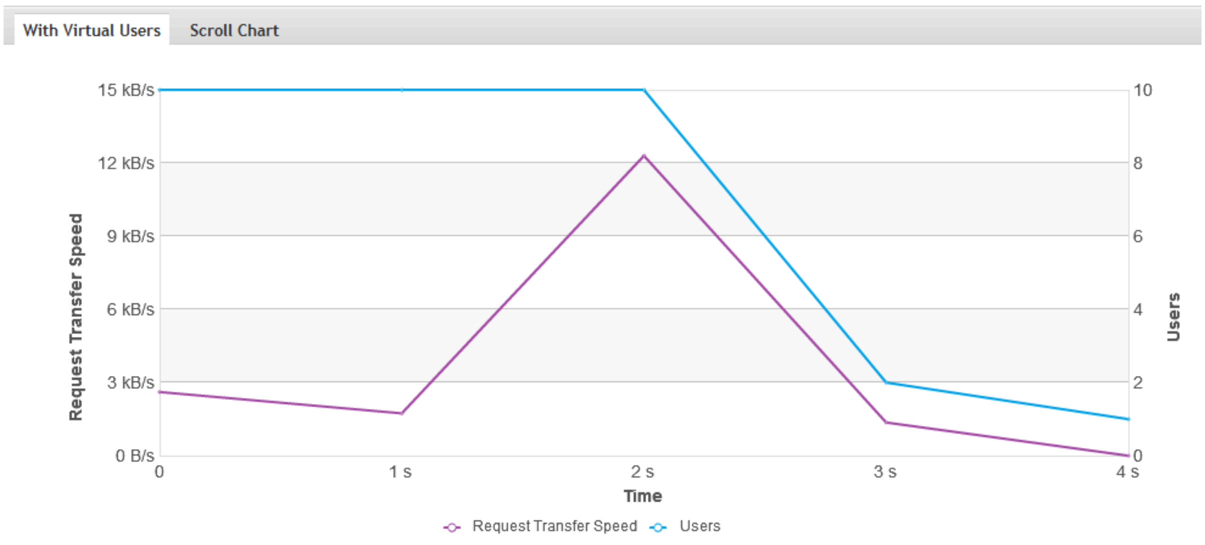
### Page Load Time

| | |
|---|---|
| Average | 2.24 s |
| Maximum | 3.37 s |
| Minimum | 2.06 s |

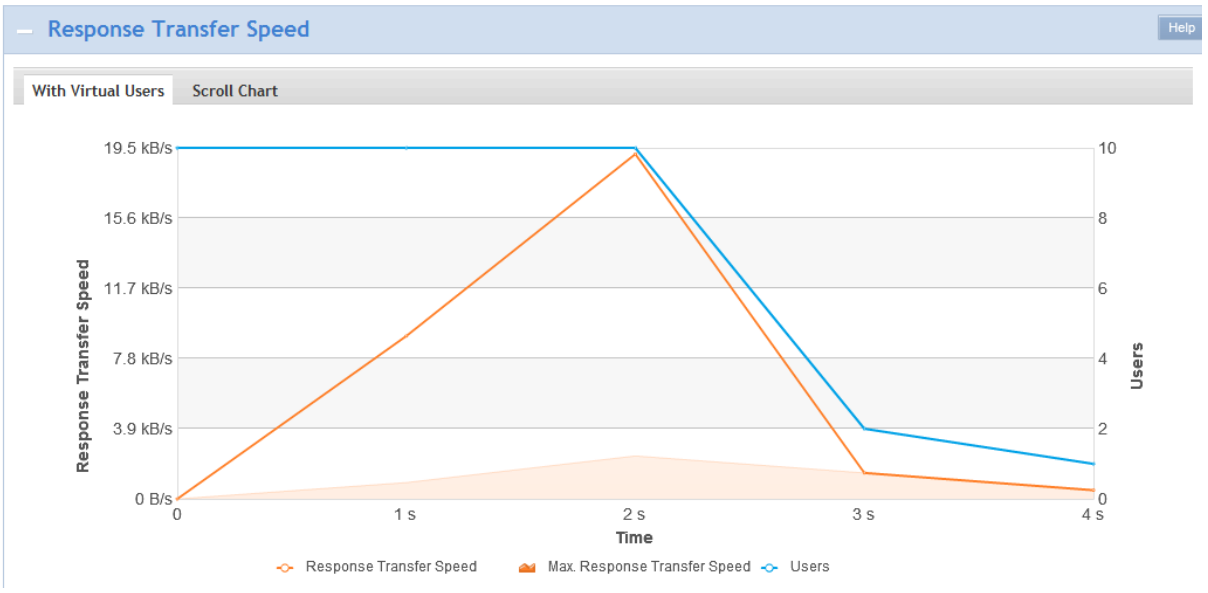
### Time to First Byte

| | |
|---|---|
| Average | 704 ms |
| Maximum | 1.60 s |
| Minimum | 289 ms |

Scenario3 Test results report

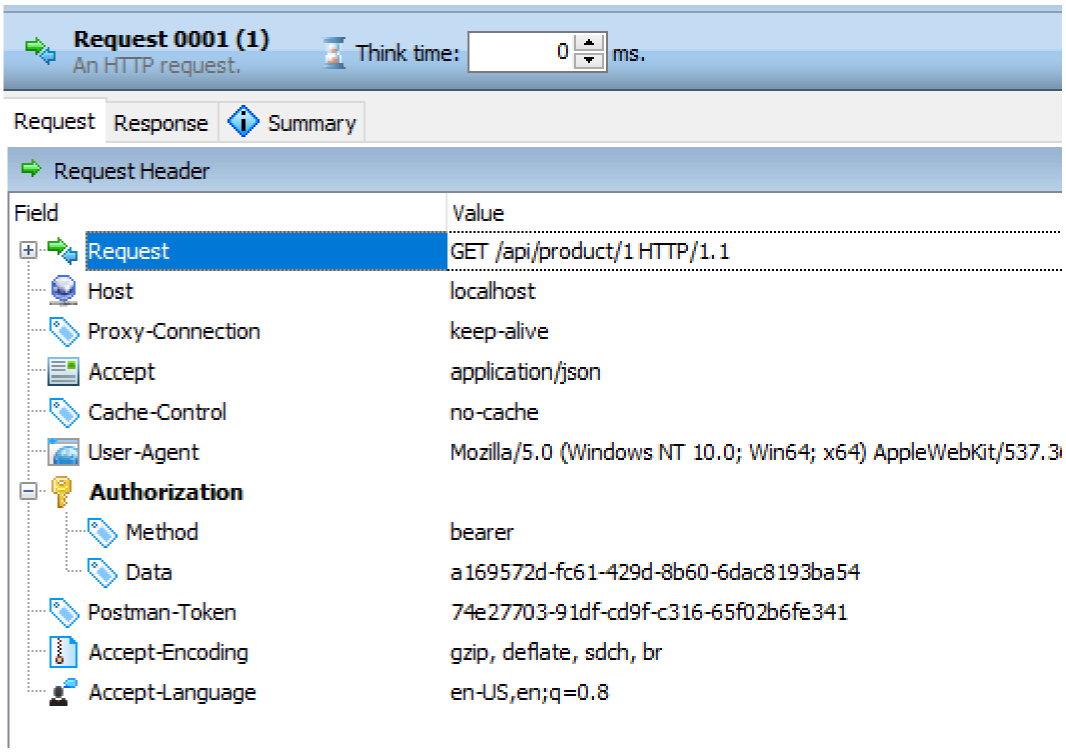


