Estudo de como incorporar controle de acesso baseado em atributos em um *framework* de segurança para microsserviços

Rafael Tenfen
Programa de Pós-Graduação em Computação Aplicada (PPGCA)
Universidade do Estado de Santa Catarina (UDESC)
Joinville, Brasil
rafaeltenfen.rt@gmail.com

***Abstract*—A arquitetura de microsserviços é uma tendência em engenharia de software para construir sistemas escaláveis e flexíveis. Essa, é uma proposta de estudo para incorporar *Attribute Based Access Control* (ABAC), buscando autorização de serviços a um *framework* de segurança de microsserviços de código aberto chamado *MicroService Security Framework* (MiSSFire).**

***Index Terms*—microsserviços, segurança, ABAC, MiSSFire, autorização**

I. Introdução

Microsserviços são uma arquitetura inspirada por *Service Oriented Architecture* combinado com o velho princípio da Unix "*do one thing and do it well*" [1]. Existem benefícios de utilizar a arquitetura de microsserviços, como: organização, lançamentos de versões de softwares mais frequentes, escalonamento independente de componentes e uma rápida adoção à novas tecnologias [2].

Os microsserviços, em seu estágio inicial de desenvolvimento, costumam ser projetados para confiar completamente uns nos outros [3], pois é complexo em termos de desenvolvimento e arquitetura, um sistema distribuído com um grande número de nós definir estabelecimento de confiança entre serviços, assim como a autenticação e autorização para usuários dos serviços.

Considerar um microsserviço confiável, representa uma premissa forte na era da conectividade, onde microsserviços podem interagir entre si de forma heterogênea e aberta. Pois, um microsserviço pode ser atacado e controlado por um adversário malicioso, comprometendo não apenas o microsserviço único, mas em um caso mais drástico, paralisando toda a aplicação.

A Netflix, uma aplicação com um grande números de nós já teve um de seus subdomínios comprometidos, e a partir desse domínio, o atacante pode servir qualquer conteúdo no domínio netflix.com, além de poder acessar os dados de seus assinantes autenticados, uma vez que a Netflix permitiu que todos os *cookies* dos usuários, fossem acessados de qualquer subdomínio [4].

Portanto, microsserviços introduzem complexidade de coordenação de comunicação entre os módulos, comparados a arquitetura monolítica, que por sua vez criam novos riscos de segurança. Além disso, traz desafios para definir autorizações, pois, cada microsserviço é uma parte independente que, em casos extremos, não podem ser confiáveis. Em particular, arquiteturas distribuídas criam problemas de controle de acesso, como o ataque *confused deputy* e a utilização de *powerful tokens* [5].

O ataque de *confused deputy* referido como '*vulnerability du jour*', é um ataque de escalonamento de privilégios, em que um microsserviço quer é confiável por outros microsserviços é comprometido, isso faz com que os demais microsserviços respondam às solicitações do microsserviço comprometidos, sem saber que o mesmo está comprometido [6]. Já o *powerful tokens*, é resultante da implementação da lógica de controle de acesso em que não é realizada a verificação necessária para o acesso em cada serviço, mas sim apenas através de um *gateway* que pode acessar todos os serviços com o mesmo *token*.

Para parcialmente atender os problemas relacionados a segurança em microsserviços, o *framework* MiSSFire tem o objetivo prover uma maneira padrão de estabelecer segurança de microsserviços [1]. Os mecanismos já implementados são *Mutual Transport Layer Security* (MTLS) para autenticação mútua de serviços e criptografia do tráfico de dados, e *JSON Web Token* (JWT) para a sua propagação. Contudo, o framework não possui mecanismo de autorização para controle de acesso como *Role Based Access Control* (RBAC) ou ABAC.

Tradicionalmente, as soluções de controle de acesso lógico têm sido baseadas principalmente na identidade da requisição de uma operação (por exemplo, leitura) sobre um recurso de banco de dados ou um objeto [7]. Uma das soluções para o controle de acesso lógico é determinar permissões para cada usuário nas rotinas e ações do sistemas, porém caso o usuário necessite de mais permissões, é necessário realizar novas inserções ou alterações nas permissões já atribuídas ao usuário e ao criar novos usuários, além do objeto usuário, é necessário atribuir também suas permissões de modo a satisfazer as necessidades de permissões de cada usuário.

