Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Implementação do Protocolo OAuth2 e OpenID Connect em uma Arquitetura Orientada a Microsserviços

Murilo Góes de Almeida

Dissertação apresentada como requisito parcial para conclusão do
Mestrado Profissional em Computação Aplicada

Orientadora
Prof.a Dr.a Edna Dias Canedo

Brasília
2022

Ficha catalográfica elaborada automaticamente,
com os dados fornecidos pelo(a) autor(a)

GG598i

Góes de Almeida, Murilo
Implementação do Protocolo OAuth2 e OpenID Connect em uma Arquitetura Orientada a Microsserviços / Murilo Góes de Almeida; orientador Edna Dias Canedo. -- Brasília, 2022.
127 p.

Dissertação (Mestrado - Mestrado Profissional em Computação Aplicada) -- Universidade de Brasília, 2022.

1. Microsserviços. 2. Segurança. 3. Autenticação. 4. Autorização. 5. Engenharia de Software. I. Dias Canedo, Edna, orient. II. Título.

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Implementação do Protocolo OAuth2 e OpenID Connect em uma Arquitetura Orientada a Microsserviços

Murilo Góes de Almeida

Dissertação apresentada como requisito parcial para conclusão do
Mestrado Profissional em Computação Aplicada

Prof.a Dr.a Edna Dias Canedo (Orientadora)
Universidade de Brasília

Prof. Dr. Sérgio Antônio Andrade de Freitas
Universidade de Brasília

Prof. Dr. Eduardo Luzeiro Feitosa
Universidade Federal do Amazonas (UFAM)

Prof. Dr. Marcelo Ladeira
Coordenador do Programa de Pós-Graduação em Computação Aplicada

Brasília, 29 de agosto de 2022

Dedicatória

Dedico este trabalho a Deus, toda honra e toda glória, agora e para sempre.

Agradecimentos

Agradeço à minha orientadora, Professora Dra. Edna Dias Canedo, pelo pronto apoio em todos os momentos que necessitei de seu auxílio, seja na elaboração da dissertação como na publicação de artigos. Estendo os agradecimentos aos demais professores do mestrado, os quais sou muito grato pelo conhecimento adquirido.

Agradeço à Polícia Militar do Estado de São Paulo, instituição a qual tenho orgulho de pertencer, que me faz buscar diuturnamente novos conhecimentos para melhor servir ao cidadão. Amplio os agradecimentos aos policiais do Centro de Inteligência da PMESP, sobretudo ao Cel Res PM João Silva Soares Castilho, que autorizou a minha mobilização ao Ministério da Justiça, sem a qual não teria qualquer possibilidade de iniciar meus estudos na Universidade de Brasília (UnB). Agradeço ainda a meus superiores hierárquicos e subordinados que me auxiliaram direta ou indiretamente na conclusão deste trabalho, Ten Cel PM Denis Fernandes, Cap PM Nakano, Sgt PM Vanessa, Sgt PM Collura, Cb PM Sônia, Sd PM Deivide e Sd PM Lellis. Aos meus antigos chefes do Centro de Processamento de Dados, Cel Res PM Márcio Roberto de Campos e Maj PM Eduardo Fernandes Gonçalves, que me desafiaram a buscar novos conhecimentos na área de tecnologia.

Agradeço aos profissionais da Diretoria de Inteligência do Ministério da Justiça por me apoiarem na execução do mestrado e no incentivo à pesquisa, sobretudo a coordenadora Fernanda Leal Antonucci, que não mediu esforços para que minha jornada virasse realidade. Agradeço ainda aos demais coordenadores, Helio Wazlawosky, Quiteria Niksic e Frederico de Melo Aguiar, que incentivaram a continuidade dos estudos. Manifesto ainda os agradecimento aos demais amigos mobilizados na mesma coordenação. Carregarei no coração a nossa amizade pelo resto da minha vida.

Agradeço sobretudo à minha família, minha mãe Sílvia e meu pai Paulo, que desde a minha infância colaboraram para que eu tomasse gosto pelos estudos, meus irmãos Leonardo e Paula e meus avós. Todos foram fundamentais para que chegasse até este momento. Agradeço de coração à minha esposa Juliana, que deu todo o apoio e carinho possível nesta jornada.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (Capes), por meio do acesso ao Portal de Periódicos.

Resumo

O Centro de Inteligência da Polícia Militar do Estado de São Paulo (CIPM/PMESP) passou por um estágio evolutivo em seus sistemas legados, que atualmente se encontram em uma arquitetura de microsserviços. Tal arquitetura divide um aplicativo em pequenos serviços, que são implementados de forma independente, com sua própria unidade de deployment. Essa arquitetura pode trazer benefícios, no entanto, também apresenta desafios, principalmente quanto aos aspectos de segurança. Portanto, surge a importância de explorar o conhecimento sobre questões de segurança em microsserviços, principalmente nos aspectos de autenticação e autorização. Soma-se ainda a natureza da atividade de inteligência que obrigatoriamente deve atingir uma série de princípios por conta da natureza sensível dessa atividade. Um desses princípios é a compartimentação, que objetiva restringir o acesso a determinadas informações somente para profissionais que tenham necessidade de conhecer. Reforça-se assim que é preciso verificar se o ambiente atual se encontra adequado em questões de autenticação e autorização, para garantir o cumprimento de tal princípio.

Este trabalho tem o objetivo de propor e avaliar a segurança de uma solução orientada a microsserviços para realizar a autenticação e autorização nos Sistemas de Inteligência da Polícia Militar do Estado de São Paulo. Para tal, é feita uma Revisão Sistemática de Literatura para checar quais são os principais desafios em autenticação e autorização na arquitetura de microsserviço, os mecanismos que lidam com tais desafios e as tecnologias *open source* que implementam tais mecanismos. Com esses achados, é apresentado um questionário de validação, para verificar se tais descobertas são observadas na indústria. O objetivo é confirmar a utilização prática do que foi encontrado, além de procurar apurar novas respostas que não foram encontradas anteriormente. Por fim, é realizada uma execução no ambiente de inteligência da PMESP com a tecnologia open-source que utiliza os mecanismos encontrados, aplicando testes de segurança no ambiente atual e o novo, com vistas de verificar se esses mecanismos melhoraram sua segurança.

Palavras-chave: Microsserviços, Segurança, Autenticação, Autorização, Revisão Sistemática de Literatura

Abstract

The Intelligence Department of the Military Police of São Paulo State (CIPM/PMESP) has been evolving their legacy systems, which are currently in a microservices architecture. This architecture splits an application into small services, which are implemented independently, with their own deployment unit. The microservice architecture can bring benefits, however, it also presents challenges, especially regarding security aspects. Therefore, it brings a need to explore knowledge about security issues in microservices, especially in authentication and authorization aspects. In addition, the characteristic of the intelligence activity must necessarily reach a series of principles, such as compartmentalization, which aims to restrict access to certain information only to professionals who need to know. This reinforces the need to verify whether the current environment is adequate in terms of authentication and authorization, to ensure compliance with this principle. This work aims to propose and evaluate the security of a microservices-oriented solution to perform authentication and authorization in the Intelligence Systems of the Military Police of the State of São Paulo. To this end, a Systematic Literature Review is carried out to check what are the main challenges in authentication and authorization in the microservice architecture, the mechanisms that deal with such challenges and the open-source technologies that implement such mechanisms. With the SLR findings, a validation survey is carried out, verifying whether such findings are observed in the industry, to confirm the practical use of what was found, in addition to seeking to verify new answers that were not found in the SLR. Finally, an implementation is carried out in the PMESP intelligence environment with the open-source technology that implements the mechanisms found, applying security tests in the current and new environment, to verify if the applied mechanisms improved the security of the application.

Keywords: Microservices, Security, Authentication, Authorization, Systematic Literature Review

Sumário

1 Introdução **1**
1.1 Problema de Pesquisa . . . 3
1.2 Justificativa . . . 5
1.3 Objetivos . . . 6
1.3.1 Objetivo Geral . . . 6
1.3.2 Objetivos Específicos . . . 6
1.4 Resultados Esperados . . . 6
1.5 Metodologia de Pesquisa . . . 7
1.6 Publicações . . . 7
1.7 Estrutura do Trabalho . . . 7

2 Embasamento Teórico **9**
2.1 Sistemas Legados . . . 9
2.2 Arquitetura Orientada a Serviços . . . 9
2.3 Arquitetura Monolítica . . . 10
2.4 Arquitetura de Microsserviços . . . 10
2.5 Contextualização dos Sistemas Desenvolvidos pelo Centro de Inteligência . . . 11
2.5.1 Mapeamento de Dados das Áreas Negociais . . . 13
2.5.2 Migração dos Sistemas Legados . . . 14
2.5.3 Adoção de Microsserviços . . . 16
2.6 Trabalhos Relacionados . . . 19
2.7 Síntese do Capítulo . . . 22

3 Revisão Sistemática de Literatura **24**
3.1 Introdução . . . 24
3.2 Questões de Pesquisa . . . 25
3.3 Processo de Busca . . . 26
3.4 Processo de *snowballing* . . . 26

3.5 Critérios de Inclusão e Exclusão . . . 26
3.5.1 Critérios de Inclusão . . . 26
3.5.2 Critérios de Exclusão . . . 27
3.5.3 Avaliação da Qualidade . . . 27
3.6 Coleta e Análise de Dados . . . 27
3.7 Resultados da RSL . . . 28
3.7.1 Avaliação da Qualidade das Revisões Realizadas . . . 28
3.7.2 Fatores de Qualidade . . . 31
3.7.3 RQ.1. . . . 32
3.7.4 RQ.2. . . . 36
3.7.5 RQ.3. . . . 42
3.8 Discussões . . . 42
3.8.1 Limitações da RSL . . . 43
3.8.2 Considerações Finais da RSL . . . 43

4 Percepção dos Profissionais de TIC **49**
4.1 Planejamento . . . 49
4.1.1 Definição das Questões . . . 49
4.1.2 Definição do Processo . . . 49
4.1.3 Avaliação do Instrumento . . . 50
4.1.4 Seleção de Participantes . . . 50
4.2 Resultados do Questionário . . . 50
4.2.1 RQ1 . . . 51
4.2.2 RQ2 . . . 55
4.2.3 RQ3 . . . 56
4.3 Síntese do Capítulo . . . 57

5 Proposta **58**

6 Implementação **61**
6.1 Tecnologias e Bibliotecas Adotadas . . . 61
6.2 Vantagens e desafios percebidos . . . 62

7 Testes de Segurança **65**
7.1 Execução de Testes com Sonarqube . . . 65
7.2 Execução de testes com OWASP Zazp . . . 67
7.3 Execução de testes com Nikto . . . 69

8 Conclusões e Trabalhos Futuros **74**

Referências **76**

Apêndice **84**

A Questionário aplicado **85**

B Descrição das principais classes utilizadas na camada de Segurança do Sistema **93**

B.1 Classes utilizadas no microsserviço auth (Responsável pelo processo de autenticação) . . . 93

B.1.1 WebSecurityConfig.class . . . 93

B.1.2 ADAuthenticationManager.class . . . 94

B.1.3 TokenAuthenticationService.class . . . 98

B.1.4 JWTLoginFilter.class . . . 103

B.1.5 JWTAuthenticationFilter.class . . . 105

B.2 Classes utilizadas no microsserviço Gateway para checagem de autorização . 105

B.2.1 WebSecurityConfig.class . . . 105

B.2.2 JWTAuthenticationFilter.class . . . 106

C Descrição das principais classes utilizadas na nova Camada de Segurança **109**

C.1 Classes utilizadas no novo serviço de Gateway . . . 109

C.1.1 application.properties . . . 109

C.1.2 WebSecurityConfig . . . 110

D Descrição das principais classes utilizadas na nova Camada de Segurança aplicado ao *front-end* **111**

D.1 Trechos da implementação do Keycloak no *front-end* . . . 111

D.1.1 shared.module.ts . . . 111

D.1.2 auth.guard.ts . . . 112

Lista de Figuras

1.1 Processo atual de autenticação dos sistemas do Centro de Inteligência da PMESP 4
1.2 Processo atual de autorização dos sistemas do Centro de Inteligência da PMESP 4

2.1 Sistemas e bases de dados isoladas, desenvolvidos no Centro de Inteligência da PMESP 12
2.2 Nova aplicação monolítica orientada a serviços após o mapeamento dos dados das áreas negociais 14
2.3 Visão do ambiente com arquitetura de microsserviços 17

3.1 Processo de aplicação do protocolo 30
3.2 Filtros aplicados no processo da RSL. 30
3.3 Quantitativo dos mecanismos encontrados 37

4.1 Desafios observados em arquitetura de microsserviços 52
4.2 Protocolos utilizados em autenticação/autorização na arquitetura de microsserviços 55
4.3 Tecnologias *open source* que implementam os protocolos utilizados em autenticação/autorização na arquitetura de microsserviços 57

5.1 Sistemas e bases de dados isoladas, desenvolvidos no Centro de Inteligência da PMESP 60

6.1 Configuração de Serviço LDAP para autenticação no Keycloak 63
6.2 Tela de login com implementação do Keycloak 64

7.1 Resultado dos testes realizados no SonarQube no ambiente atual. 66
7.2 Categorias das severidades encontradas no ambiente atual. 66
7.3 Categorias de segurança encontradas no ambiente atual. 67
7.4 Resultado dos testes realizados no SonarQube no novo ambiente. 68
7.5 Categorias das severidades encontradas no ambiente novo. 68

7.6 Resultado dos testes realizados no OWASP Zap no antigo ambiente. 70
7.7 Resultado dos testes realizados no OWASP Zap no novo ambiente. 71
7.8 Resultado dos testes realizados no Nikto no ambiente atual. 72
7.9 Resultado dos testes realizados no Nikto no novo ambiente. 73

Lista de Tabelas

3.1 Estudos Selecionados 29
3.2 Ranqueamento de pontuação de acordo com as Avaliações de Qualidade . . 31
3.3 Média da pontuação de qualidade dos estudos por ano 32
3.4 Desafios relacionados com autenticação e autorização em microsserviços . . . 45
3.5 mecanismos de segurança utilizados em arquitetura de microsserviços 46
3.6 Tecnologias *open source* que implementam segurança em arquitetura de microsserviços 47
3.7 Vinculação dos Desafios, Mecanismos e Soluções open source 48

4.1 Número e descrição dos participantes 51
4.2 Desafios evidenciados em falhas de segurança observadas em projetos de TIC 54

5.1 Protocolos e Tecnologias no Sistema de microsserviços do CIPM/PMESP . . 58

6.1 Comparativo de número de classes e linhas de código 63

7.1 Comparação de quantitativo de problemas e nota atribuída pela ferramenta SonarQube 69

Capítulo 1

Introdução

A segurança pública no Brasil é dever do Estado, garantido pela Constituição Federal, ou seja, cabe ao poder público a responsabilidade de preservar a ordem pública, a segurança das pessoas e do patrimônio. Existem instituições responsáveis pela aplicação da lei criada pela Constituição Federal [1], cada qual com uma função específica no âmbito da segurança pública. O Brasil é uma república federativa, dividida em 26 estados mais o Distrito Federal. Cada estado possui o seu próprio sistema de segurança pública, com instituições responsáveis pela prevenção e repressão aos crimes. Os órgãos de Segurança Pública do Brasil estão definidos no artigo 144 da Constituição Federal [1]:

> Art. 144. A segurança pública, dever do Estado, direito e responsabilidade de todos, é exercida para a preservação da ordem pública e da incolumidade das pessoas e do patrimônio, através dos seguintes órgãos:
>
> I - polícia federal;
>
> III - polícia ferroviária federal;
>
> IV - polícias civis;
>
> V - polícias militares e corpos de bombeiros militares;
>
> VI - polícias penais federal, estaduais e distrital. (BRASIL, 1988).