Scenario 3 Request Transfer Speed



Scenario 3 Response  Transfer Speed

## Appendix B4



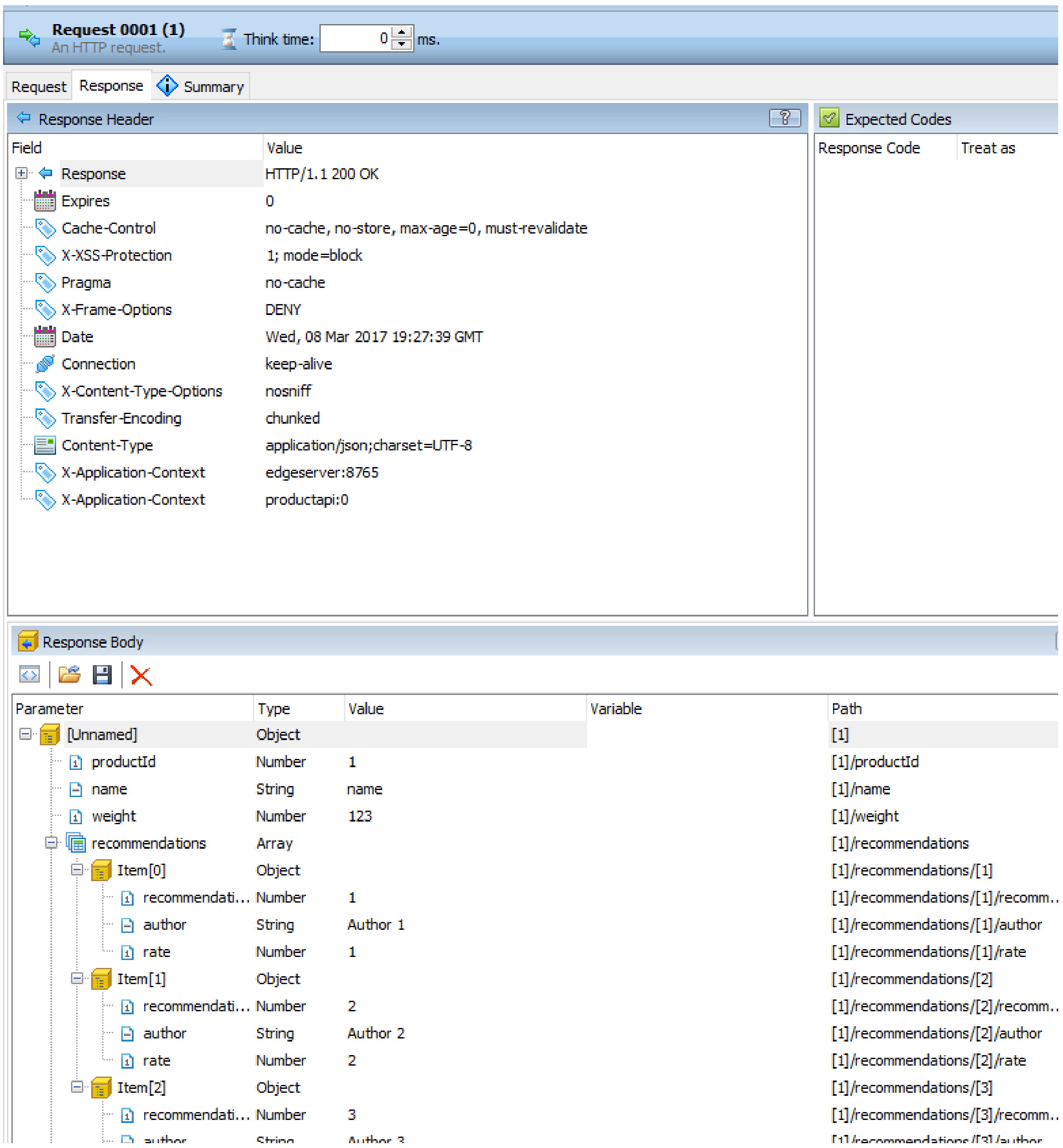
Request 0001 (1)
An HTTP request.
Think time: 0 ms.

Request | Response | Summary

Request Header

| Field | Value |
|---|---|
| Request | GET /api/product/1 HTTP/1.1 |
| Host | localhost |
| Proxy-Connection | keep-alive |
| Accept | application/json |
| Cache-Control | no-cache |
| User-Agent | Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.3 |
| **Authorization** | |
| Method | bearer |
| Data | a169572d-fc61-429d-8b60-6dac8193ba54 |
| Postman-Token | 74e27703-91df-cd9f-c316-65f02b6fe341 |
| Accept-Encoding | gzip, deflate, sdch, br |
| Accept-Language | en-US,en;q=0.8 |