Dessa forma, para um sistema em que possui 4 módulos: contabilidade, fiscal, tributário e recursos humanos, cada módulo com 10 rotinas, um usuário fiscal deve ter permissões

para realizar as ações de alterar, excluir, ler e inserir objetos em todas as rotinas do módulo fiscal, porém ao ser promovido e alterar a sua função para fiscal tributário contábil, é necessário adicionar permissões para todas as ações das 10 rotinas do módulo tributário e também as ações das rotinas do módulo da contabilidade. Portanto, para um sistema com muitos módulos, utilizar essa definição de controle de acesso é inviável devido à elevada taxa de manutenção.

Para resolver o problema da elevada taxa de manutenção na troca de funções foi desenvolvido o controle de acesso baseado em funções (RBAC). A ideia de RBAC é que os usuários não tem acesso discricionário aos objetos do sistema, em vez disso, as permissões de acesso do usuário são associadas as suas funções dentro da organização [8], conforme apresentado na Figura 1. Assim, as regras e mapeamento de controle de acesso do sistema estão atreladas a funções e não mais aos usuários, rotinas e ações. Desse modo, conforme o exemplo anterior, o usuário fiscal ao ser promovido a outra função, é necessário apenas alterar a função do usuário, sem a necessidade de modificar as regras de permissões definidas por funções.

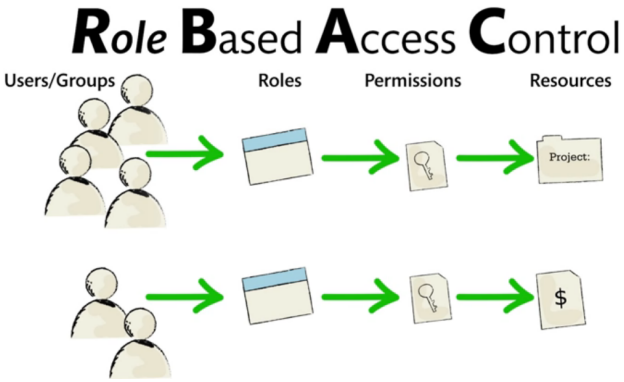

Fig. 1. Fluxo RBAC. Adaptado de [9].

Contudo, em grandes organizações devido a grande quantidade de regras, funções e projetos, as definições de controle de acesso baseado em funções se torna caótico para realizar sua manutenção, conforme Apresentado na Figura 2. Além disso, RBAC apresenta dificuldade em realizar definições de acesso em que é necessário maior especificidade do que apenas a função do usuário, como por exemplo permitir o usuário acesso ao módulo de finanças somente após o fim trimestre.

Dessa forma, devido a falta de granularidade de especificações do RBAC, uma solução mais escalável e flexível para controle de acesso é apresentada como ABAC. ABAC utiliza atributos do usuário, contexto e ambiente para especificar regras usando atributos para realizar a determinação de controle de acesso [12].

O objetivo principal de ABAC como um modelo de controle de acesso é atender aos requisitos de ambientes altamente heterogêneos [10], como os ambientes de microsserviço. Pois, com a sua utilização é possível definir qualquer atributo para especificar um acesso, como localização, hora do dia, departamento, função, qualificação e até mesmo frequência podem ser levados em consideração para realizar a especificação [11].

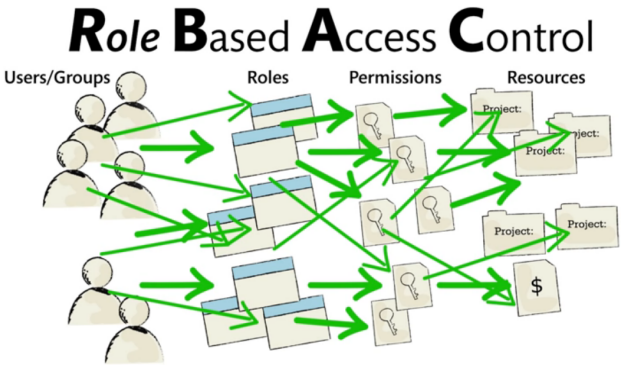

Fig. 2. Fluxo RBAC de grandes organizações. Adaptado de [9].