A entidade responsável pela prevenção e repressão imediata de crimes se chama Polícia Militar (PM). A Polícia Militar do Estado de São Paulo (PMESP), possui um Centro de Inteligência, responsável pelas atividades de inteligência de segurança pública em seu território e faz parte do subsistema de inteligência de segurança pública do Brasil [2]. Tal unidade desenvolve e mantém os seus próprios sistemas, bem como gerencia a infraestrutura de tecnologia em que eles são hospedados. Todas essas atividades são desenvolvidas por policiais militares do serviço de inteligência, em função da sensibilidade dos dados que são processados e armazenados, a maioria deles sigilosos. Assim, o centro de inteligência não utiliza mão de obra terceirizada, exclusivamente policiais militares aptos para atuação

na atividade de inteligência. Dentro desta perspectiva, mostra-se difícil encontrar policiais com habilidades na área de tecnologia, pois tais qualificações não são requisitos no processo de admissão policial. Portanto, a disponibilidade de policiais com conhecimento de tecnologia atuando na área de inteligência policial é muito reduzida se comparado a outras instituições públicas. Assim, aqueles que possuem um certo conhecimento na área, quando se trata de desenvolvimento de sistemas, ainda é incipiente.

Em função do pequeno número de policiais disponíveis, ao final de 2017, os sistemas desenvolvidos por policiais do Centro de Inteligência da PMESP não seguiam nenhum tipo de padrão, tampouco possuíam integração com serviços ou bancos de dados. Muitos sistemas atuavam forma isolada, sem conversar entre si. Cada sistema possuía a sua própria autenticação, *front-end*, *back-end* e banco de dados. O mais preocupante foi o fato dos sistemas, mesmo de diferentes áreas de negócio dentro da inteligência policial, não se comunicarem entre si, nem com outros sistemas corporativos da Polícia Militar. Consequentemente, comportavam como "caixas-pretas", e, na maioria dos casos, armazenavam exatamente o mesmo tipo de dado em locais diferentes, o que causava várias redundâncias de forma desnecessária. Por último, mas não menos importante: algumas áreas negociais da inteligência não possuíam um sistema desenvolvido e costumavam armazenar dados importantes em arquivos como planilhas ou mesmo documentos impressos, com padrões estabelecidos pelos próprios analistas, sem qualquer tipo de integração com outras áreas.

Houve um processo de modernização desses sistemas legados supracitados, que hoje se encontram em uma arquitetura orientada a microsserviços. Isso proporcionou a integração entre diversos sistemas e bases de dados distintas, mesmos externas à área da inteligência, o que ajudou a resolver problemas de redundância no armazenamento de dados, interoperabilidade entre sistemas, reuso e manutenibilidade. O estilo arquitetural de um microsserviço é representado por um ecossistema de pequenos serviços. Cada um roda em um processo próprio e se comunica por mecanismos leves, geralmente uma API via protocolo HTTP, e são construídos em torno de recursos de negócio e implantados de forma independente [3].

Durante o processo de migração dos sistemas legados, além da modernização dos sistemas já existentes, foi realizado um mapeamento das necessidades da área negocial de inteligência. Assim, surgiram novas funcionalidades e desafios, sobretudo relacionados com a segurança dos sistemas. Neste cenário, este trabalho irá investigar as melhores práticas de segurança em microsserviços, mais especificamente nos aspectos de autenticação e autorização.

1.1 Problema de Pesquisa

A Polícia Militar do Estado de São Paulo dispõe de serviços de autenticação, como por exemplo o Active Directory (AD) e o "Módulo de Segurança" (MS), este último de desenvolvimento próprio da PMESP. Não existem políticas para padronização do uso desses autenticadores, tampouco ações para agrupar em uma plataforma única. Desta forma, encontramos alguns sistemas corporativos utilizando o AD e outros, o Módulo de Segurança. Importante salientar que os serviços mencionados atuam apenas no processo de autenticação. A autenticação é o processo de determinar se alguém ou algo é, de fato, quem afirma ou declara ser [4]. Autorização é o processo de dar permissão para alguém para fazer ou possuir algo [4]. Os aspectos relacionados à autorização acabam sendo responsabilidade das aplicações de forma individual. Neste sentido, cada aplicação disponibilizada na PMESP possui uma maneira própria para verificar se o usuário é autorizado a utilizar o sistema e quais funcionalidades. Verifica-se assim a importância de padronizar um acesso único aos sistemas da PMESP e também a forma em que são coletadas as autorizações do usuário para acessar determinado sistema ou funcionalidade interna.

No sistema desenvolvido para a área de inteligência, a camada de autenticação se caracteriza como um microsserviço dentro do ecossistema hospedado na infraestrutura do Centro de Inteligência da PM, conforme apresentado na Figura 1.1. O microsserviço realiza consulta ao serviço do Active Directory para verificar se o usuário é quem se declara e, caso positivo, verifica-se através de um banco de dados interno se ele possui permissão para utilizar a plataforma e após a confirmação, é montado um Json Web Token (JWT)[5], contendo as suas permissões no sistema. Este JWT é utilizado pelo cliente e trafega entre os diversos microsserviços, que também checam de maneira individual, se o usuário possui permissão para o acesso, conforme demonstrado na figura 1.2. Essa checagem se dá com a decodificação do JWT para verificação das regras de acesso do usuário, que ficam embutidas nele, além da própria validade do token, que expira a cada 12 horas de uso.

Como a aplicação possui uma arquitetura de microsserviços, existe a preocupação sobre aspectos de segurança, já que o acesso indevido em algum destes serviços pode comprometer o sistema como um todo. Deve se atentar em cada um, diferentemente da abordagem monolítica, onde essa preocupação se limita apenas a sua única unidade de *deployment*.

Dentro do mapeamento realizado junto às áreas negociais de inteligência de segurança pública, notou-se que alguns módulos do sistema poderiam ser acessados por policiais fora do âmbito da inteligência, ou mesmo externo à instituição. Todos os integrantes da Polícia Militar do Estado de São Paulo possuem credenciais nos autenticadores supramencionados. Todavia, não existe previsão para uso pelo público externo à instituição.

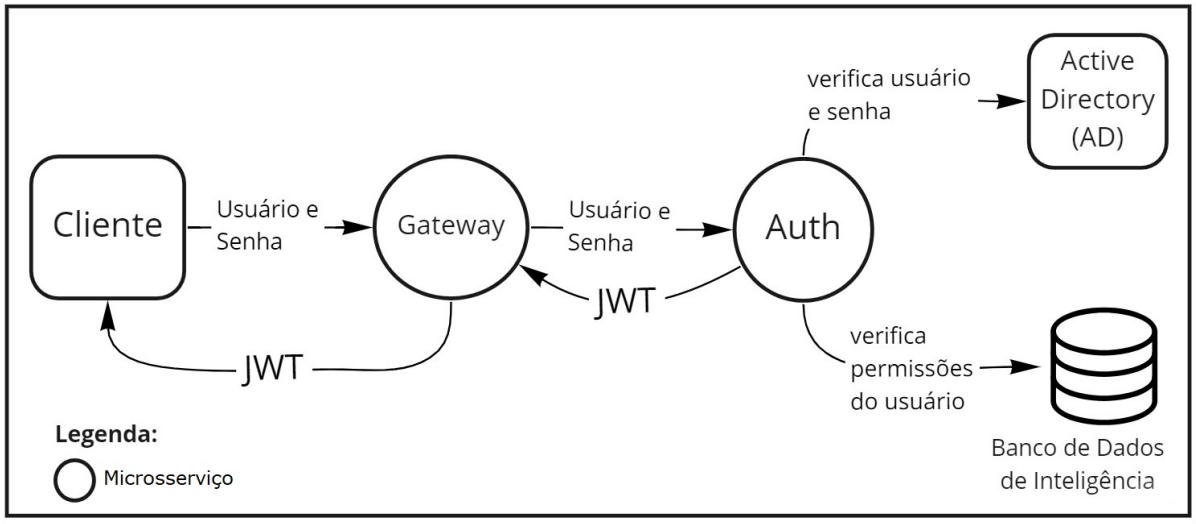


Figura 1.1: Processo atual de autenticação dos sistemas do Centro de Inteligência da PMESP

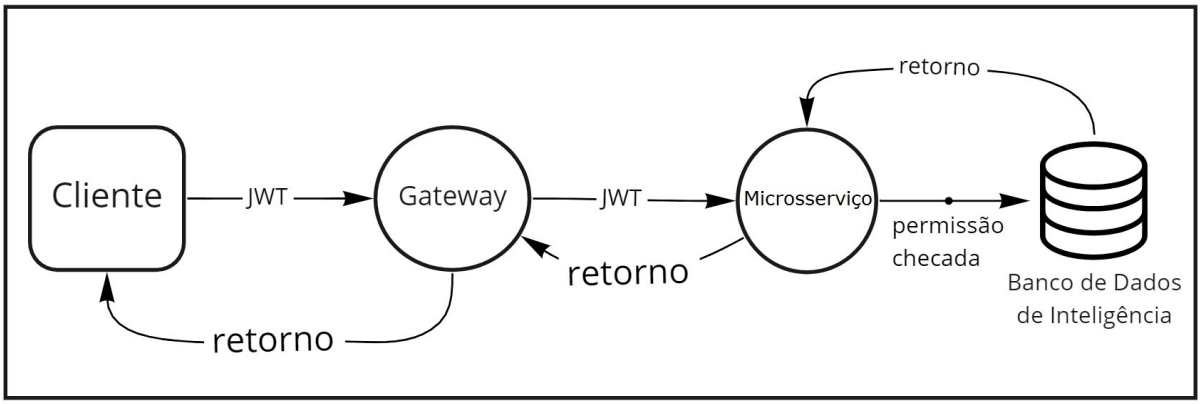


Figura 1.2: Processo atual de autorização dos sistemas do Centro de Inteligência da PMESP

Diante do cenário supracitado, verificou-se a necessidade de se implementar uma solução de segurança que ajude a lidar com os desafios de autenticação e autorização em um ambiente de microsserviços, proporcionando ainda uma padronização neste processo que possa ser utilizado não somente pelos sistemas desenvolvidos na área de inteligência, mas por toda a instituição PMESP. Tal implementação poderia ainda ser acessível no ambiente de internet para usuários externos à instituição e também por outros elementos internos fora da estrutura de inteligência. Essa solução possibilitaria ainda a integração com os autenticadores internos da instituição (AD e MS), além dos externos providos por aplicações consolidadas no ambiente de internet (Google, Facebook, Twitter).

Neste trabalho, será apresentado o panorama atual dos sistemas do Centro de Inteligência da Polícia Militar do Estado de São Paulo, os sistemas de autenticação disponíveis

no âmbito da instituição, levantar os principais desafios relacionados com autenticação e autorização em um ambiente de microsserviços. Outro objetivo é estudar os mecanismos de segurança que lidam com tais desafios e implementar os que forem mais adequados para o problema apresentado, preferencialmente utilizando tecnologia *open source*.

1.2 Justificativa

Existe a necessidade e interesse institucional da Polícia Militar do Estado de São Paulo (PMESP) em evoluir os seus sistemas, visando otimizar o processo de análise e diagnóstico dos problemas de segurança pública. O Sistema de Gestão da Polícia Militar do Estado de São Paulo (Gespol)[6] prevê o desenho de Tecnologia de Informação e Comunicação objetivando o acesso dos policiais aos bancos de dados e sistemas inteligentes. Todavia, determinados dados podem ser de natureza sigilosa, sobretudo na área de inteligência policial, devendo ser observado o conceito de contra-inteligência, definido pela Lei No 9.883/99 como "a atividade que objetiva neutralizar a inteligência adversa"[7].

O profissional de inteligência, dentro da sua área de atuação, deve respeitar alguns princípios que norteiam a sua atividade, tendo como um dos principais a compartimentação, que objetiva restringir o acesso a determinadas informações somente para profissionais que tenham necessidade de conhecer[8]. Desta forma, ao mesmo tempo que os sistemas precisam estar integrados e disponíveis, também devem ser cuidadosamente observados os critérios de acesso a eles, sobretudo em questões de autenticação e autorização.

Tal preocupação não se resume somente aos mecanismos de autenticação e autorização em que há uma interação direta do usuário, mas também dentro da arquitetura do sistema, sobretudo neste caso em tela apresentado pelo Centro de Inteligência da PMESP, onde é percebido o uso de microsserviços. Os microsserviços costumam ser desenhados de maneira que exista uma relação de confiança entre eles [9]. Dessa forma, a falta de observância aos aspectos de autenticação e autorização em um único microsserviço pode afetar todo o seu ecossistema. Independente da arquitetura implementada, os aspectos de autenticação e autorização são relevantes, considerando-os como elementos chave para os mecanismos de segurança[10].

Como a PMESP dispõe de serviços de autenticação próprios, que já são utilizado pelas maiorias das aplicações institucionais, embora sem critérios de padronização, entende-se que tal implementação também poderia ajudar a unificar o processo de autenticação e autorização de todas as aplicações institucionais, criando-se um portal de autenticação único para uso geral.

1.3 Objetivos

1.3.1 Objetivo Geral

O objetivo geral deste trabalho é propor e avaliar a segurança de uma solução orientada a microsserviços para realizar a autenticação e autorização nos Sistemas de Inteligência da Polícia Militar do Estado de São Paulo.

1.3.2 Objetivos Específicos

Para atingir o objetivo geral deste trabalho, os seguintes objetivos específicos foram definidos:

- Investigar a arquitetura, tecnologias e ferramentas utilizadas no desenvolvimento de sistemas no Centro de Inteligência da PMESP;
- Realizar uma Revisão Sistemática de Literatura (RSL) para identificar quais são os principais desafios envolvendo a autenticação e autorização em uma arquitetura de microsserviços, os mecanismos que lidam com tais desafios e soluções open-source que implementam os mecanismos levantados;
- Através da aplicação de um questionário, investigar com os profissionais de TI quais achados da RSL são mais utilizados na indústria;
- Selecionar dentre os achados da RSL e análise do questionário, os mecanismos para realização da autenticação e autorização aderentes ao contexto dessa pesquisa;
- Operacionalizar um diagnóstico do processo de autenticação e autorização implementado no sistema do Centro de Inteligência da PMESP;
- Implementar a solução proposta e avaliar a sua segurança executando testes automatizados;
- Realizar melhorias na solução proposta, caso necessário.