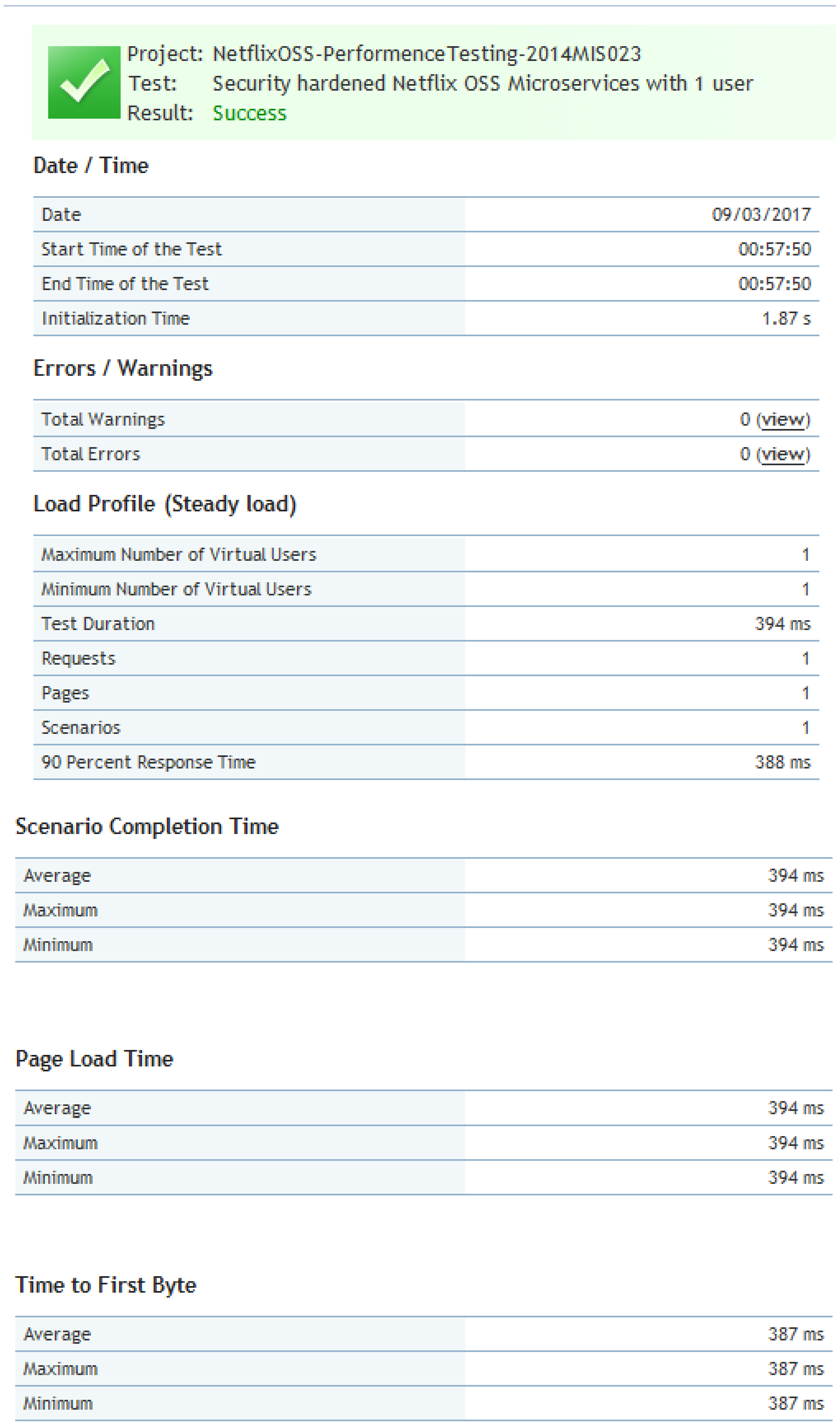
Scenario 4 Request



| Field | Value |
|---|---|
| Response | HTTP/1.1 200 OK |
| Expires | 0 |
| Cache-Control | no-cache, no-store, max-age=0, must-revalidate |
| X-XSS-Protection | 1; mode=block |
| Pragma | no-cache |
| X-Frame-Options | DENY |
| Date | Wed, 08 Mar 2017 19:27:39 GMT |
| Connection | keep-alive |
| X-Content-Type-Options | nosniff |
| Transfer-Encoding | chunked |
| Content-Type | application/json;charset=UTF-8 |
| X-Application-Context | edgeserver:8765 |
| X-Application-Context | productapi:0 |