Assim, utilizar ABAC no ecossistema de microsserviços, permite definir uma especificação com contexto para cada objeto ou conjunto de objetos de modo que facilite a manutenibilidade do controle de acesso para projetos ou serviços que já existem, como também para os que estão por vir, conforme ilustrado na Figura 3.

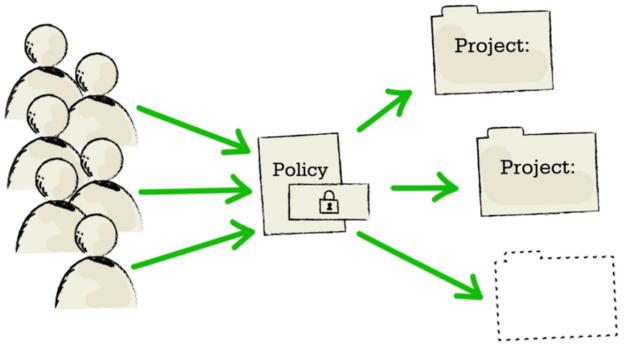

Fig. 3. Fluxo ABAC. Adaptado de [9].

II. Objetivo

Estudar a possibilidade de incorporar *Attribute Based Access Control* em um *framework* de segurança para microsserviços de código aberto.

III. Justificativa

Com a crescente tendência da arquitetura de microsserviços pela indústria de software, surge também a urgência de acompanhar a segurança dos serviços prestados, pois já dizia William H. Webster ”security is always seen as too much until the day it’s not enough”.

A autenticação dos serviços já é oferecida pelo *framework* MiSSFire, através da autenticação baseada em *tokens* utilizando JWT que é uma forma muito utilizada por várias redes de microsserviços na indústria de software, algum exemplos podem ser encontrados em [13]. A autenticação usando JWT, permite que o usuário se autentique apenas uma vez, obtendo um *token* JWT como prova de sua autenticação bem sucedida. Esse *token* é incluído em cada requisição, e cada microsserviço pode validar o *token* e extrair a identidade do usuário.

Contudo, a autorização dos serviços não é abordada por MiSSFire, e a sua aplicação pode trazer benefícios ao projeto em relação ao mecanismo de controle de acesso entre os serviços, tais como tomar decisões de controle de acesso sem conhecimento prévio do usuário que realizou a requisição e evita a necessidade de autorizações explícitas a serem atribuídas diretamente a usuários individuais, além de permitir especificar com granularidade as regras de controle de acesso devido a possibilidade de levar em consideração quaisquer atributos relacionados ao usuário.

IV. MiSSFire

MiSSFire é um *framework* implementado para a segurança de microsserviços com foco estabelecer comunicação segura entre serviço. A ideia principal era proporcionar segurança, escalabilidade e automação. Os mecanismos de segurança implementados atualmente são MTLS e JWT [1].

O framework teve o seu desenvolvimento focado na disponibilização de dois serviços expostos através de *Representational State Transfer* (REST) *Application Programming Interface* (API) para realizar a comunicação entre os serviços de acordo com o protocolo MTLS e padrão JWT.

A. Autenticação MTLS

Para a possibilitar a comunicação entre os microsserviços utilizando o protocolo MTLS, um serviço de *Certificate Authority* (CA) é definido como parte principal da auto-hospedada *Public Key Infrastructure* (PKI). Assim, é gerado um certificado raiz autoassinado para assinar o *Certificate Signing Request* (CSR) de outros serviços.

A solução apresentada para a confiança de comunicação entre os microsserviços baseada em MTLS constitui em uma ordem cronológica de fatos: o serviço CA gera um certificado raiz autoassinado e uma chave compartilhada, então um *hash* criptografado do certificado e a chave compartilhada são extraídos do serviço CA e fornecidos no novo serviço de modo automático se previamente configurado ou manualmente. Então, o novo serviço estabelece a conexão *Transport Layer Security* (TLS) unilateral com o serviço CA, verifica a identidade do CA e, se for bem sucedido, envia um CSR e uma chave compartilhada, após a verificação bem sucedida do CSR recebido e da chave compartilhada, o serviço CA assina de volta o certificado recém emitido e os dois serviços começam a se comunicar por MTLS, conforme apresentado pela Figura 4.