1.4 Resultados Esperados

Com a implementação dos mecanismos de autenticação e autorização, espera-se os seguintes resultados:

- Reduzir possibilidades de sucesso em ataques cibernéticos às aplicações institucionais da PMESP, avaliando a segurança da solução proposta em comparação à atual;

- Isentar a instituição de gastos financeiros para desenvolvimento da solução ou compra de software proprietário;
- Disponibilizar a solução proposta com maior facilidade de manutenção e evoluções futuras, com um modelo arquitetural simplificado e aderente aos novos padrões;
- Proporcionar segurança, performance, manutenibilidade e interoperabilidade entre a aplicação e demais sistemas institucionais.

1.5 Metodologia de Pesquisa

Foi realizada uma Revisão Sistemática de Literatura (RSL) de acordo com as diretrizes propostas por Kitchenham e Charles [11] e a estruturação aplicada por Kitchenham et al.[12]. A RSL se encontra no Capítulo 4 e ajuda buscar um melhor entendimento sobre as respostas das questões de pesquisa definidas neste trabalho.

Posteriormente, foi realizado um *survey* de acordo com o guia elaborado por Kitchenham [13], para verificar se os achados encontrados na RSL estão alinhados com as práticas da indústria, além de proporcionar a busca de novos desafios, mecanismos e soluções que lidam com autenticação e autorização no ambiente de microsserviços e possam ser úteis para responder as questões de pesquisa.

Após a análise dos resultados, foram selecionados os mecanismos e as respectivas soluções identificadas que estejam de acordo com os achados da pesquisa.

1.6 Publicações

Essa dissertação resultou nas seguintes publicações:

1. Authentication and Authorization in Microservices Architecture: A Systematic Literature Review MG de Almeida, ED Canedo Applied Sciences 12 (6), 3023.
2. The Adoption of Microservices Architecture as a Natural Consequence of Legacy System Migration at Police Intelligence Department MG de Almeida, ED Canedo International Conference on Computational Science and Its Applications, 354-369

1.7 Estrutura do Trabalho

Este trabalho está organizado em 6 capítulos, além deste, e consiste em:

- **Capítulo 2:** *Embasamento Teórico.* Apresenta os conceitos teóricos necessários para o entendimento deste trabalho, bem como a evolução arquitetural dos sistemas do Centro de Inteligência da Polícia Militar do Estado de São Paulo (PMESP).
- **Capítulo 3:** *Revisão Sistemática de Literatura.* Discorre sobre a Revisão Sistemática de Literatura, detalhando o mecanismo utilizado, o processo de execução e análise dos resultados;
- **Capítulo 4:** *Percepção dos Profissionais de TIC.* Descreve a elaboração, aplicação e análise do questionário sobre os achados da Revisão Sistemática de Literatura, com o objetivo de verificar os mecanismos mais utilizados na indústria e quais estariam mais aderentes ao sistema de inteligência da PMESP, de acordo com os profissionais de TIC.
- **Capítulo 5:** *Proposta.* Apresenta uma proposta de solução de autenticação e autorização em microsserviços baseada nos achados da RSL e na percepção dos profissionais que responderam ao questionário.
- **Capítulo 6:** *Conclusão e Trabalhos Futuros.* Apresenta a conclusão deste trabalho, bem como os trabalhos futuros.

Capítulo 2

Embasamento Teórico

Este capítulo apresenta uma revisão dos principais conceitos relacionados ao tema deste trabalho, tal como uma contextualização dos sistemas desenvolvidos no Centro de Inteligência da PMESP.

2.1 Sistemas Legados

Os sistemas legados podem ser definidos informalmente como "grandes sistemas de software que não sabemos como lidar, mas que são vitais para a organização"[14]. Geralmente, os sistemas legados se tornam muito difíceis de manter e evoluir com o tempo. No entanto, a maioria deles desempenha um trabalho crucial para a organização. Consequentemente, em um determinado momento, os sistemas legados custam muito esforço, causando um paradoxo, pois, embora sejam vitais para a organização, ao mesmo tempo estão causando diversos problemas para ela.

Bisbal et al. [15] sugeriram metodologias de migração, seja qual for a nova tecnologia ou arquitetura. Eles mencionam uma metodologia mais agressiva onde todo o sistema legado é abandonado e redesenhado do zero, porém, também é possível migrar aproveitando o que já existe. Também existem estudos sobre a migração para arquiteturas específicas, como a migração de sistemas legados para uma plataforma orientada a objetos [16] ou até mesmo a migração para arquiteturas mais modernas, como microsserviços [17].

2.2 Arquitetura Orientada a Serviços

O termo Arquitetura Orientada a Serviços (SOA) foi mencionado pela primeira vez na indústria pelo Gartner Group em 1996 [18]. Existem diferentes definições sobre Arquitetura Orientada a Serviços, no entanto, elas concordam que é um paradigma que melhora

a flexibilidade em sistemas [19]. SOA é uma arquitetura de software baseada em serviços, que são componentes de software fracamente acoplados que podem ser orquestrados para aumentar a agilidade de negócios [20]. A Arquitetura Orientada a Serviços traz algumas vantagens, como confiabilidade, escalabilidade, reusabilidade, acoplamento fraco, facilidade de manutenção, entre outras [20]. O conhecimento sobre SOA auxilia na compreensão da arquitetura de microsserviços, pois seus conceitos são relacionados [20].

2.3 Arquitetura Monolítica

A Arquitetura Monolítica é uma estratégia tradicional para o desenvolvimento de *software*, em que as funções do sistema são encapsuladas na mesma aplicação [21]. Se a aplicação monolítica for simples, é vantajosa, pois é fácil de desenvolver, testar e implantar [21]. Caso contrário, pode ser mais complicada de gerenciar em alguns cenários, pois ela pode compartilhar os recursos físicos da mesma máquina, como memória, bancos de dados ou arquivos. Um problema bastante notado nesta arquitetura está relacionado à escalabilidade e aos aspectos relacionados à mudança e evolução da aplicação [22].

Em outras palavras, se um aplicativo monolítico possui muitas funcionalidades, não é possível gerenciar individualmente a quantidade de recursos físicos que cada função deve consumir. Por fim, se alguma funcionalidade tiver que ser reparada ou evoluída, todo aplicativo haverá de ser afetado, pois se trata de uma aplicação única, não importa qual seja a função.

2.4 Arquitetura de Microsserviços

Microsserviços são pequenos sistemas escaláveis que representam recursos de negócios limitados e são frequentemente implantados de forma independente [22]. Cada microsserviço funciona como um aplicativo independente que representa uma única funcionalidade de negócios. A escalabilidade independente é um grande diferencial em relação ao monolito, pois neste caso, quando um microsserviço consome mais recursos, é fácil dimensionar os recursos físicos adequados para aquela aplicação específica [23].

É importante mencionar que os microsserviços também possibilitam desenvolvimento independente. Uma única unidade pode ser desenvolvida por uma equipe, que pode usar uma determinada linguagem de programação [23]. O desenvolvimento independente também traz outros benefícios, segundo Bucchiarone et al. [22], “Uma vez que cada microsserviço representa uma única capacidade de negócios, que é entregue e atualizada de forma independente, descobrir bugs ou adicionar pequenas melhorias não tem nenhum impacto em outros serviços e em seus lançamentos”. Apresentam algumas vantagens

em relação ao monolito, pois uma simples manutenção ou evolução pode afetar toda a aplicação [22]. Cada microsserviço pode ser desenvolvido e gerenciado por uma única equipe, ou seja, é possível ter todo um ecossistema de microsserviços e diferentes grupos trabalhando com cada aplicação, isolados [22].

A adoção da arquitetura de microsserviços pode trazer algumas desvantagens e desafios, como uma implantação mais complexa, desempenho (geralmente microsserviços se comunicam pela rede), consumo de recursos físicos e outros [24]. Portanto, a adoção da arquitetura de microsserviços nunca será necessariamente melhor do que o monolítico. Tudo irá depender de uma avaliação de cenário.

2.5 Contextualização dos Sistemas Desenvolvidos pelo Centro de Inteligência

O principal objetivo desta seção é apresentar o contexto tecnológico e arquitetural dos sistemas desenvolvidos pelo Centro de Inteligência da Polícia Militar do Estado de São Paulo (PMESP), que passou de uma fase de migração dos sistemas legados atingindo a arquitetura de microsserviços. Também traz as seguintes contribuições:

1. Em um processo de migração de sistemas legados, a arquitetura de microsserviços não é necessariamente a melhor opção a ser implementada imediatamente;
2. O uso de uma arquitetura monolítica aliada a tecnologias modernas ajuda a equipe de desenvolvimento a entender melhor as áreas de negócio e a própria tecnologia e arquitetura, que posteriormente podem ser transformadas em microsserviços;
3. A necessidade de usar uma arquitetura de microsserviços pode surgir naturalmente quando a arquitetura monolítica se torna insustentável, mas fácil de ser dividida em microsserviços;
4. Os detalhes sobre o processo de migração em um contexto de serviço público em que a equipe de desenvolvimento é pequena e com pouca experiência sobre o assunto;

Em função do reduzido número de policiais no trabalho de desenvolvimento de sistemas no Centro de Inteligência da PMESP, somado com o conhecimento limitado do efetivo em boas práticas de engenharia de software, ao final de 2017, os sistemas desenvolvidos não seguiam nenhum tipo de padrão, tampouco possuíam integração com serviços ou bancos de dados. Isso significa que haviam muitos sistemas legados que sequer "conversavam" entre si. Cada sistema possuía seu próprio sistema de autenticação, *front-end*, *back-end* e banco de dados.

O processo de desenvolvimento de software utilizado naquele momento envolvia o uso de uma ferramenta visual do tipo *scaffold*, que gerava automaticamente todo o código do sistema usando uma abordagem de *model first*, ou seja, baseado em um banco de dados previamente implementado. O *scaffold* pode ser uma boa solução para aprendizagem de programação e é encorajado para programadores novatos com conhecimento e experiência limitados [25]. Os policiais que trabalharam nesta época justificaram o uso deste processo pelo fato de haver um quadro de funcionários reduzido, além da expectativa das áreas negociais em receberem um sistema finalizado em um curto prazo.

Esse processo acelerou a operação de desenvolvimento e entrega dos sistemas, porém, não atendeu às expectativas dos usuários. A manutenção do sistema era muito complicada, pois o código gerado pela ferramenta era muito difícil de ler e editar. O mais preocupante foi o fato dos sistemas, mesmo de diferentes áreas de negócio dentro da inteligência policial, não se comunicarem entre si, nem com outros sistemas corporativos da Polícia Militar. Consequentemente, se comportavam como "caixas-pretas", e na maioria dos casos, armazenavam exatamente o mesmo tipo de dado em locais diferentes, o que causava várias redundâncias. Por último, mas não menos importante, algumas áreas de negócio de inteligência que não possuíam um aplicativo desenvolvido, costumavam armazenar dados importantes em arquivos como planilhas, sem qualquer tipo de integração com outras áreas. A Figura 2.1 mostra como os sistemas e banco de dados eram organizados neste contexto.

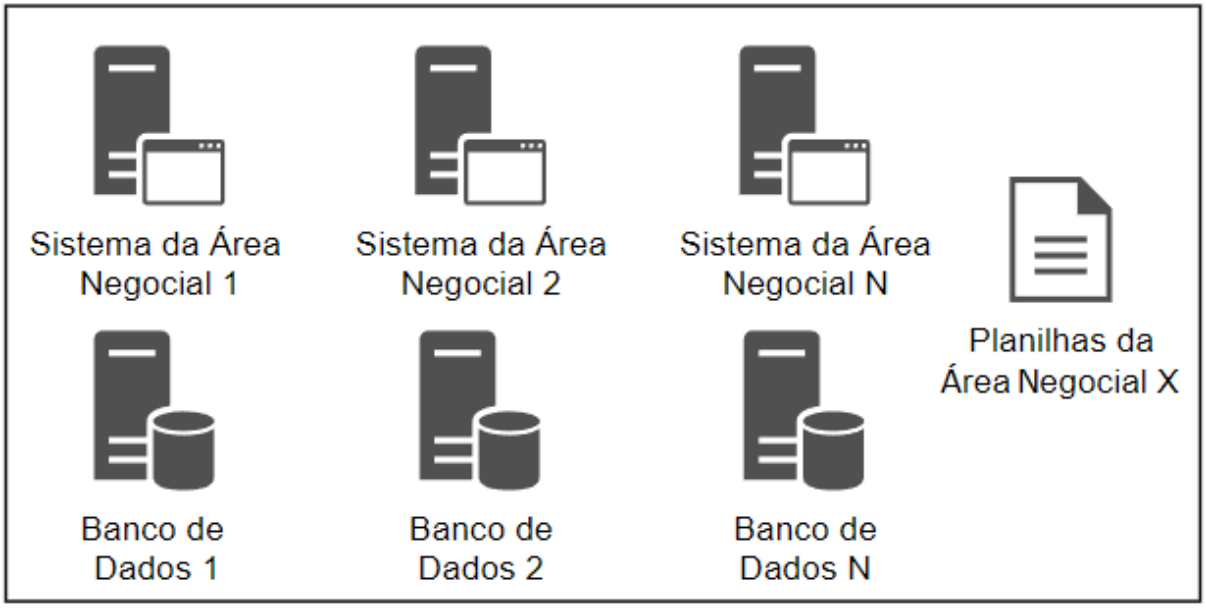


Figura 2.1: Sistemas e bases de dados isoladas, desenvolvidos no Centro de Inteligência da PMESP

De acordo com o cenário apresentado, os gestores do Centro de Inteligência da PMESP

entenderam a necessidade de modernização da arquitetura e tecnologias dos sistemas e bases de dados, com foco na integração de dados e a interoperabilidade dos sistemas.

2.5.1 Mapeamento de Dados das Áreas Negociais

A Polícia Militar do Estado de São Paulo possui uma Diretoria de Tecnologia da Informação e Comunicação (DTIC), separado da área de tecnologia do Centro de Inteligência e também é administrada por policiais militares. O setor, possui fábrica de softwares e infraestrutura próprias e é responsável por sistemas operacionais e administrativos das áreas de negócios da Polícia Militar do Estado de São Paulo (exceto inteligência), como policiamento operacional, recursos humanos, logística, saúde e financeiro. A DTIC difere da área de tecnologia do Centro de Inteligência por utilizar profissionais terceirizados, gerenciados por policiais.

