Framework de Autenticação para FaaS em Nuvem Pública

Matheus de Sousa Lemos Fernandes[1], **Leonardo Rebouças de Carvalho**[2], and **Aleteia Patricia Favacho de Araujo**[3]

[1,2,3]Departamento de Ciência da Computação - Universidade de Brasília (UnB) - Brasília - DF - Brazil

[1]

Abstract. *Access control is an important functionality on internet applications. On managed cloud providers, however, access control security is often simplified, and user access is often done with simple usernames and passwords, which does not provide enough security for cloud based applications. This paper presents a framework developed to authenticate users on cloud providers, in order to access FaaS services. To ensure user authenticity, this framework uses OpenID Connect, a modern authentication technology.*

Resumo. *Controle de acesso é uma importante funcionalidade em aplicações na internet. Em plataformas de nuvem, entretanto, a segurança no controle de acesso acaba sendo simplificada, e o acesso é frequentemente realizado utilizando os comuns usuário e senha, o que muitas vezes não oferece a segurança necessária em um ambiente de nuvem pública. Este trabalho apresenta um* framework *desenvolvido para autenticar usuários junto a um provedor de nuvem, de modo a utilizar serviços de* FaaS. *Para garantir a autenticidade do usuário, utilizou-se o OpenID Connect, uma moderna tecnologia de autenticação.*

1. Introdução

Função como um Serviço (*Function as a Service*) é um paradigma de computação que envolve a definição de unidades atômicas de código, chamadas funções, na qual cada função deve realizar apenas uma tarefa. Essas funções podem ser chamadas por diversos meios (chamadas de *API*, gatilhos, *logs*), e seus resultados podem ser retornados ao usuário ou conectadas a outros serviços de nuvem [AWS 2021].

Ambientes de *FaaS*, entretanto, não possuem mecanismos de autenticação próprios, e dependem dos mecanismos de acesso disponíveis na própria plataforma. O uso de um serviço dentro do próprio provedor pode muitas vezes não ser estratégico para o usuário do serviço. O usuário muitas vezes precisa manter os dados de seus usuários em seus próprios ambientes privados.

Nesse caso, é vantajoso utilizar uma ferramenta de autenticação própria, utilizando padrões abertos e previamente auditados.

Assim sendo, esse *framework* apresenta um mecanismo de autenticação utilizando a técnica de *Single-Sign-On* (*SSO*) [Indu et al. 2018], delegando a um provedor de identidade especializado a tarefa de confirmar a identidade dos usuários da plataforma.

2. Framework de autenticação

Para controle de acesso em funções como serviço, o *framework* desenvolvido neste projeto realiza autenticação e autorização do usuário. Para isso, é utilizado o Keycloak [Keycloak 2021], uma ferramenta de controle de acesso, para criar usuários, autenticá-los por meio de um navegador *web*, e verificar que os mesmos possuem autorização para executar funções em um provedor previamente configurado. Tais permissões são definidas pelo administrador do servidor onde são executados tanto o Keycloak quanto a *API* que recebe as requisições de usuário.

Uma vez que o usuário é autenticado, o *token* gerado é enviado ao provedor de *FaaS*, junto com os parâmetros de execução da função, que verifica se as permissões recebidas são compatíveis com as permissões definidas na função do provedor de *FaaS*.

O *framework* é composto de uma *API RESTful*, acessível por meio de qualquer aplicação cliente que realize chamadas *HTTP*, a qual permite ao usuário se cadastrar e chamar funções em um provedor de *FaaS*. A aplicação se comunica com o Keycloak para verificar um usuário, e caso o usuário tenha acesso ao Keycloak, a ferramenta gera um *token*, que contém as informações necessárias para o acesso devido ao provedor de nuvem. A Figura 1 apresenta a visão geral do framework proposto.

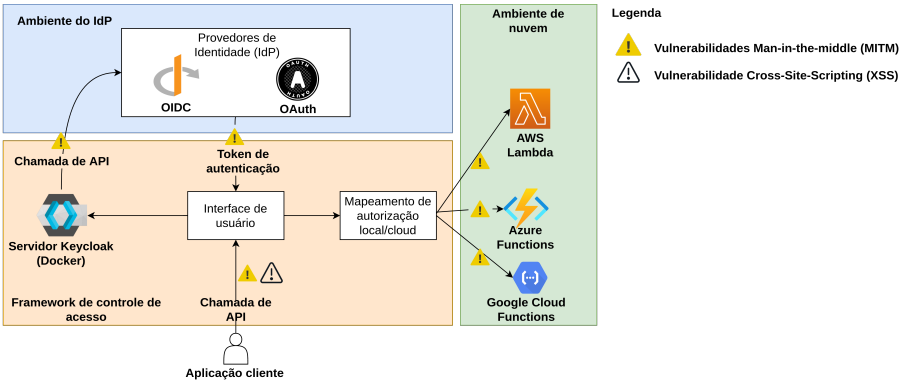

Figure 1. Diagrama de funcionamento da aplicação.