B. Autenticação JWT

Para aplicar a autenticação através do JWT, é realizado através do *Security Token Service* (STS) reverso que é um componente do padrão *WS-Trust* [14] que estende do padrão *WS-Security* para emitir, renovar e validar *tokens* de segurança, o STS reverso é uma tendência na industria de software para propagação de JWT na rede de microsserviços. Dessa forma, a autenticação de usuário a serviço baseada em *tokens* permite o transporte da identidade do usuário e do estado da sessão do usuário como demais atributos relacionados ao usuário

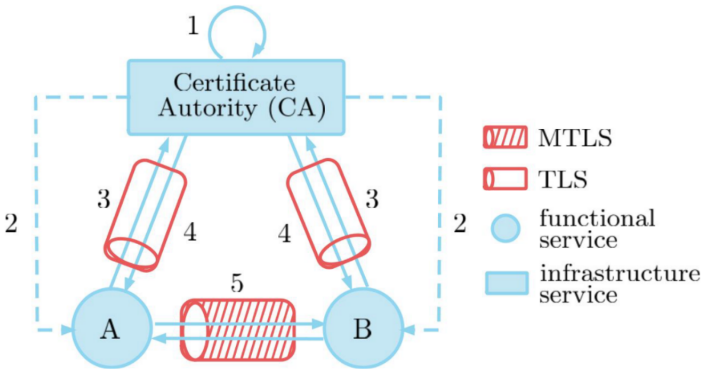

Fig. 4. Fluxo MTLS [1].

de maneira segura e descentralizada. Após o usuário ser autenticado, um *token* de segurança que representa o usuário é gerado para o uso interno na rede de microsserviços.

Assim, o processo de autenticação baseado em *token* genérico para microsserviços que permite a autenticação do usuário para a comunicação com o serviço e propagação de dados da sessão e atributos do usuário funciona de seguinte forma: A solicitação do usuário atinge a API do *Gateway*, então o *Gateway-API* solicita a autenticação do usuário redirecionado para o serviço de autenticação de usuário, solicitando um *token* de segurança, caso o usuário for autenticado com sucesso, o serviço de *token* de autenticação gera um *token* que representa o usuário fornecido dentro do sistema, assim retornando o *token* de segurança, o *token* de segurança é passado junto com a solicitação do usuário para os serviços de *downstream* conforme demonstrado na Figura 5. Desse modo, o *token* é designado apenas para o uso interno de comunicação e autenticação entre os serviços e em nenhum momento é fornecido ao usuário e a cada nova requisição do usuário esse processo acontece novamente para a criação de um novo *token*.

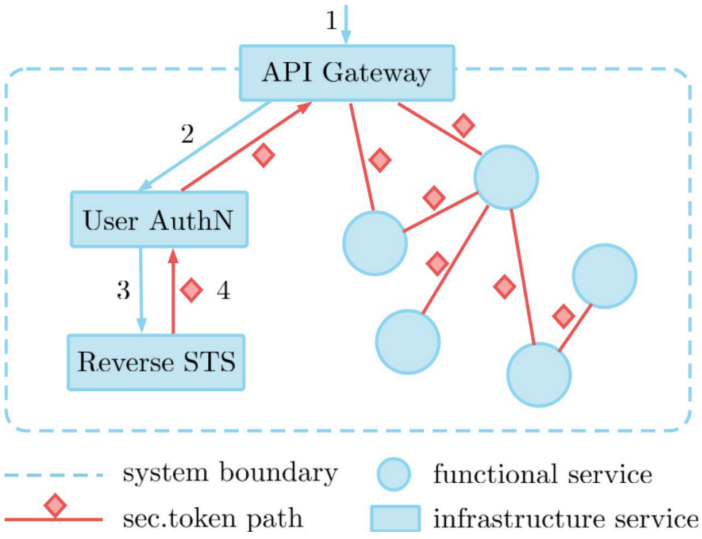

Fig. 5. Fluxo JWT [1].