Houve um consenso entre o nível estratégico de inteligência de que os sistemas desenvolvidos por este segmento deveriam armazenar apenas os dados relativos às suas próprias áreas de negócios, ou seja, apenas o que é produzido pelo serviço de inteligência. Dados fora do domínio, como por exemplo, recursos humanos, deveriam ser acessados por meio daquele sistema que armazena esses dados originais, utilizando algum tipo de integração sistêmica.

As fontes de dados das diferentes áreas de negócio externas à inteligência foram mapeadas e se verificou que a maioria delas estavam sob responsabilidade da Diretoria de Tecnologia da Informação da Polícia Militar. Houve possibilidade de acessar algumas dessas fontes de dados usando webservices já desenvolvidos por aquela diretoria. Desta forma, dados externos, que antes eram replicados em um banco de dados do Centro de Inteligência, passaram a ser acessados diretamente por meio de serviços e consequentemente o Centro de Inteligência deixou de armazenar dados redundantes, que consumiam volume de armazenamento desnecessário e por vezes se encontravam desatualizados.

Durante a atividade de mapeamento de processos de negócio na DTIC foi possível ainda constatar que estava disponível um serviço de autenticação que poderia ser integrado aos sistemas de inteligência, denominado Active Directory (AD). AD é um repositório comum de informações sobre objetos que residem na rede, como usuários, entre outros [26]. Concluímos que o processo de autenticação deveria ser providenciado pelo AD, pois os usuários e senhas armazenadas naquele serviço eram utilizados na maioria dos sistemas policiais, portanto, o usuário teria menos probabilidade de esquecê-lo e, mesmo que isso acontecesse, eles teriam que lidar diretamente com a DTIC, desonerando o Centro de Inteligência deste tipo de incumbência. Assim, os sistemas de inteligência deixaram de verificar a autenticação do usuário em banco de dados próprio, pois ela seria prestada pelo

serviço AD, tendo as bases e sistemas de inteligência o papel autorizador, ou seja, verificar se aquele usuário possui autorização para acessar a aplicação e quais funcionalidades.

O correto entendimento sobre quais tipos de dados estão relacionados ao serviço de inteligência policial possibilitou o desenvolvimento de um sistema mais flexível, integrado e sem redundância de dados, além de abrir mais descobertas para integração, como foi o caso observado na adoção do serviço de autenticação. O resultado desse mapeamento forneceu uma nova integração do projeto do sistema, conforme representado na Figura 2.2. Após esse processo, tanto a equipe de desenvolvimento quanto os tomadores de decisão reforçaram o entendimento de que a integração do sistema por meio de uma arquitetura orientada a serviços ajuda a reduzir a redundância de dados e os mantém com garantia de integridade.

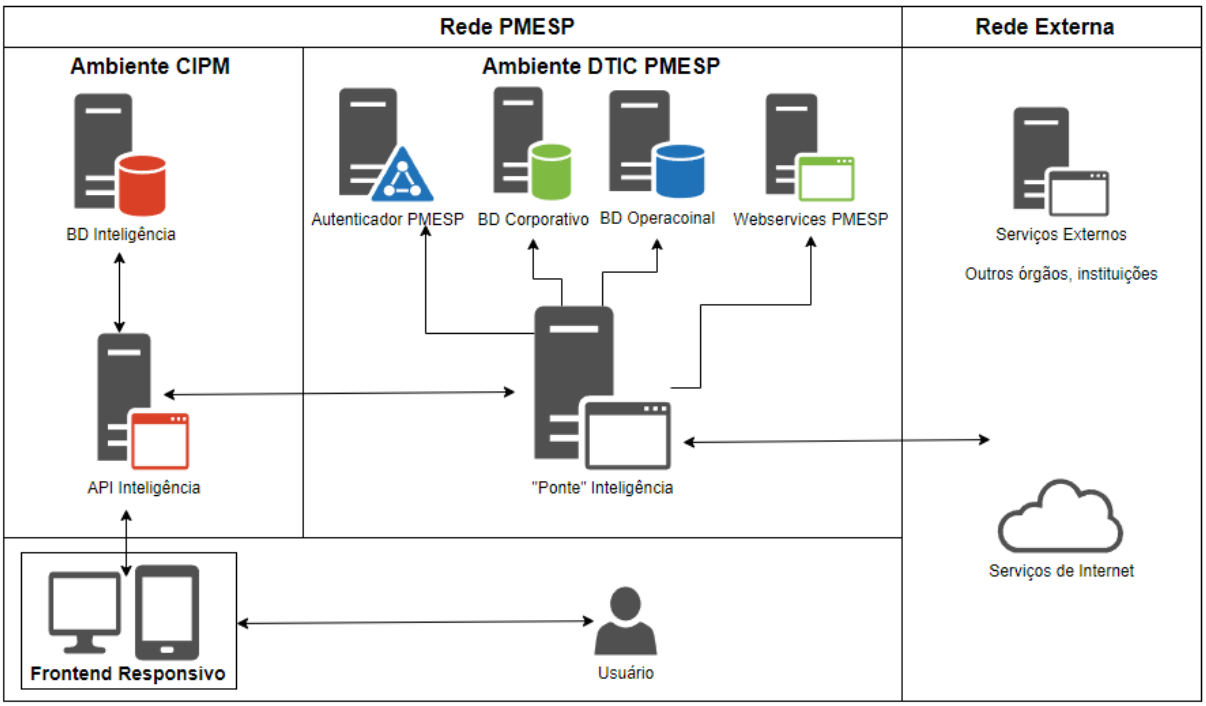


Figura 2.2: Nova aplicação monolítica orientada a serviços após o mapeamento dos dados das áreas negociais

2.5.2 Migração dos Sistemas Legados

Após o processo de entendimento e mapeamento dos dados das diferentes áreas negociais, foi proposta a construção de uma plataforma utilizando um único front-end que concentrasse todas as funcionalidades do sistema, mas o acesso do usuário estivesse de acordo com suas permissões. Diferente do sistema legado, houve uma separação total do sistema de*front-end* com o *back-end*, sendo este último uma *API RESTful*, capaz de se comunicar

com webservices externos de diferentes áreas de negócio e banco de dados. Em aplicações *RESTful*, os dados são trafegados utilizando o protocolo HTTP [27] e implementa REST (Representational State Transfer)[28], que utiliza a figura do recurso como uma abstração da informação a ser trafegada. O uso de uma abordagem *RESTFUL* pareceu fácil de entender e implementar. Viabiliza a integração com diferentes clientes, como um aplicativo móvel e aumenta as possiblidades além de um *front-end web*.

O uso de uma ferramenta *scaffold* foi abandonado, e a equipe de desenvolvimento passou a produzir seu próprio código, aproveitando esse momento para adotar um repositório GIT, que não existia desde então. GIT é um sistema de controle de versão (VCS) que permite o rastreamento das mudanças que são no código fonte da aplicação [29]. A equipe de desenvolvimento não sabia como funcionava o GIT, portanto, foi necessário treinamento, não só relacionado aos detalhes técnicos, mas também para criar consciência da importância de se trabalhar com versionamento de código em um repositório. Surgiu certa dificuldade de uso pela equipe no início, no entanto, houve consenso de que se trata de uma ferramenta poderosa que pode melhorar o desenvolvimento do código e documentação [29].

Nesse primeiro momento, não foi adotada a arquitetura de microsserviços, pois a equipe possuía pouco conhecimento sobre o assunto, somado com a dificuldade em saber modelar cada microsserviço de acordo com o contexto da aplicação. Por isso, preferiu-se adotar uma arquitetura monolítica, mas sem descartar uma possível migração futura. A organização do código-fonte foi cuidadosamente organizada de acordo com cada área negocial atendida ou funcionalidade desempenhada, principalmente os pacotes, classes e seus respectivos nomes. Tal separação contribuiu para que a equipe de desenvolvimento tivesse um maior entendimento do que realmente poderia ser quebrado em microsserviço, caso surgisse a necessidade, ao longo da evolução da aplicação.

Ao longo dos meses, novas funcionalidades foram adicionadas ao sistema, tanto de novas áreas de negócio que não estavam integradas na plataforma, como a evolução das existentes. Isso fez com que o projeto, mesmo bem organizado, passasse a ter um maior número de classes e linhas de códigos. A quantidade elevada de novas funcionalidades fez com que se levasse mais tempo para realizar o *build* da aplicação no ambiente de produção. Percebeu-se ainda, que uma simples mudança em um módulo específico, afetava todos usuários, sem distinção, pois todas as funcionalidades do sistema rodavam na mesma unidade de *deployment*. O grande número de recursos também dificultava o monitoramento de quais eram mais acessados pelos usuários, ou mesmo quais consumiam mais recursos de hardware. Se por algum motivo o aplicativo estivesse lento, ele era dimensionado na íntegra, sem a possibilidade de identificar o real motivo da lentidão.

Não havia um processo de integração contínua no sentido de automação de *builds*, ou

seja, quando era necessária uma atualização do sistema, a *build* era gerada e substituída pela antiga manualmente, o que tornava o sistema indisponível por um determinado momento. Essa indisponibilidade, mesmo que temporária, causava alguns incômodos, pois os usuários poderiam perder trabalhos que estivessem em andamento naquele momento. O processo manual de atualização de *builds* também se mostrava arriscado, pois, caso algum erro ocorresse, o processo de reversão também seria executado de forma manual, sem qualquer garantia de que funcionasse.

Por conta do aumento da complexidade do código-fonte e os motivos já elencados, embora houvesse um processo bem organizado, causava insegurança para a equipe de desenvolvimento na realização de manutenções ou evoluções, principalmente em relação aos desenvolvedores mais novos. A mesma insegurança também era presente por parte da equipe mais experiente, temendo que a transferência das responsabilidades do projeto para a equipe iniciante pudesse de alguma forma causar problemas que afetassem toda a aplicação. Por isso, a maior carga de trabalho ficou com os desenvolvedores experientes, que preferiram realizar ações que, em tese, a equipe iniciante seria incapaz de atuar.

Por essas razões, o uso da abordagem de microsserviços passou a ser considerado. A expectativa da equipe de desenvolvimento e dos tomadores de decisão era a de que quebrar esse monólito em microsserviços resolveria a maioria desses problemas e talvez não fosse arriscado. Isto porque naquele momento o projeto estava muito bem organizado e modularizado, somado com o conhecimento sobre as funcionalidades e áreas de negócios que já estavam mais claras para todas as pessoas envolvidas neste projeto.

2.5.3 Adoção de Microsserviços

A aplicação monolítica estava sendo desenvolvida com a linguagem de programação Java, utilizando-se o *framework* Spring Boot. De acordo com a documentação oficial, o Spring fornece tudo o que se precisa para adotar a linguagem Java em um ambiente corporativo [30] e, felizmente, este mesmo ecossistema Spring possui as suas próprias bibliotecas para a implementação de microsserviços, que são encontradas dentro da tecnologia *Spring Cloud*. *Spring Cloud* é um projeto abrangente composto por várias bibliotecas relacionadas com a construção de aplicativos baseados em nuvem e microsserviços [31]. Ele fornece, entre outras possibilidades, uma dependência chamada *Discovery Server*, que se comporta como um servidor onde os outros microsserviços podem realizar uma inscrição, sempre que é inicializado. O Spring Cloud também fornece outra dependência que funciona como um *API gateway* e também se inscreve no *Eureka Server*, verificando quais aplicativos estão inscritos nele e, finalmente, organiza todas as rotas a partir de um único endereço.

Houve considerável facilidade na adoção das tecnologias, pois a equipe de desenvolvimento já estava familiarizada com a linguagem de programação Java e o framework

Spring. A separação do projeto em microsserviços também não foi um obstáculo, pois, depois de um certo tempo trabalhando com a abordagem monolítica, tomando cuidando para organizar o código em pacotes e classes específicas de acordo com a área de domínio, ajudou a ter um maior entendimento de quais dessas funcionalidades deveriam se apresentar como um microsserviço independente. Dessa forma, a aplicação monolítica foi dividida a princípio em 13 (treze) microsserviços independentes. A Figura 2.3 mostra o novo ambiente usando microsserviços na infraestrutura de inteligência.

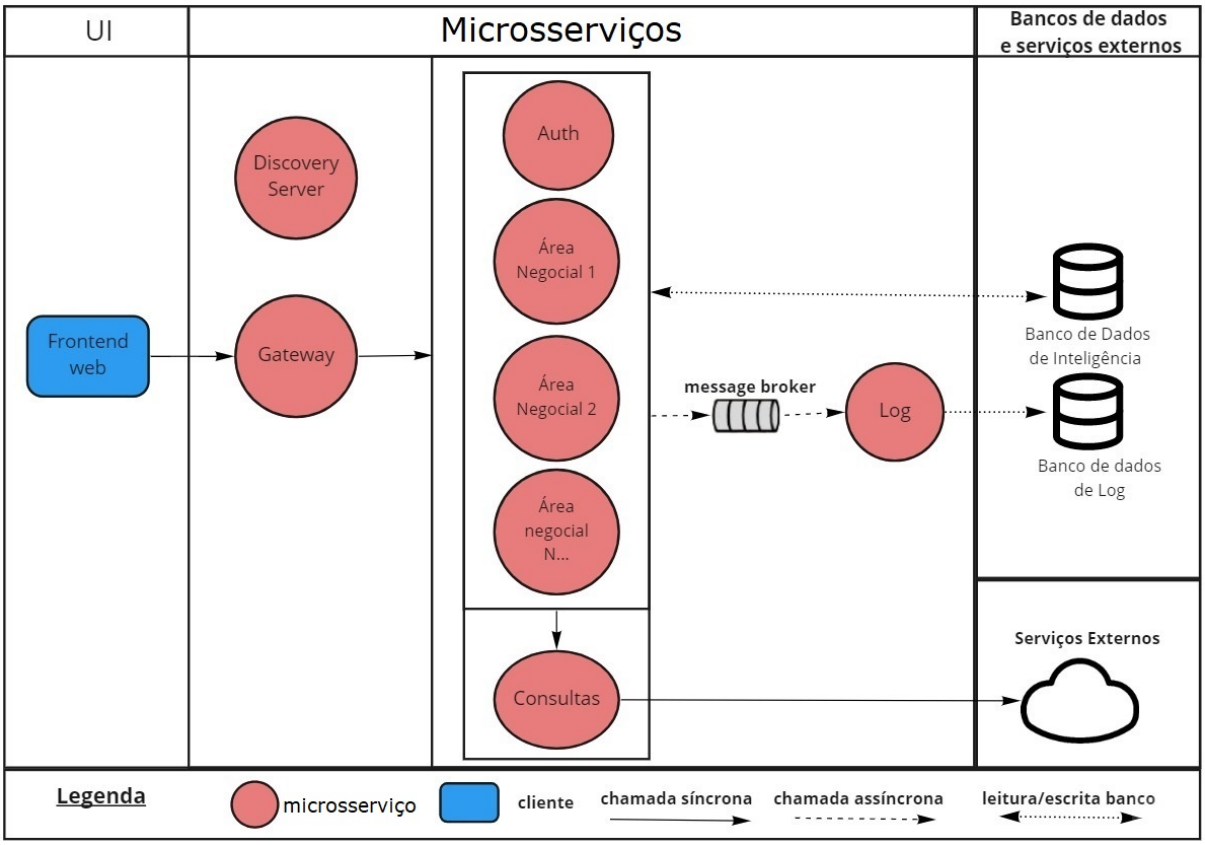


Figura 2.3: Visão do ambiente com arquitetura de microsserviços

Vantagens

O processo de migração para microsserviços levou a equipe de desenvolvimento a verificar quais ferramentas seriam mais adequadas para sustentar os aplicativos. Todos os microsserviços foram transformados em contêineres Docker, e o sistema operacional Windows Server foi substituído por uma distribuição Linux. O uso do Docker faz com que as aplicações sejam empacotadas com as dependências necessárias para o seu funcionamento e traz vantagens como velocidade, portabilidade, escalabilidade, e entrega rápida [32].