| Parameter | Type | Value | Variable | Path |
|---|---|---|---|---|
| [Unnamed] | Object | | | [1] |
| productId | Number | 1 | | [1]/productId |
| name | String | name | | [1]/name |
| weight | Number | 123 | | [1]/weight |
| recommendations | Array | | | [1]/recommendations |
| Item[0] | Object | | | [1]/recommendations/[1] |
| recommendati... | Number | 1 | | [1]/recommendations/[1]/recomm.. |
| author | String | Author 1 | | [1]/recommendations/[1]/author |
| rate | Number | 1 | | [1]/recommendations/[1]/rate |
| Item[1] | Object | | | [1]/recommendations/[2] |
| recommendati... | Number | 2 | | [1]/recommendations/[2]/recomm.. |
| author | String | Author 2 | | [1]/recommendations/[2]/author |
| rate | Number | 2 | | [1]/recommendations/[2]/rate |
| Item[2] | Object | | | [1]/recommendations/[3] |
| recommendati... | Number | 3 | | [1]/recommendations/[3]/recomm.. |
| author | String | Author 3 | | [1]/recommendations/[3]/author |

Scenario 4 Response



Project: NetflixOSS-PerformenceTesting-2014MIS023
Test: Security hardened Netflix OSS Microservices with 1 user
Result: Success

### Date / Time

| | |
|---|---|
| Date | 09/03/2017 |
| Start Time of the Test | 00:57:50 |
| End Time of the Test | 00:57:50 |
| Initialization Time | 1.87 s |

### Errors / Warnings

| | |
|---|---|
| Total Warnings | 0 (view) |
| Total Errors | 0 (view) |

### Load Profile (Steady load)

| | |
|---|---|
| Maximum Number of Virtual Users | 1 |
| Minimum Number of Virtual Users | 1 |
| Test Duration | 394 ms |
| Requests | 1 |
| Pages | 1 |
| Scenarios | 1 |
| 90 Percent Response Time | 388 ms |

### Scenario Completion Time

| | |
|---|---|
| Average | 394 ms |
| Maximum | 394 ms |
| Minimum | 394 ms |

### Page Load Time

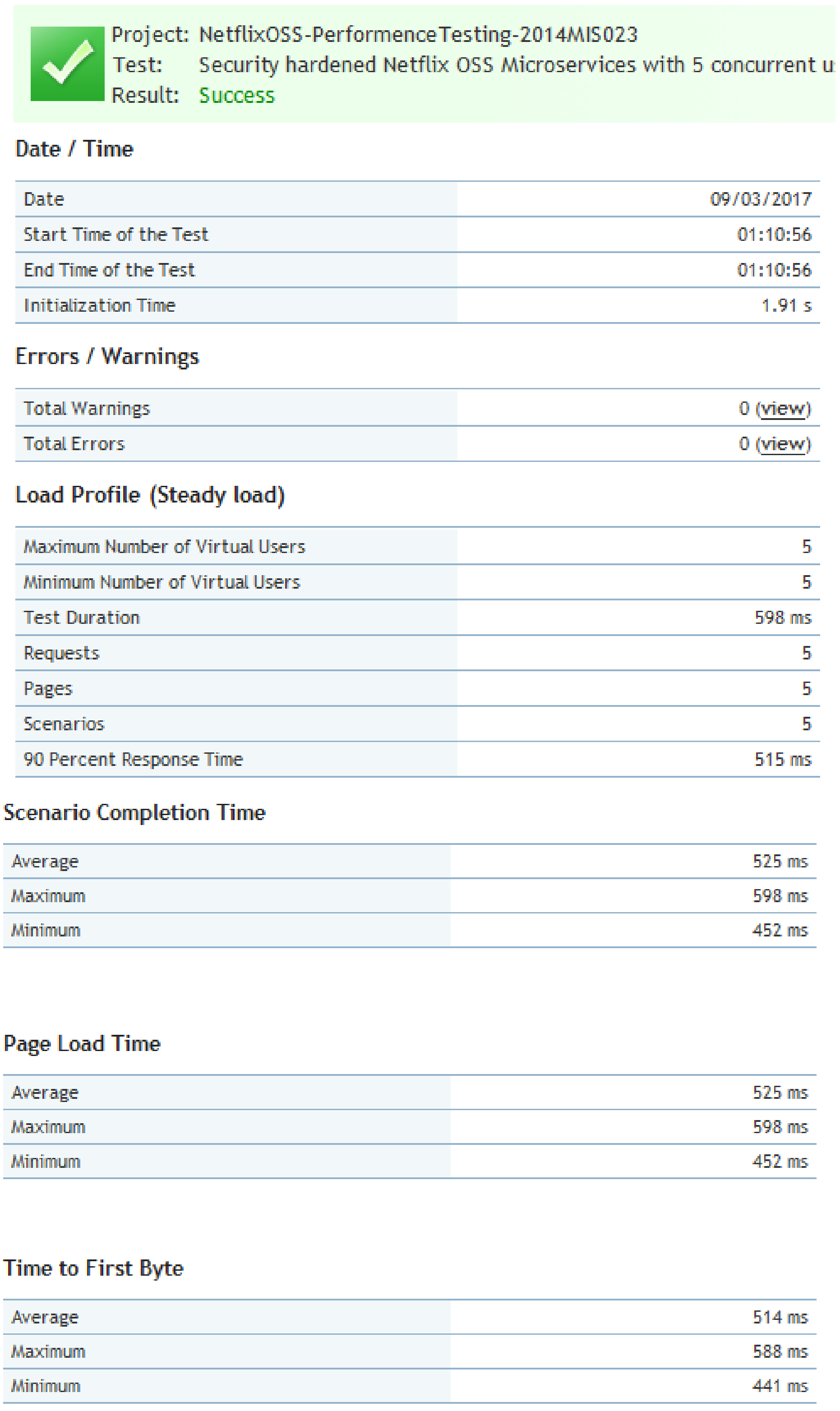
| | |
|---|---|
| Average | 394 ms |
| Maximum | 394 ms |
| Minimum | 394 ms |