C. Autorização

Atualmente o *framework* MiSSFire não possui implementação ou definição de mecanismos de controle de acesso, pois sua implementação inicial foi designada para definir um padrão de segurança com foco apenas em autenticação. Contudo, como o *framework* é de código aberto, licenciado por *GNU Public License* no repositório do GitHub (ttps://github.com/yarygina/MiSSFire) é possível estender a sua implementação e ampliar suas funcionalidades com foco na autorização de API's *endpoints*, de modo que permita a definição com especificidades de atributos do usuário para cada uma das rotas disponíveis em cada serviço na rede de microsserviços.

Assim, para realizar o mecanismo de controle de acesso de modo granular nos serviços da rede o modelo ABAC é o mais apropriado, permitindo especificar os acessos do usuário para cada rota de cada um dos serviços na rede.

V. ABAC

O modelo de controle de acesso baseado em atributos é flexível pois não precisa definir uma relação prévia entre o usuário e o recurso em que está sendo requisitado. Caso as regras de acesso de um recurso mude, é necessário apenas alterar as políticas de acesso ou os valores dos atributos necessários [15].

Os atributos no modelo ABAC podem ser considerados como características de qualquer especifidade que pode ser definido e ao qual um valor pode ser atribuído. O ABAC depende da avaliação de atributos do usuário, atributos do recurso a ser acessado, políticas predefinidas, condições do ambiente e estado atual para permitir ações do usuário que realiza a requisição.

Todos os sistemas em que incluem o modelo de controle de acesso ABAC possuem a capacidade de avaliar atributos, ambiente e estado. Além disso, o usuário não deve ter a liberdade de alterar os valores de seus próprios atributos de autorização, porque apenas eventos ou autoridades de segurança do sistema devem ser capazes de atualizar ou criar novos atributos de autorização e assim atribuir novos valores com bases nas permissões autoritativas do recurso [16].

Assim, quando uma requisição é realizada pelo usuário, os atributos e regras de controle de acesso são avaliados pelo mecanismo de controle de acesso baseado em atributos para forneces uma decisão de controle de acesso. Na forma básica do ABAC, o mecanismo de controle de acesso contém um ponto de decisão das regras a serem respeitadas e um ponto de aplicação das regras conforme ilustrado pela Figura 6.

VI. Incorporar ABAC ao MiSSFire

Para adicionar o mecanismo de controle de acesso baseado em atributos, é necessário modificar funcionalidades já existentes como o serviço de autenticação JWT e API *Gateway* e adicionar novas funcionalidades como disponibilizar um serviço para realizar a aplicação das regras de controle de acesso com base nos atributos do usuário, atributos do recurso a ser acessado, políticas predefinidas, condições do ambiente

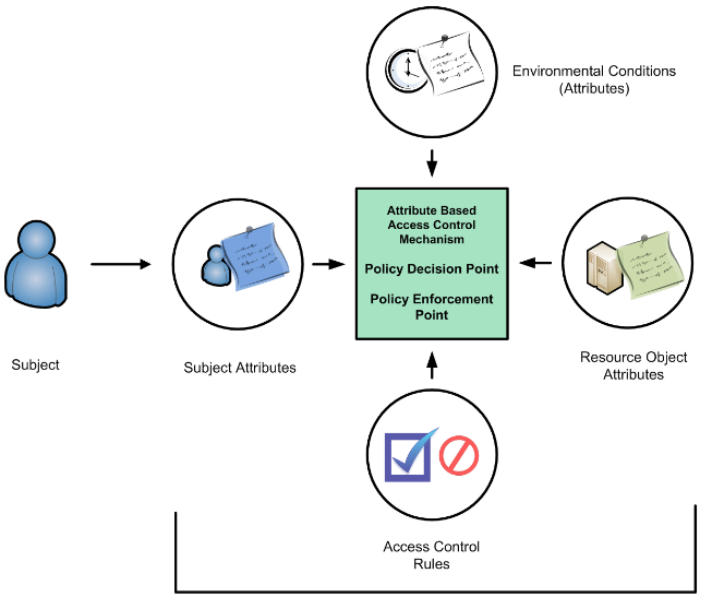

Fig. 6. Modelo básico ABAC. Adaptado de [16].

e estado atual. Além disso, como a comunicação entre os serviços da rede de microsserviços é realizado através da rede e em sua maioria com requisições REST com métodos *Hypertext Transfer Protocol* (HTTP) (GET, POST, PUT, DELETE) é necessário definir um padrão de especificação de políticas com as regras de acesso para cada rota e método de cada serviço da rede de microsserviços.