Também se aproveitou a oportunidade para resolver a questão da integração contínua. Devido à migração para a nova arquitetura, cada microsserviço passou a ter um repositó-

rio de código específico, portanto, optou-se por integrar o fluxo de trabalho GitFlow com um serviço Jenkins de integração contínua, criando-se um fluxo de *deploy* automático para cada microsserviço, dentro de um ambiente específico (desenvolvimento, homologação e produção). Dependendo da *branch* onde o código foi salvo, há o *deploy* automático dentro de determinado ambiente, como por exemplo, a subida de um código na *branch* denominada *feature* faria o *deploy* no ambiente de desenvolvimento, ou a *branch master* faria o *deploy* no servidor de produção. O Jenkins é uma ferramenta de integração contínua que pode simplificar e acelerar a entrega, ajudando a automatizar o processo de implementação do sistema [33]. O uso de tal ferramenta permitiu automatizar a construção, teste e implementação de cada microsserviço, facilitando muito o processo de entrega contínua e adoção do GitFlow, que é uma abstração importante para o gerenciamento de estado de fluxo que permite atingir níveis mais elevados de segurança, proveniência, facilidade de programação e suporte para vários aplicativos [34]. Além de colaborar no processo em si, enriqueceu a visão da equipe de desenvolvimento sobre boas práticas de versionamento de código.

Dividir o aplicativo monolítico em partes menores também ajudou os desenvolvedores mais experientes a ter maior confiança em disponibilizar o código-fonte para que os iniciantes o explorassem, pois alguns desses novos microsserviços de maneira isolada se tornaram simples de entender e manipular. Houve consenso de que, caso fossem feitas alterações incorretas de código em algum microsserviço e este fosse publicado em ambiente produtivo, o impacto seria mínimo, pois apenas aquele específico seria afetado, o que de fato chegou a acontecer, sendo fácil reverter para o estado anterior. Com isso, cada desenvolvedor ficou responsável por uma aplicação específica, de acordo com seu conhecimento técnico. Aumentou a produtividade e fez com que toda a equipe se sentisse mais envolvida com o projeto e mais resoluta em tornar seu microsserviço mais bem fatorado e evoluído.

O uso de microsserviços em contêineres facilitou o gerenciamento da escalabilidade, possibilitando uma distribuição mais racional dos recursos de hardware entre eles. Durante a época em que o sistema era monolítico, ficava evidente que certas funcionalidades do aplicativo eram menos utilizadas em relação a outras, e após a separação, houve uma melhor distribuição de recursos como processamento e memória em aplicações específicas, ao contrário de outras que não exigiam tantos recursos computacionais.

Desvantagens e desafios

Apesar de todas as vantagens mencionadas, após a migração completa no ambiente de produção da arquitetura monolítica para os microsserviços, alguns novos problemas começaram a ser identificados. A primeira aconteceu quando se percebeu que um microsserviço

específico consumia uma quantidade expressiva de memória e afetava também aqueles que se comunicavam com ele. Essa dificuldade evidenciou lentidão na comunicação com o banco de dados, o que até então não havia sido observado na abordagem monolítica, visto que este utilizava um recurso computacional superior ao que é utilizado em cada contêiner individualmente. O momento foi aproveitado para melhorar a qualidade das consultas realizadas no banco de dados, além de otimizar o gerenciamento dos *pools* de conexão. Desta forma, o que parecia uma dificuldade, acabou se revelando vantajoso, pois a equipe de desenvolvimento adquiriu novos conhecimentos e superou um desafio que dificilmente seria visto no cenário monolítico, causando uma refatoração positiva no código-fonte e melhor configuração de comunicação com o banco de dados.

Igualmente importante notar é o fato de que cada microsserviço funciona como uma aplicação independente, portanto, depende de uma quantidade mínima de recursos físicos, principalmente memória. Assim, se a aplicação possui um grande número de microsserviços, os recursos físicos computacionais disponíveis devem ser maiores em comparação com a arquitetura monolítica. Isso foi observado durante este processo porque a máquina virtual que hospeda o aplicativo monolítico tem 16 GB de memória RAM em comparação com 20 GB na máquina virtual que executa o ecossistema de microsserviços.

Por fim, como estes microsserviços se comunicam através de chamadas HTTP em um ambiente de rede, foi necessário aprofundar o entendimento sobre chamadas síncronas e assíncronas, algo que até então não era percebido. Para tal, foi implementado um serviço de fila com a biblioteca Apache Kafka, uma plataforma de código aberto que consegue operacionalizar as requisições assíncronas[35].

2.6 Trabalhos Relacionados

Esta seção tem o objetivo de apresentar trabalhos relacionados com o exposto nesse capítulo.

Luz et al. [23] estudou a adoção de microsserviços em instituições governamentais no Brasil. Eles argumentam que há muitos benefícios na adoção de microsserviços, incluindo maior escalabilidade, produtividade e capacidade de manutenção. No entanto, pode ser uma tarefa desafiadora nas Instituições Governamentais, principalmente porque exige não apenas o entendimento de novas técnicas e ferramentas, mas também aumenta a necessidade de automatizar tarefas relacionadas à implantação e monitoramento de software.

Também concluem que é difícil quebrar um sistema monolítico em microsserviços, principalmente porque faltam diretrizes com descrições mais precisas sobre a granularidade esperada em um microsserviço. De fato, tanto as vantagens quanto as desvantagens foram enfrentadas no processo de migração da PMESP, sendo uma grande dificuldade re-

lacionada às habilidades da equipe de desenvolvimento e como modelar os microsserviços, que foram alcançadas durante o processo de modernização.

Agilar et al. [36] estudaram uma Abordagem Orientada a Serviços para a Modernização de Sistemas Legados em uma instituição universitária do governo. A utilização da Arquitetura Orientada a Serviços (SOA) é mencionada como forma de solucionar a modernização do sistema, utilizando a arquitetura Representational State Transfer (REST). Em linhas gerais, o uso de REST se justifica pela simplicidade e facilidade de entendimento, além do fato de os sistemas em estudo serem capazes de trabalhar com requisições HTTP. A utilização de uma Arquitetura Orientada a Serviços com REST no Departamento de Inteligência se justificou com tais argumentos.

Carvalho et al. [37] realizaram um estudo sobre a extração de microsserviços configuráveis e reutilizáveis de sistemas legados, entrevistando especialistas que atuaram no processo de migração de arquitetura monolítica para microsserviços. A maioria dos especialistas entrevistados respondeu que a variabilidade é útil ou muito útil para extração de microsserviços. Variabilidade é a capacidade de criar variantes de sistema para diferentes segmentos de mercado ou contextos de uso[38]. Eles também observaram que a extração de microsserviços pode aumentar a personalização do software. No caso dos sistemas legados da inteligência da PMESP, ao final, percebemos que a maioria dos microsserviços construídos estava relacionada a um segmento de negócio específico da atividade de inteligência policial, facilitando a personalização e reuso.

Agilar et al. [39] escreveram um mapeamento sistemático sobre modernização de sistemas legados. De acordo com o trabalho, a principal contribuição do estudo é caracterizar a modernização de software de acordo com a literatura existente. Discutindo os termos relacionados, classifica as contribuições de pesquisas relacionadas e apresenta os principais motivos que incentivam um esforço de modernização de software (também de acordo com a literatura). Embora a literatura apresente uma grande quantidade de publicações sobre modernização de software, elas geralmente foram propostas sem qualquer avaliação prática, ou seja, são poucos os estudos que relatam experiências de sucesso na modernização de sistemas legados.

O processo de migração realizado no Departamento de Inteligência da Polícia Militar de São Paulo não se baseou em um método ou técnica específica divulgada, mas foi feito de forma empírica de acordo com a real necessidade de modernização. A experiência dos profissionais durante o processo de migração ajudou a equipe a perceber que um sistema legado pode ser modernizado inicialmente usando uma abordagem monolítica e depois de algum tempo, poderia ser necessário o uso de microsserviços, que seriam executados com facilidade devido a um melhor domínio do negócio aprendido durante o processo de modernização.

Laigner et al. [40] realizaram um estudo sobre migração de sistemas legados de Big Data para Arquitetura Event-Driven em Microsserviços. Aplicaram pesquisa-ação e investigaram os motivos para a adoção desta arquitetura. Intercederam para definir a nova arquitetura e documentaram os desafios e as lições aprendidas. Entre as conclusões mencionadas no artigo, vale ressaltar que, em suas palavras, definir um microsserviços muito cedo no processo de desenvolvimento, pode resultar em uma definição equivocada. Embora os microsserviços tenham muitos benefícios, se os requisitos não forem suficientemente maduros, podem trazer mais problemas do que vantagens.

Mazzara et al. [41] realizaram um estudo de caso baseado no FX Core, um sistema de missão crítica do Dankse Bank, o maior banco da Dinamarca. O processo de migração passou por etapas semelhantes às identificadas no departamento de inteligência policial de São Paulo. Eles também preferiram iniciar as funcionalidades em uma única unidade de aplicação, que em suas palavras, "permite que a equipe e a organização uniformizem a visão, mas também o entendimento da abordagem específica e do padrão de codificação". Após esta etapa, a divisão em novos serviços será natural, como foi experimentado neste estudo. Outra observação importante é a "atração" da arquitetura de microsserviços com DevOps. Mazzara et al. [41] relataram que "devops e microsserviços parecem ser um par indivisível para organizações que visam entregar aplicativos e serviços em alta velocidade". Quando começamos a migrar o sistema monolítico para a arquitetura de microsserviços, não estávamos pensando em DevOps, porém, percebemos que seria necessário implementar algumas ferramentas para otimizar o processo de manutenção e implantação, o que foi realizado.

Fritzsch et al. [42] investigaram o processo de migração de 14 sistemas em diferentes domínios e tamanhos. Eles concluíram que a intenção de migrar um sistema legado é superar as dificuldades de manutenibilidade, que vem acompanhada de alguns sintomas como custos, mudanças caras ou propensas a causar efeitos colaterais e dificuldade de análise. Há também problemas de rastreabilidade, longos tempos de inicialização e tempo de inatividade durante as atualizações, além de dificuldades para aplicar atualizações em geral. Este problema é bastante semelhante ao que foi relatado sobre a situação dos sistemas legados no Departamento de Inteligência. Eles também analisaram o motivo dessas empresas escolherem microsserviços. Os principais argumentos utilizados foram: escalabilidade da arquitetura, equipes de desenvolvimento, visando unidades menores, mais gerenciáveis e de fácil manutenção. Esses argumentos também convergem com o que foi relatado anteriormente, ou seja, dividir o sistema em pequenas unidades de aplicação facilitaria a distribuição de responsabilidade para a equipe de desenvolvimento, além de melhorar a manutenibilidade de cada serviço.

Por fim, é muito importante mencionar a abordagem *Monolith First* de Fowler [43],

que alerta que a adoção direta em uma arquitetura de microsserviços é arriscada. Ele explica que um monolito permite explorar tanto a complexidade de um sistema quanto os limites de seus componentes. Ele fornece algumas maneiras de executar uma estratégia de monolito em primeiro lugar, como projetar um monolito cuidadosamente, prestando atenção à modularidade dentro do software, tanto nos limites da API quanto na forma como os dados são armazenados. A maneira como a arquitetura monolítica foi desenvolvida durante a migração do sistema legado ajudou a quebrar a aplicação da inteligência da PMESP em microsserviços. Isto porque, ao ter a devida atenção nessa etapa fez com que a equipe de desenvolvimento tivesse mais conhecimento sobre as funcionalidades do sistema e áreas de negócio. Consequentemente, o processo de migração tornou-se mais simples de ser executado.

2.7 Síntese do Capítulo

Com base no que foi exposto neste capítulo, verificou-se que em determinado momento, os sistemas legados devem ser evoluídos, pois se tornam muito difíceis de entender e prestar manutenção e por vezes são de difícil uso. Durante a migração para tecnologias e arquiteturas modernas, a adoção de microsserviços não é necessariamente a única forma de evolução, embora essa abordagem traga muitos benefícios que são mencionados neste capítulo.

A adoção da arquitetura monolítica não significa que o sistema esteja desatualizado. Nesse caso, foi um bom ponto de partida para implementação de novas tecnologias e arquiteturas orientadas a serviços, como o padrão REST. O processo de migração no caso em tela ajudou a compreender melhor as áreas de atuação da inteligência e as demais da PMESP. Dessa forma, os microsserviços não foram adotados como ponto de partida, mas uma consequência inevitável em todo o processo de migração. Tornaram-se uma necessidade natural no momento em que surgiram certas dificuldades na arquitetura monolítica, que passaram a dificultar o processo de desenvolvimento, seja no uso de recursos humanos ou tecnológicos.

A adoção inicial da arquitetura monolítica, tomando os cuidados necessários na organização do código-fonte no projeto, possibilitou uma maior facilidade na construção dos microsserviços. Possibilitou um entendimento mais claro do que realmente deve ser um microsserviço.

A adoção final desta arquitetura possibilitou as vantagens de escalabilidade, flexibilidade, controle, monitoramento, automação de implementação, maior envolvimento da equipe de desenvolvimento e aumento de produtividade. Ocasionou considerável evolu-

ção nos sistemas desenvolvidos e mantidos pelo Departamento de Inteligência da Polícia Militar do Estado de São Paulo.

Capítulo 3

Revisão Sistemática de Literatura

Neste capítulo foi realizada uma Revisão Sistemática de Literatura (RSL), de acordo com as diretrizes propostas por Kitchenham e Charles [11] e Kitchenham et al.[12]. Uma RSL é um meio para identificar, avaliar e interpretar todos os trabalhos disponíveis relevantes para uma questão de pesquisa específica, ou área de tópico, ou fenômeno de interesse" [11]. Além disso, foi utilizada a ferramenta online Parsifal [44] para apoiar na triagem e análise dos estudos identificados durante a condução da RSL.