Na Figura 1, os pontos marcados por exclamações elucidam as vulnerabilidades mais comuns em mecanismos de autenticação: interceptação de mensagens (ataques *man-in-the-middle*, ou *MITM*) e *Cross Site Scripting* (*XSS*) [Navas and Beltrán 2018].

Para executar a função no ambiente de nuvem remoto, o usuário deve se autenticar junto à aplicação. Para isso uma aplicação cliente deve executar os seguintes passos:

1. Criação de usuário (caso não exista);
2. Geração de URL de autenticação;
3. *Login* de usuário no Keycloak;

Em seguida, o usuário receberá um *token* no formato *JSON Web Token* (*JWT*), que deve ser usado para autenticar o usuário no ambiente de nuvem. Isso é feito passando-se o *token* no campo de autorização de uma requisição *HTTP*. No *token* recebido são retornadas também as permissões de usuário. As permissões são definidas utilizando Role Based Access Control (RBAC), de forma a restringir as permissões do usuário às permissões definidas pelo "papel" (*role*) associado [Indu et al. 2018].

3. Experimento

O aspecto mais relevante a ser avaliado é a segurança dos dados em trânsito entre as partes. Para que os dados não sejam capturados por um agente malicioso (atacante), é necessário o uso de criptografia de chave pública entre o usuário e a aplicação. Para isso, foram gerados certificados *TLS*, que são utilizados para criptografar a comunicação entre as partes. Os certificados e chaves foram gerados utilizando a ferramenta OpenSSL [OpenSSL 2021].

Para verificar que os dados de usuário foram corretamente criptografados e que não poderiam ser interpretados por um atacante, foi utilizada a ferramenta Wireshark [Wireshark 2021]. O Wireshark é um *packet sniffer*, e foi utilizado para verificar que os dados entre as partes não foram capturados em texto plano ao trafegar pela rede.

As táticas de mitigação descritas são voltadas para ataques de interceptação de fluxo (*flow interception*), que incluem ataques *MITM*, e ataques de manipulação da *URL* de redirecionamento [Navas and Beltrán 2018]. Para fins de teste, a aplicação foi configurada de três formas:

1. Execução sem o uso de certificados;
2. Execução com certificado para *API*;
3. Execução com certificado para *API* e Keycloak.

Em cada um dos experimentos foram executados os passos para execução do serviço de do tipo *FaaS*. Para cada experimento foi gerado um arquivo *pcapng* resultante do Wireshark, com os pacotes capturados na interface de rede local (*loopback*). Dessa forma, foi possível verificar de forma objetiva que a solução proposta protege dados sensíveis do usuário em todos os passos da comunicação entre os componentes da aplicação.

4. Conclusão

Aplicações em um ambiente de função como serviço muitas vezes podem ter sua segurança negligenciada, seja por excesso de confiança nos mecanismos do provedor ou por falta de conhecimento dos desenvolvedores. A ferramenta apresentada neste artigo oferece um mecanismo para proteger dados sensíveis durante todo o processo de chamada de funções em um ambiente de nuvem. Assim evita-se que informações privadas dos usuários sejam capturadas por agentes maliciosos e garantir o acesso seguro ao ambiente de *FaaS*.

References

AWS (2021). Aws lambda - product features. https://aws.amazon.com/lambda/features/. Acessado: Outubro, 2021.

Indu, I., Anand, P., and Bhaskar, V. (2018). Identity and access management in cloud environment: Mechanisms and challenges.

Keycloak (2021). Keycloak. https://www.keycloak.org/. Acessado: Outubro,2021.

Navas, J. and Beltrán, M. (2018). Understanding and mitigating openid connect threats.

OpenSSL (2021). Openssl - cryptography and ssl/tls toolkit. https://www.openssl.org/. Acessado: Outubro, 2021.

Wireshark (2021). Wireshark. https://www.wireshark.org/. Acessado: Outubro, 2021.