A. Padrão de Especificação

O padrão de especificação com as regras de acesso é definido como um arquivo de configuração é no formato *JavaScript Object Notation* (JSON), pois nele é possível especificar o nome do serviço, as rotas de cada serviço como *strings*, além da possibilidade de especificação de rotas com *wildcards* para agrupamento de regras, definir os atributos a serem utilizados na validação como dados do usuário, especifidades do recurso a ser acessado, caracterização do ambiente e atributos do estado atual, tipos dos atributos (*string, int, float, timestamp*) para realizar o parse dos dados, formato dos atributos para realizar a formatação dos atributos de forma correta, como datas podem seguir o padrão 'DD/MM/AAAA' ou 'MM/DD/AAAA' e operadores para as comparações dos atributos como igual, diferente, maior, menor, contém e não contém, por fim o conforme a estrutura apresentada pela Figura 7. Assim, possibilitar a granularidade das especificações de controle de acesso.

Como exemplo de uso do padrão de especificação das regras, tem-se como exemplo um usuário realizando requisição para o módulo de finanças, com o método POST para a rota "../project/{projectID}/salary" com o intuito de adicionar um salário, entretanto conforme definido no arquivo de configuração apresentado na Figura 8, só sera permitido o usuário a realizar essa operação caso ele tenha o seu atributo "created_at" maior que a data "01/12/2020", conforme descrito

Fig. 7. Estrutura de especificações no formato JSON.

Fig. 8. Exemplo de especificações de regra.

no objeto *value* do único atributo especificado no *array* de *atributes*.

B. Modificação

Como modificação inicial, terá como prioridade atualizar a versão da linguagem de programação utilizada nas funcionalidades do MiSSFire de Python 2.7 para o Python 3.7, para seguir e utilizar a mesma linguagem no desenvolvimento do serviço de controle de acesso baseado em atributos. Após realizar a atualização da versão de linguagem de programação, será modificado o serviço de *reverse* STS de autenticação JWT para adicionar no *payload* do *token* os dados de atributos de autorização do usuário.

Além de ter a necessidade de modificar o serviço de *gateway* para que passe pelo novo serviço de validação de especificação das regras definidas no JSON. Por fim, terá a necessidade de adicionar o novo serviço à configuração do serviço de MTLS e assinar o novo CSR, para que seja possível o a comunicação de modo mais seguro entre o serviço de validação das regras e os demais serviços da rede de microsserviços.

C. Fluxo de execução

O fluxo de execução de uma requisição então deve seguir inicialmente conforme apresentado pela Figura 5 até o momento de conectar com os demais serviços da rede, antes de executar esse passo, deverá então passar a requisição pelo novo serviço de validação das regras.

O novo serviço nesse momento deve ler as regras no arquivo de formato JSON referente ao serviço em que a requisição está sendo realizada, verificar a rota solicitada, a caso no arquivo existam especificações de atributos, todos os atributos devem ser verificados e aprovados para que a requisição siga o caminho para o serviço solicitado.

Em casos em que o serviço de validação não aprove a requisição, deverá então retornar a resposta para o usuário com o código HTTP 405 "*METHOD NOT ALLOWED*" e uma especificação do motivo de sua solicitação ser negada, para que o usuário consiga compreender o motivo de receber o acesso negado ao serviço requisitado, todo esse processo ocorre sem que a requisição chegue ao serviço solicitado.

VII. Considerações finais e Trabalhos Futuros

Essa especificação, contribui para que ocorra um melhor entendimento no funcionamento do framework MiSSFire e também uma possível definição de padronização do controle de acesso baseado em atributos pela adição de um novo serviço e um arquivo de configurações especificado no formato JSON conforme apresentado na Subseção VI-A.