3.1 Introdução

O estilo de arquitetura de microsserviços é representado por um ecossistema de pequenos serviços, cada um executando em seu próprio processo e se comunicando por meio de protocolos leves, como HTTP (Hypertext Transfer Protocol), construídos em torno de recursos de negócios e implantados independentemente [3]. Dividir um aplicativo em microsserviços pode trazer alguns benefícios, como otimizar o gerenciamento, escalabilidade, disponibilidade e confiabilidade [24, 45]. No entanto, existem desafios em relação à segurança, pois, neste caso, deve-se observar uma atenção individual em cada microsserviço desenvolvido. É diferente do estilo monolítico onde as estratégias de segurança são aplicadas em uma única vez [45, 46]. Além disso, existem poucas demonstrações práticas na literatura que descrevam soluções para melhorar a segurança de arquiteturas orientadas a serviços [46].

Independentemente da arquitetura implementada, os aspectos de autenticação e autorização são relevantes, considerando-os como elementos chave para os mecanismos de segurança [10]. Autenticação é o processo de determinar se alguém ou algo é, de fato, quem eles afirmam ser. Autorização é o processo de dar a alguém ou algo permissão para fazer ou possuir algo [4]. Existem mecanismos que tratam de questões de autorização e autenticação, como o OAuth 2.0, o padrão para autorização delegada, e o OpenID Con-

nect, a camada de autenticação sobre o OAuth 2.0 [9]. É importante observar que há uma distinção entre autenticação de usuário e autenticação de serviço. No caso de autenticação entre microsserviços, existem mecanismos específicos para isso, como o Mutual Transport Layer Security (MTLS) [9]. Usando MTLS, cada microsserviço identificará legitimamente com quem está falando, além de garantir a confidencialidade e integridade dos dados nesta comunicação [47].

De acordo com alguns estudos, os microsserviços geralmente são projetados de forma que haja uma relação de confiança entre eles [45, 48]. No entanto, é possível encontrar arquiteturas de microsserviços que usam o paradigma "confiança zero"[49]. Neste último caso, existe a premissa de que a confiança nunca é concedida de forma implícita, mas deve ser continuamente avaliada [50]. Assim, a falta de observação sobre autenticação e autorização em um único microsserviço pode afetar todo o ecossistema. É importante que estudos relacionados a questões de segurança em microsserviços enfatizem aspectos que envolvam autenticação e autorização. Portanto, realizamos uma Revisão Sistemática de Literatura (RSL) para identificar na literatura os estudos que abordam autenticação e autorização em ambientes de microsserviços, quais são seus desafios, mecanismos de segurança utilizados para lidar com esses desafios e tecnologias de código aberto que implementam os mecanismos identificados na revisão. O foco no código aberto é fornecer tecnologias que possam reduzir custos, acesso gratuito ao código-fonte e personalização [51]. Existem vantagens para o uso de código aberto no setor público, como por exemplo, evitar o domínio do monopólio no mercado [51]. Até mesmo softwares desenvolvidos por empresas comerciais também estão sendo lançados sob licenças de código aberto [52]. É importante notar que a adoção do código aberto, embora tenha a vantagem de ser livre, não necessariamente trará um custo/benefício adequado para a organização [53]. Portanto, recomenda-se que sua opção seja baseada em métricas como o Custo Total de Propriedade (TCO), instrumento que avalia o custo de adaptação, gerenciamento e manutenção do software proposto [53].

3.2 Questões de Pesquisa

Conduzimos a RSL com o objetivo de responder as seguintes questões de pesquisa (RQ):

RQ.1: Quais são os desafios relatados na literatura em relação a autenticação e autorização no contexto dos sistemas de arquitetura de microsserviços?

RQ.2: Quais são os mecanismos utilizados na literatura para lidar com os desafios de autenticação e autorização em arquitetura de microsserviços ?

RQ.3: Quais são as principais soluções de tecnologia *open source* que implementam os mecanismos de autenticação e autorização identificados na literatura?

3.3 Processo de Busca

Para identificar os estudos na literatura, foi realizada uma busca automática nas principais bases digitais da área de Ciência da Computação. As bases digitais utilizadas na revisão sistemática de literatura foram: DBLP (https://dblp.uni-trier.de/); IEEE Digital Library (http://ieeexplore.ieee.org); Scopus (http://www.scopus.com).

A string de busca utilizada nas bases digitais foi definida de acordo com as palavras chaves que devem obrigatoriamente aparecer nos resultados:

(“MICROSERVICE"OR “MICROSERVICES") AND (“SECURITY"AND “AUTHENTICATION"AND “AUTHORIZATION") AND (“CHALLENGE"OR “PROBLEM*"OR “ISSUE*"OR “SOLUTION*"OR “PROTOCOL*"OR “MECHANISM*"“STRATEG*"OR “IMPLEMENTATION*"OR “OPENSOURCE"OR “OPEN-SOURCE"OR “OPEN SOURCE").*

3.4 Processo de *snowballing*

A aplicação do *snowballing* tem como objetivo evitar que estudos relevantes sejam omitidos [54]. Nesse processo, são verificadas referências sobre o objeto da pesquisa em cada estudo selecionado. Assim, nós buscamos novos artigos onde os estudos selecionados foram citados.

3.5 Critérios de Inclusão e Exclusão

Os critérios de seleção dos estudos primários procuram identificar os estudos que forneçam informações sobre as questões de pesquisa. Assim, foram definidos os seguintes critérios de inclusão e exclusão, baseados nas questões de pesquisa:

3.5.1 Critérios de Inclusão

- Estudos que tratam de desafios envolvendo autenticação e autorização em microsserviços;
- Estudos relacionados aos mecanismos de segurança que lidam com problemas de autenticação e autorização em microsserviços;

- Estudos relacionados às tecnologias *open source* que implementam mecanismos ou estratégias de segurança.

3.5.2 Critérios de Exclusão

- Estudos que não abordam o objeto de pesquisa;
- Estudos anteriores a 2010;
- Estudos duplicados;
- Estudos publicados como *short paper*.

3.5.3 Avaliação da Qualidade

Para diferenciar os estudos selecionados de acordo com critérios de qualidade, foi verificado em cada estudo selecionado, se eles respondem as questões de pesquisa. Os critérios adotados foram:

1. O objetivo da pesquisa está claramente descrito?
2. Os autores descrevem a limitação do estudo?
3. O estudo identifica desafios envolvendo autenticação e autorização em arquitetura de microsserviços?
4. O estudo identifica os mecanismos que mitigam os desafios envolvendo autenticação e autorização em arquitetura de microsserviços?
5. O estudo apresenta soluções que implementam os mecanismos de segurança utilizando tecnologia *open source*?

A resposta de cada pergunta do critério de qualidade recebeu uma pontuação, conforme: 1) Sim (1); 2) Parcialmente (0.5); 3) Não (0). Embora os estudos primários tenham sido selecionados através de critérios específicos, há uma avaliação individual da qualidade de cada estudo para verificar quais deles se encontram com maior alinhamento às questões de pesquisa que foram definidas (Seção 3.2).

3.6 Coleta e Análise de Dados

Foram coletados os seguintes dados nos estudos primários selecionados: 1) Os desafios de autenticação e/ou autorização encontrados em microsserviços; 2) Os mecanismos que

lidam com os desafios de autenticação e/ou autorização encontrados em microsserviços; 3) As tecnologias *open source* que implementam os mecanismos que lidam com os desafios de autenticação e/ou autorização em microsserviços.

Os desafios, mecanismos e soluções identificados foram organizados na forma de ranqueamento para verificação dos mais mencionados nos estudos primários. Este ranking tem o objetivo de mostrar quais manuscritos possuem mais respostas sobre as questões de pesquisa. Isso não significa que os manuscritos com classificação mais baixa sejam piores que os primeiros colocados em qualidade. Significa apenas que os manuscritos com melhor classificação possuem mais informações para responder as questões de pesquisa. Posteriormente, os itens mais presentes nos estudos foram submetidos a uma análise individual para melhor entendimento sobre os seus conceitos básicos. Por fim, foi verificado quais mecanismos em específico lidam com os desafios encontrados e quais tecnologias *open source* poderiam ser implementadas.

3.7 Resultados da RSL

Esta seção apresenta os resultados da execução da revisão sistemática de literatura. A Figura 3.1 apresenta o processo do protocolo proposto para executar a RSL, com as respectivas quantidades dos estudos identificados em cada etapa do protocolo. Na busca automática realizada nas bases digitais usando a *string* de busca supramecionada, foram encontrados 22 artigos. Esses estudos foram submetidos ao processo de *snowballing*, resultado em 13 novos artigos selecionados. Dos 22 artigos encontrados inicialmente, 11 foram eliminados devido aos critérios de exclusão (5 estudos não abordavam o objeto da pesquisa e 6 eram duplicados). Desta forma, 11 estudos primários foram selecionados nas bases digitais e 13 estudos na execução do *snowballing*, totalizando 24 estudos primários. Os estudos primários selecionados são apresentados na Tabela 3.1. Os filtros aplicados durante a RSL baseados nos critérios de inclusão e exclusão estão demonstrados na Figura 3.2.

3.7.1 Avaliação da Qualidade das Revisões Realizadas

De acordo com os critérios de qualidade, os estudos selecionados foram devidamente analisados e pontuados, conforme apresentado na Tabela 3.2. Todos os estudos primários mencionaram desafios envolvendo autorização e autenticação em microsserviços (AQ3), além de mecanismos ou estratégias para mitigar tais problemas (AQ4), mesmo que de forma parcial. Todavia, há uma quantidade menor de trabalhos (15) que mencionem tecnologias *open source* que implementam o mecanismo (AQ5). Em geral, os estudos estão claros quanto ao objetivo (AQ1), mas 11 deles não descrevem as suas limitações (AQ2).

Tabela 3.1: Estudos Selecionados

ID	Ano	Título	Ref.
S1	**2021**	Security in microservice-based systems: A Multivocal literature review	[46]
S2	**2021**	Security in microservices architectures	[45]
S3	**2020**	Authentication and authorization in microservice-based systems: survey of architecture patterns	[47]
S4	**2020**	Information system development for restricting access to software tool built on microservice architecture	[55]
S5	**2020**	Research on Unified Authentication and Authorization in Microservice Architecture	[56]
S6	**2020**	Secure Edge Computing Management Based on Independent Microservices Providers for Gateway-Centric IoT Networks	[57]
S7	**2019**	Applying Spring Security Framework and OAuth2 To Protect Microservice Architecture API	[58]
S8	**2019**	A survey on security issues in services communication of Microservices-enabled fog applications	[59]
S9	**2019**	Enhancing security to the MicroService (MS) architecture by implementing Authentication and Authorization (AA) service using Docker and Kubernetes	[60]
S10	**2019**	Implementing secure applications in smart city clouds using microservices	[61]
S11	**2019**	Microservice Security Agent Based On API Gateway in Edge Computing	[62]
S12	**2019**	Securing Microservices	[63]
S13	**2019**	Security Mechanisms Used in Microservices-Based Systems: A Systematic Mapping	[64]
S14	**2018**	Authentication and authorization orchestrator for microservice-based software architectures	[65]
S15	**2018**	Defense-in-depth and Role Authentication for Microservice Systems	[66]
S16	**2018**	Fine-Grained Access Control for Microservices	[67]
S17	**2018**	Overcoming Security Challenges in Microservice Architectures	[9]
S18	**2018**	Security considerations for microservice architectures	[68]
S19	**2018**	Unified account management for high performance computing as a service with microservice architecture	[69]
S20	**2017**	A Secure Microservice Framework for IoT	[70]
S21	**2017**	Access control with delegated authorization policy evaluation for data-driven microserviceworkflows	[71]
S22	**2017**	Authentication and Authorization of End User in Microservice Architecture	[72]
S23	**2017**	Integrating Continuous Security Assessments in Microservices and Cloud Native Applications	[73]
S24	**2015**	Security-as-a-Service for Microservices-Based Cloud Applications	[48]

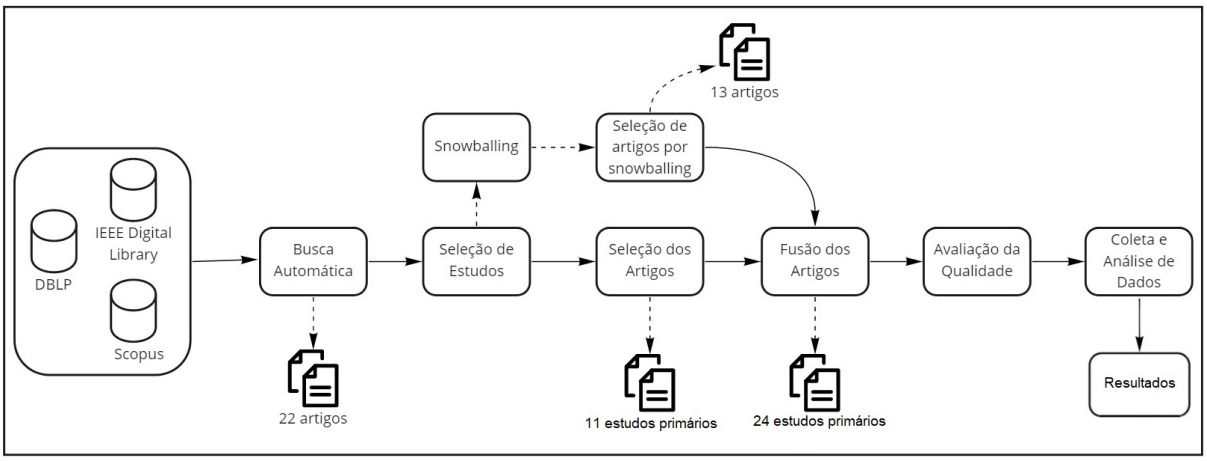


Figura 3.1: Processo de aplicação do protocolo

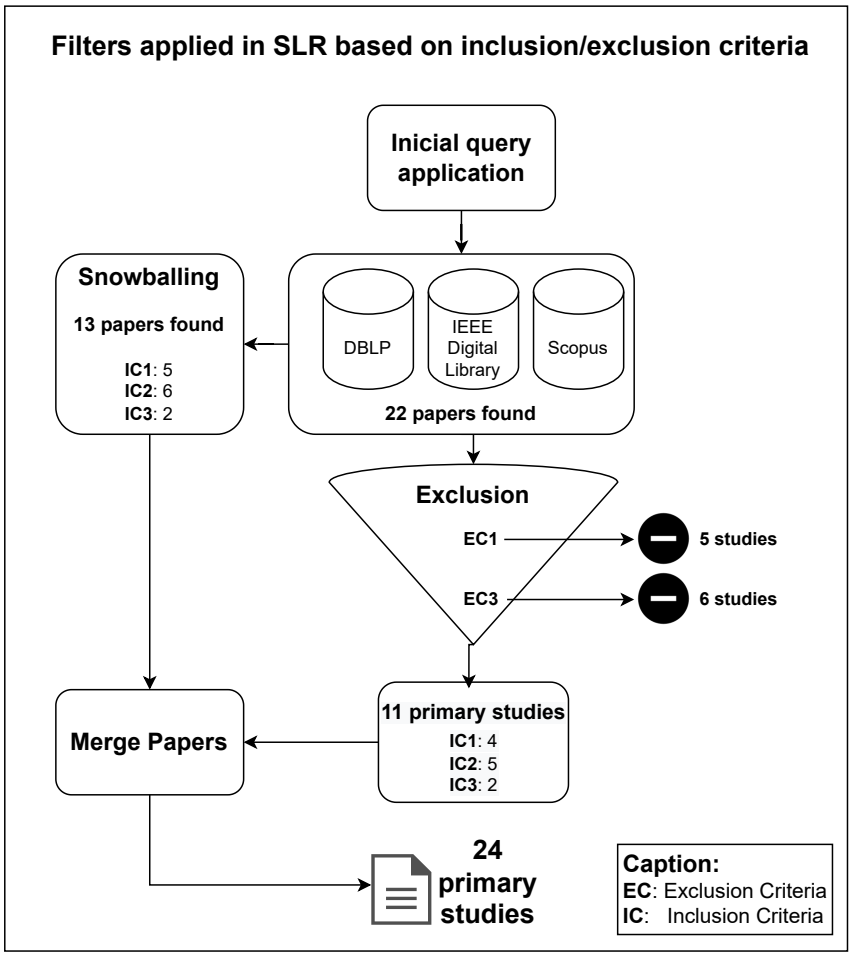


Figura 3.2: Filtros aplicados no processo da RSL.