### Time to First Byte

| | |
|---|---|
| Average | 387 ms |
| Maximum | 387 ms |
| Minimum | 387 ms |

Scenario4 Test results report

## Appendix B5



Project: NetflixOSS-PerformenceTesting-2014MIS023
Test: Security hardened Netflix OSS Microservices with 5 concurrent u
Result: Success

### Date / Time

| | |
|---|---|
| Date | 09/03/2017 |
| Start Time of the Test | 01:10:56 |
| End Time of the Test | 01:10:56 |
| Initialization Time | 1.91 s |

### Errors / Warnings

| | |
|---|---|
| Total Warnings | 0 (view) |
| Total Errors | 0 (view) |

### Load Profile (Steady load)

| | |
|---|---|
| Maximum Number of Virtual Users | 5 |
| Minimum Number of Virtual Users | 5 |
| Test Duration | 598 ms |
| Requests | 5 |
| Pages | 5 |
| Scenarios | 5 |
| 90 Percent Response Time | 515 ms |

### Scenario Completion Time

| | |
|---|---|
| Average | 525 ms |
| Maximum | 598 ms |
| Minimum | 452 ms |

### Page Load Time

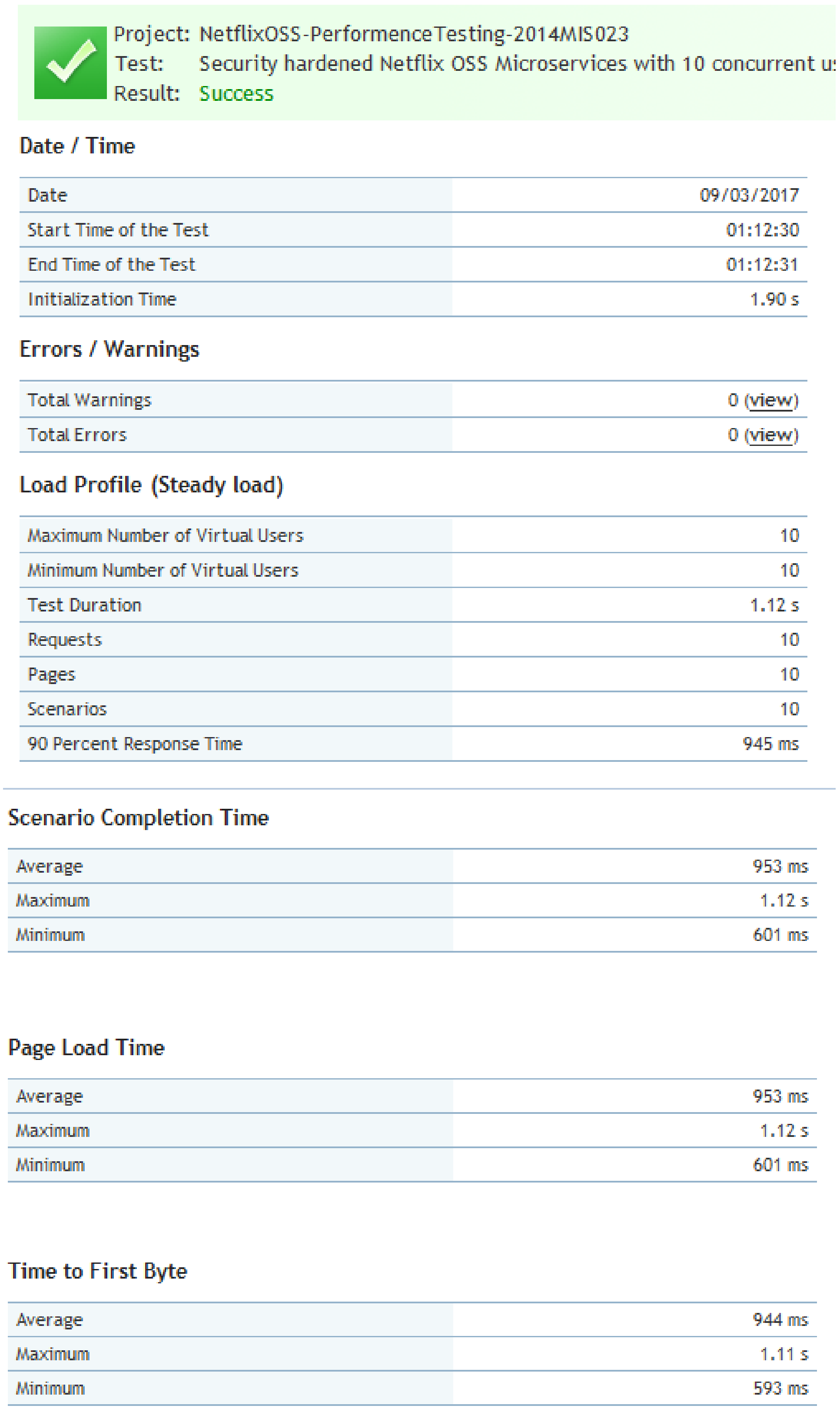
| | |
|---|---|
| Average | 525 ms |
| Maximum | 598 ms |
| Minimum | 452 ms |

### Time to First Byte

| | |
|---|---|