Por fim, um dos trabalhos futuros mais importantes é dar sequência na implementação do modelo de controle de acesso baseado em atributos no *framework* MiSSFire de acordo com

o apresentado na Seção VI, além de que o objetivo é que a implementação seja aceita por [1] e a nova funcionalidade seja incorporada ao *framework* MiSSFire.

Após a realização da implementação deve ser reproduzido o teste de desempenho executado por [1] para verificar a modificação na taxa de requisições por segundos a serem realizadas pela quantidade de *workers* na rede de microsserviços com duas novas linhas no gráfico, uma linha com o demonstrativo referente apenas ao uso de "*Tokens+ABAC*" e outra com o uso de "*Tokens+MTLS+ABAC*".

REFERENCES

[1] Yarygina, Tetiana, and Anya Helene Bagge. "Overcoming security challenges in microservice architectures." 2018 IEEE Symposium on Service-Oriented System Engineering (SOSE). IEEE, 2018.

[2] C. Pautasso, O. Zimmermann, M. Amundsen, J. Lewis, and N. Josuttis,"Microservices in practice (part 1): Reality check and service design,"IEEE Software, vol. 34, pp. 91–98, January-February 2017.

[3] Dragoni, Nicola, et al. "Microservices: yesterday, today, and tomorrow." Present and ulterior software engineering. Springer, Cham, 2017. 195-216.

[4] Sun, Yuqiong, Susanta Nanda, and Trent Jaeger. "Security-as-a-service for microservices-based cloud applications." 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom). IEEE, 2015.

[5] Nehme, Antonio, et al. "Fine-Grained Access Control for Microservices." International Symposium on Foundations and Practice of Security. Springer, Cham, 2018.

[6] Rajani, Vineet, Deepak Garg, and Tamara Rezk. "On access control, capabilities, their equivalence, and confused deputy attacks." 2016 IEEE 29th Computer Security Foundations Symposium (CSF). IEEE, 2016.

[7] Hu, Vincent C., et al. "Guide to attribute based access control (abac) definition and considerations (draft)." NIST special publication 800.162 2013.

[8] Ferraiolo, David, Janet Cugini, and D. Richard Kuhn. "Role-based access control (RBAC): Features and motivations." Proceedings of 11th annual computer security application conference. 1995.

[9] Attribute Based Access Control — NCCoE," Nist.gov, 2018. [Online]. Available: https://www.nccoe.nist.gov/projects/building-blocks/attribute-based-access-control. [Accessed: 30-Nov-2020]

[10] Yu, Dongjin, et al. "A survey on security issues in services communication of Microservices-enabled fog applications." Concurrency and Computation: Practice and Experience 31.22 (2019): e4436.

[11] Jin, Xin, Ram Krishnan, and Ravi Sandhu. "A unified attribute-based access control model covering DAC, MAC and RBAC." IFIP Annual Conference on Data and Applications Security and Privacy. Springer, Berlin, Heidelberg, 2012.

[12] Biswas, Prosunjit, Ravi Sandhu, and Ram Krishnan. "Label-based access control: An ABAC model with enumerated authorization policy." Proceedings of the 2016 ACM International Workshop on Attribute Based Access Control. 2016.

[13] B. Doerrfeld, "How to control user identity within microservices,"Nordic APIs blog, 2016, [Accessed Nov. 20, 2020]. https://nordicapis.com/how-to-control-user-identity-within-microservices/.

[14] Nadalin, M. Goodner, M. Gudgin, D. Turner, A. Barbir, andH. Granqvist, "OASIS Standard Specification. WS-Trust 1.4," Apr2012, [Accessed Dez. 8, 2020].

[15] Hemdi, Marwah, and Ralph Deters. "Using REST based protocol to enable ABAC within IoT systems." 2016 IEEE 7th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON). IEEE, 2016.

[16] V. C. Hu, D. Ferraiolo, R. Kuhn, A. Schnitzer, K. Sandlin, R. Miller, and K. Scarfone, "Guide to Attribute Based Access Control (ABAC) Definition and Considerations," National Institute of Standards and Technology, NIST SP 800-162, Jan. 2014.