Tabela 3.2: Ranqueamento de pontuação de acordo com as Avaliações de Qualidade

ID	AQ1	AQ2	AQ3	AQ4	AQ5	Total
S1	1	1	1	1	1	5.0
S8	1	0.5	1	1	1	4.5
S16	1	1	1	1	0.5	4.5
S23	1	1	1	0.5	1	4.5
S17	1	1	1	1	0.5	4.5
S21	1	0.5	0.5	1	1	4.0
S5	1	0	1	1	1	4.0
S6	1	0.5	0.5	1	1	4.0
S13	1	1	1	1	0	4.0
S7	1	0.5	0.5	0.5	1	3.5
S3	1	0	1	1	0.5	3.5
S15	0.5	0	1	1	1	3.5
S10	1	0.5	1	1	0	3.5
S11	1	0	0.5	1	1	3.5
S20	1	1	0.5	0.5	0	3.0
S14	1	0	1	1	0	3.0
S12	1	0	1	1	0	3.0
S2	1	0	1	1	0	3.0
S19	1	0.5	0.5	0.5	0.5	3.0
S4	0.5	0	1	0.5	0.5	2.5
S24	0.5	0.5	1	0.5	0	2.5
S22	0.5	0	0.5	0.5	0.5	2.0
S18	0.5	0	0.5	0.5	0	1.5
S9	0	0	0.5	0.5	0	1.0

3.7.2 Fatores de Qualidade

Foi realizada uma checagem para entender se existe algum tipo de relação entre a pontuação de qualidade e o ano em que o estudo foi publicado. Embora seja possível verificar que a média do *score* aumentou ao longo dos anos, o desvio padrão e o coeficiente de variação mostram que os dados estão heterogêneos, não sendo possível concluir que a qualidade tenha aumentado de fato ao longo do período, conforme demonstrado na Tabela 3.3. É possível verificar nesta situação que o desvio padrão aumenta na mesma proporção da média, além do coeficiente de variação se encontrar em grau elevado. Foram realizadas análises nos dados extraídos dos estudos selecionados com o objetivo de responder as questões de pesquisa.

Tabela 3.3: Média da pontuação de qualidade dos estudos por ano

	2015	**2017**	**2018**	**2019**	**2020**	**2021**
Número de Estudos	1	4	6	7	4	2
Média da Avaliação	2,5	3,38	3,33	3,29	3,50	4,00
Desvio Padrão	0	1,1087	1,1255	1,1127	0,7071	1,4142
Coeficiente de variação	0	0,3285	0,3376	0,3386	0,2020	0,3536

3.7.3 RQ.1.

Os desafios identificados em relação a autenticação e autorização do contexto dos sistemas de arquiteturas de microsserviços são expostos na Tabela 3.4. Tais desafios foram apresentados de acordo com o número de menções nos estudos selecionados, portanto, não significa que estes sejam os mais críticos em termos de vulnerabilidades ou a proporção que elas ocorrem em um ambiente real de microsserviços. O número de menções aos desafios encontrados nos estudos não reflete necessariamente um nível de prioridade em que devem ser observados em um ambiente prático. Dentre os desafios identificados, os cinco mais mencionados na literatura foram: "Comunicação entre microsserviços" (13 menções), "Confiança entre microsserviços comprometida por acesso não autorizado" (12 menções), "Preocupação individual para cada microsserviço" (12 menções), "Aumento na superfície de ataques" (12 menções), e "Controle de acesso ao microsserviço" (10 menções).

No geral, os trabalhos que mencionaram os desafios existentes, realizaram comparações da arquitetura de um sistema monolítico com o de microsserviços, explicando que no modelo monolítico, há apenas uma superfície para ser protegida. Todavia, no ambiente de microsserviço, cada serviço autônomo deve ser um ponto de preocupação em relação à segurança. Isso faz com que haja uma maior complexidade para manter todo este ecossistema devidamente protegido. Embora cada serviço precise de atenção especial, Yarygina e Bagge [9] alertaram que o provisionamento manual de segurança de centenas ou milhares de instâncias de serviços é inviável. Pereira-Vale et al. [46] comparam aplicação monolítica com microsserviço utilizando uma métrica KLOC (kilo Lines of Code). Eles também mencionam que em um aplicativo monolítico, a cada 100 kloc, haverá uma média de 39 vulnerabilidades. Com a mesma quantidade de linhas de código em uma aplicação de microsserviço, haverá uma média de 180 vulnerabilidades. Eles alertam que na decomposição do monolítico em microsserviços a segurança precisa ser uma propriedade global, não a soma das defesas de segurança locais.

Nehme et al. [63] discutem a importância da autenticação e autorização no contexto da segurança de microsserviços. Eles mencionaram que "Microsserviços só devem ser invocados após solicitar autenticação e, idealmente, com solicitação de autorização caso os níveis de privilégios estejam disponíveis.". Pereira-Vale et al. [64] realizaram um mapeamento

sistemático sobre mecanismos de segurança utilizados em microsserviços e descobriram que os mecanismos de segurança mais relatados estão relacionados a autorização, autenticação e credenciais. Banat et al. [65] também concordam que autenticação e autorização precisam ser observadas cuidadosamente em uma arquitetura de microsserviços, pois este cenário apresenta muitos pontos de acesso para usuários e demais partes da aplicação. Eles argumentaram que sendo os dados especialmente sensíveis, o ponto crucial do desenvolvimento é a autenticação e o processo de autorização. Cao et al. [69] propõem a implementação de um mecanismo global de autenticação e autorização chamado *Unified Account Management as a Service* (UAMS). Nesta implementação, eles usaram uma API RESTful dividida em vários microsserviços. Todos os dados confidenciais são transferidos criptografados pelo protocolo HTTPS.

Os estudos também destacaram que os microsserviços possuem característica de se comunicarem entre si, geralmente através do protocolo HTTP, e isso é um ponto de atenção que difere da abordagem tradicional monolítica que deve ser devidamente analisado e observado no aspecto de segurança. Em relação à questão da comunicação os autores mencionaram a implementação do *Transport Layer Security* (TLS), usado para proteger os canais de comunicação [66].

Os desafios apresentados, em geral, se complementam ou agem de maneira transversal. Mateus-Coelho et al.[45], afirmaram que "os microsserviços são frequentemente projetados para confiar em seus pares e, caso um deles seja comprometido e acessado indevidamente, é possível que haja uma grande vantagem para que todos os outros sejam explorados". Dongjin et al. [59] concordam e afirmam que "quando um único serviço é controlado por um invasor, pode influenciar maliciosamente outros serviços". Nehme et al. [67] mencionam um problema de controle de acesso que pode ser encontrado na arquitetura de microsserviços chamado "confused deputy attack". Em suas palavras, é um ataque de escalada de privilégios no qual um microsserviço, que é dado como confiável por outros microsserviços, torna-se comprometido. Sun et al. [48] trouxeram a preocupação com a confiança entre os serviços e afirmam que o "comprometimento de um único microsserviço pode derrubar toda a aplicação". Eles também relataram o desafio de monitorar e auditar a interação de microsserviços na rede e propuseram o projeto de uma infraestrutura de segurança como serviço para aplicações em nuvem baseadas em microsserviços, que ajude a monitorar a rede visando encontrar possíveis comportamentos não esperados em um cenário de comunicação entre eles.

Pereira-Vale et al.[46] afirmaram que a comunicação entre microsserviços são expostas através do ambiente de rede, o que cria uma potencial superfície de ataque. Os autores também mencionaram a problemática do aumento na superfície de ataques, pois a decomposição de uma aplicação em várias aumenta a superfície de ataque e a segurança da

aplicação se torna mais difícil de gerenciar, devido ao fato dela se tornar uma soma de várias defesas independentes, em vez de ser gerenciada de uma maneira global, como na abordagem monolítica.

Nguyen e Baker [58] alertaram que a comunicação via rede entre os microsserviços pode ocorrer no ambiente de internet, o que aumenta além da exposição, o número de possíveis atacantes. Kramer et al. [61] reforçam essa preocupação de implementar aplicações seguras em nuvens de cidades inteligentes usando microsserviços, pois neste cenário, lidam com uma enorme quantidade de dados, incluindo informações confidenciais sobre infraestrutura e cidadãos. Xu et al. [62] compartilham essa questão, utilizando microsserviço em dispositivos IoT. Lu et al. [70] também se preocupa com a segurança em dispositivos IoT, com arquitetura de microsserviços, principalmente por causa de dados confidenciais que podem ser compartilhados entre serviços. Eles incentivam o uso de gateways de API, que eliminarão todas as preocupações sobre microsserviços, pois todas as interações com os componentes serão realizadas com o gateway da API. Safaryan et al. [55] também se preocupam com a comunicação entre os diferentes microsserviços, pois ela é realizada através da interação em rede. É necessário proteger cada um dos serviços e a própria rede. Eles também alertam sobre a necessidade de um padrão a ser implementado. A falta de um padrão correto pode comprometer o ambiente de rede. Nguyen e Baker [58] concordaram sobre a necessidade de observar a comunicação entre os serviços. Banati et al. [65] argumentam que a rede usada na comunicação de microsserviços pode ser protegida com ferramentas de administração do sistema como VPN, Firewall e HTTPS.

Dongjin et al.[59] e Jander et al.[66], também apresentaram uma preocupação de que caso um simples serviço seja controlado por um atacante, ele poderia de forma maliciosa influenciar todos os outros serviços. Preocupações relacionadas com comunicação entre microsserviços, aumento de exposição da superfície, controle de acesso e cuidado individual em cada microsserviço, a estratégia do API Gateway, e uso de mecanismos como OAuth2 e JWT foram amplamente mencionados. Eles também pensaram na indústria, que não está totalmente ciente das questões de segurança envolvendo microsserviços.

O API Gateway ajuda a limitar a exposição entre microsserviços pois as requisições serão centralizadas nele, e não mais em um microsserviço diretamente [47]. Jin et al. [57] mencionam que o API Gateway protege um ambiente de microsserviços porque filtra todas as solicitações. Eles também propuseram um Edge Gateway para gerenciar microsserviços. Nehme et al. [63] não recomendam a validação do token de acesso no nível do gateway pois esta função precisa ser executada em um servidor de autenticação. Em contraste, Torkura et al. [73] propõem um gateway de segurança usado como controle de segurança para aplicar as políticas de segurança. Eles também alertaram sobre a descoberta, ou seja, um recurso de gateway que permite que um microsserviço se inscreva

nele. Se o serviço de descoberta puder aceitar qualquer assinatura, microsserviços vulneráveis poderiam ser enviados para ambientes de produção. O OAuth2 é um protocolo de autorização popular e poderia proteger o acesso aos microsserviços de um acesso não autorizado, já que são emitidos tokens de acesso para clientes confiáveis, que poderiam acessar determinados serviços [59]. Nguyen e Baker [58] explicaram que o OAuth 2.0 não é usado apenas em aplicativos baseados na web, mas pode ser aplicado em serviços de *back-end* sem necessidade de navegador da web ou interação do usuário. ShuLin and JiePing [56] mencionaram que o JWT é um padrão aberto (RFC 7519) que define uma maneira compacta e independente para transmitir informações com segurança entre as partes como um objeto JSON. Essas informações podem ser verificadas e são confiáveis porque estão assinadas digitalmente.

Barabanov e Makrushin[47] alertaram que a implementação de autorização diretamente no código fonte de cada microsserviço pode gerar problemas futuros, sobretudo em times diferentes trabalhando em microsserviços independentes, já que qualquer nova atualização de segurança deve ser realizada em todos os projetos, de maneira individual. He e Yang [72] seguiram na mesma linha e alertam que imitar a forma da estrutura monolítica em cada microsserviço tem várias deficiências, principalmente quando um novo serviço se junta ao sistema, onde será necessário implementar a função de segurança novamente. Eles propuseram uma solução criando um serviço específico focado em autenticação e autorização, como resultado, cada serviço é focado em seu próprio negócio, o que garante melhor escalabilidade e desacoplamento do sistema.

Jander et al. [66] mencionaram que diferentes times podem implementar sua própria abordagem de segurança interna no microsserviço que possuem responsabilidade, mas isso deve requerer conhecimento mais especializado das equipes. Torkura et. al[73] trouxeram a preocupação de que o desenvolvimento por times diferentes pode trazer microsserviços com padrões distintos em desenvolvimento, somado com o uso de tecnologias diversas, as quais devem implementar os mesmos padrões de segurança para que os microsserviços estejam devidamente alinhados.

Lu et al.[70] afirmaram que é completamente possível o desenvolvimento de microsserviços por diferentes times e mesmo por diferentes empresas, desde que haja um alinhamento sobre as implementações de segurança em cada microsserviço. Por fim, é importante mencionar ainda que, a falta de *patterns* de segurança em microsserviços, somado com os poucos estudos a respeito, tanto teórico como de ordem prática, pode influenciar no gerenciamento do desenvolvimento da arquitetura. Torkura et al.[73] alertaram que são encontradas diversas literaturas que destacam problemas de segurança em arquiteturas de microsserviços, todavia, nenhum deles oferece soluções práticas para lidar com essas situações.

Pereira-Vale et al. [46] alertaram sobre o emprego de um serviço de autenticação e autorização a ser utilizado em uma arquitetura de microsserviços. Tal servidor deve ser robusto o suficiente para autenticar o usuário e realizar as validações de token que são feitas a cada solicitação do cliente. A falta de preocupação com esses desafios pode causar um único ponto de falha (Single Point of Failure - SPOF), ou seja, se esta parte do sistema falhar, afetará toda a aplicação [74]. ShuLin e JiePing[56] afirmaram que o servidor de autenticação pode afetar o desempenho de todo o sistema, principalmente se houver várias requisições a ele.