| Average | 514 ms |
| Maximum | 588 ms |
| Minimum | 441 ms |

Scenario 5 Test results report

## Appendix B6



Project: NetflixOSS-PerformenceTesting-2014MIS023
Test: Security hardened Netflix OSS Microservices with 10 concurrent u
Result: Success

### Date / Time

| | |
|---|---|
| Date | 09/03/2017 |
| Start Time of the Test | 01:12:30 |
| End Time of the Test | 01:12:31 |
| Initialization Time | 1.90 s |

### Errors / Warnings

| | |
|---|---|
| Total Warnings | 0 (view) |
| Total Errors | 0 (view) |

### Load Profile (Steady load)

| | |
|---|---|
| Maximum Number of Virtual Users | 10 |
| Minimum Number of Virtual Users | 10 |
| Test Duration | 1.12 s |
| Requests | 10 |
| Pages | 10 |
| Scenarios | 10 |
| 90 Percent Response Time | 945 ms |

### Scenario Completion Time

| | |
|---|---|
| Average | 953 ms |
| Maximum | 1.12 s |
| Minimum | 601 ms |

### Page Load Time

| | |
|---|---|
| Average | 953 ms |
| Maximum | 1.12 s |
| Minimum | 601 ms |

### Time to First Byte

| | |
|---|---|
| Average | 944 ms |
| Maximum | 1.11 s |
| Minimum | 593 ms |

Scenario 6 Test results report