Preuveneers e Joosen [71] alertaram sobre o fluxo de dados entre microsserviços. Mesmo que cada microsserviço esteja protegido individualmente, é importante observar se o fluxo de trabalho está válido. Nesse estudo, eles apresentaram uma estrutura orientada a fluxo de trabalho para evitar a comunicação não esperada entre microsserviços.

Mateus-Coelho et al.[45] enumeraram alguns exemplos de mecanismos que devem ser observados durante o desenvolvimento de uma arquitetura de microsserviços: senhas complexas, autenticação, falhas de segurança web (quais são as falhas mais observadas), pessoas e processos. Eles também listaram os riscos de segurança de aplicativos da web mais críticos: injeção, *broken authentication* e gerenciamento de sessão, *cross-site scripting*, *broken access control*, configuração incorreta de segurança, exposição de dados confidenciais, proteção insuficiente a contra ataques, falsificação de solicitações entre sites, componentes com vulnerabilidades conhecidas e sob APIs protegidas. Todos eles podem ser explorados em um ambiente de microsserviço. Nguyen e Baker [58] também apontaram riscos de segurança na web e realizaram experimentos usando ataque CSRF, ataque XSS e ataque de força bruta em um *endpoint* de API protegido por OAuth 2.0. Neste caso, todos os testes foram impedidos pela configuração proposta na Prova de Conceito apresentada no estudo.

3.7.4 RQ.2.

Os mecanismos identificados na literatura utilizados para lidar com os desafios de autenticação e autorização em arquitetura de microsserviços são apresentados na Tabela 3.5. O Protocolo OAuth 2.0 foi o mais mencionado (16 menções), seguido de JWT (14 menções), API Gateway (14 menções), OpenID Connect (9 menções) e Single Sign-ON (8 menções). A Figura 3.3 apresenta o quantitativo de ocorrência dos mecanismos utilizados nos estudos selecionados. Alguns mecanismos estão presentes em apenas um estudo. É importante salientar que, em linhas gerais, os mecanismos identificados não precisam ser implementados de forma única, ou seja, eles podem coexistir no mesmo ambiente, cada qual com uma finalidade específica. Identificamos nos estudos selecionados algumas implementações em que os mecanismos atuam em conjunto [46, 45, 47, 57, 63, 65, 67, 9, 71], para mitigar

possíveis vulnerabilidades envolvendo autenticação e autorização no ambiente de microsserviços. É importante observar que alguns estudos apenas apontam os mecanismos, sem explicar profundamente ou demonstrar uma implementação prática, como o trabalho de Pereira-Vale et al. [64], que está mais focado em realizar um mapeamento sistemático dos mecanismos de segurança, sem explorá-los profundamente.

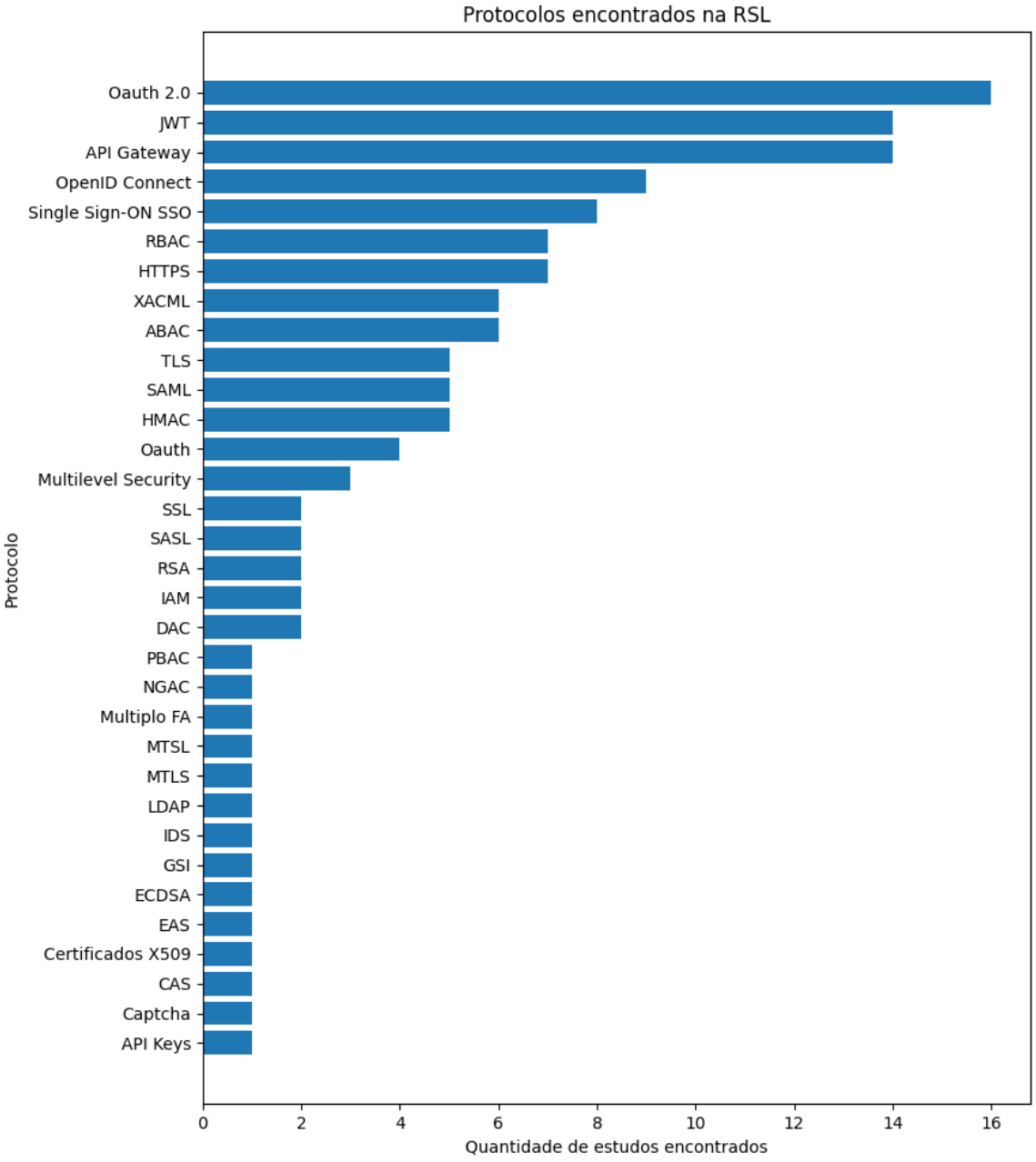


Figura 3.3: Quantitativo dos mecanismos encontrados

Nas próximas subseções, apresentamos uma breve descrição dos 5 mecanismos mais mencionados/utilizados nos estudos selecionados.

OAuth2 (Open Authorization)

O protocolo OAuth2 foi definido pela RFC 6749 [75]. De acordo com Bánáti et al.[65] o OAuth2 é uma estrutura de autorização que permite aos usuários acessarem diferentes serviços sem a necessidade de compartilhar suas credenciais. Em um cenário prático, o usuário se autentica e recebe um código de autorização ou um token de acesso, que podem ser utilizados para acessar recursos, sem a necessidade de contatar novamente o servidor de autorização ou mesmo ter que informar novamente o usuário e senha [59]. Os tokens de acesso são validados em cada solicitação para algum serviço [62, 71]. Esse procedimento gera um risco por afetar a performance de uma arquitetura distribuída, pois, se houver muitas requisições, o servidor de autorização poderá ser afetado em termos de performance [56].

O OAuth2 é um dos protocolos mais utilizados pelas arquiteturas de microsserviços para delegação de acessos [59, 67] e pode ser empregado tanto em aplicações web como em serviços de *back-end*, além de atender tanto a proposta de autenticação como autorização [58, 64]. É importante mencionar que o OAuth2 é amplamente utilizado como protocolo de autorização para proteger serviços que utilizam o estilo REST (*Representational State Tranfer*) [57, 59, 63], além de adotar o protocolo HTTPS na comunicação de dados [59].

JWT (JSON Web Token)

O JWT foi definido pela RFC 7519 [5]. É um padrão aberto que traz uma forma compacta e independente de transmitir informações com segurança entre as partes utilizando um objeto JSON (*Javascript Object Notation*). Essas informações podem ser verificadas e são confiáveis, pois são assinadas digitalmente com a utilização de um segredo [56]. Ele possui um formato dividido em três partes: header, payload e assinatura, com o *header* separado em duas partes, tipo de token e o algoritmo que está sendo usado, como HMAC, SHA256 ou RSA [65].

O JWT possui vantagem em relação aos tokens tradicionais pois a sua verificação pode ser feita diretamente no servidor de recurso, sem se conectar com o servidor de autenticação [56, 72]. Com JWT é possível recuperar informações do usuário diretamente do token [56, 60]. Além de informações do usuário, é comum encontrar em um JWT as suas permissões e tempo de validade do token [72]. O JWT possui aderência em aplicações do tipo *stateless*, ou seja, aquelas que não guardam sessão no lado do servidor, pois eles não são armazenados no *back-end*, ficando com o cliente que o possui e devendo ser utilizado em cada requisição do cliente ao recurso [60]. Em um ambiente de microsserviços, os JWT podem ser trafegados durante a comunicação entre eles [71]. Por fim, vale salientar que o JWT consegue se integrar com o protocolo OAuth 2.0 [57, 46].

API Gateway

No ambiente de microsserviços, O API Gateway atua como um intermediário entre o cliente e os microsserviços. Fornece um ambiente de rede privada que permite a troca de dados privados [45, 62], ou seja, os clientes não se comunicam diretamente com os serviços, mas apenas com um único gateway, que por sua vez se comunica com o serviço requisitado. Ele pode ser uma entrada que realiza o filtro das solicitações do cliente e faz o devido encaminhamento para o microsserviço [57]. Checa as credenciais do usuário, para conferir se ele possui a devida autorização [73, 9].

Percebemos que o API gateway é uma técnica para diminuir a exposição dos microsserviços. No entanto, é importante em um trabalho futuro comparar a comunicação em diferentes cenários. Esses cenários podem estar usando ou não o gateway de API entre o cliente e os microsserviços. Consequentemente, será possível coletar os pontos fortes e fracos de ambas as abordagens e verificar ainda a possibilidade de cenários híbridos.

Lu et al.[70], afirmaram que o API Gateway pode agregar múltiplos microsserviços em uma única interface de cliente, sendo um elemento que fica entre o cliente e o serviço requisitado. O uso do API Gateway ajuda a diminuir a exposição dos sistemas, pois os microsserviços ficam todos protegidos por trás do API Gateway [47]. Embora tenham sido apontadas diversas vantagens para a sua implementação, o seu uso pode não se mostrar vantajoso quando ele se torna um ponto de decisão único, pois, se ocorrer falha neste elemento, toda a aplicação pode se tornar inacessível [47]. Um API Gateway pode utilizar serviços como o JWT e OAuth2 [47, 57, 59, 62, 9, 70, 72].

OpenID Connect

O OpenID é um padrão aberto de autenticação que garante os usuários possuírem apenas uma identidade digital para várias aplicações ou serviços [65]. Dongjin et al. [59] afirmaram que é uma camada de autenticação sobre o protocolo OAuth, que permite que os serviços leiam as informações básicas do usuário. Nehme et al. [63, 67] afirmaram que o OpenID é construído no topo do protocolo OAuth. Yarygina e Bage [9] reforçaram que o OpenID provisiona a identidade do usuário. O OpenID pode ser utilizado em conjunto com o OAuth2[46, 47, 71].

Há diferença entre OpenID e OpenID Connect (OIDC). De acordo com o site da OpenID Foundation, “OpenID Connect executa muitas das mesmas tarefas que o OpenID 2.0, mas o faz de uma maneira amigável via API, passível de uso por aplicativos nativos e móveis” [76]. Eles também explicam que “OpenID Connect define mecanismos opcionais para assinatura e criptografia robustas. Enquanto a integração do OAuth 1.0a e do

OpenID 2.0 exigia uma extensão, no OpenID Connect, os recursos do OAuth 2.0 são integrados ao próprio protocolo" [76].

SSO (Single Sign-On)

O SSO permite que um usuário seja autenticado somente uma vez quando se loga em um sistema em particular, para que possa acessar todos os recursos e serviços daquele sistema sem precisar de outra autenticação [65, 72]. Segundo Banati et al. [65], a principal proposta deste mecanismo é a troca das credenciais de autorização e não a autenticação por si só. Os autores ainda reforçaram que o mecanismo garante uma autenticação unificada em microsserviços e a implementação deste recurso pode melhorar a experiência de usuário [69]. Da mesma forma que o API Gateway, a implementação de um servidor SSO pode causar um *Single Point of Failure*, ou seja, caso existam problemas nesse sistema, toda aplicação pode ficar comprometida, pois centraliza toda autenticação de um sistema [72]. É possível a implementação de um sistema Single Sign-On baseado em OAuth2[61, 65, 71].

HTTPS

O HTTPS (*Hyper Text Transfer Protocol Secure*), é definido pela RFC 2818 [77]. Ele descreve o uso de HTTP sobre TLS (*Transport Layer Security*). O uso do protocolo HTTPS garante que a comunicação seja criptografada [61]. Ele fornece um canal entre dois hosts identificados por certificados [66]. O uso de HTTPS não se limita apenas a criptografar dados, mas garante que determinado cliente esteja se comunicando com quem ele deseja [45].

RBAC

O RBAC (*Role-Based Access Control*), é usado no processo de autorização [46]. É um modelo de controle de acesso baseado em identidade [71]. O uso de um controle de acesso baseado em função aumenta a flexibilidade do sistema, porque a função definirá qual acesso é permitido ao cliente [56]. O RBAC é um modelo de controle de acesso centrado no usuário e não leva em conta o relacionamento entre a entidade solicitante e o recurso [9]. As funções de autorização RBAC podem ser incorporadas em tokens JWT como um atributo adicional [9].

ABAC

O ABAC (Attribute-Based Access Control), segundo Preuveneers e Joosen [71], "concede direitos de acesso a assuntos através do uso de políticas ou regras que combinam vários tipos de atributos para facilitar o acesso do usuário aos corretos recursos sob as corretas

condições". Complementaram que oferece mais expressividade e flexibilidade em relação a outros modelos de controle de acesso como o RBAC. O objetivo principal do ABAC de acordo com Yu et al. [59] é "um modelo de controle de acesso para atender aos requisitos de ambientes altamente heterogêneos, como ambiente *multi-cloud*". Eles também apontaram o benefício do gerenciamento e orquestração de segurança centralizada que protegerá o aplicativo de acordo com políticas consistentes. O ABAC é recomendado para ser usado quando há recursos de de autorização refinada, como acesso a uma chamada de API específica [9].

XACML

O XACML (eXtensible Access Control Markup Language) é definido pela RFC 7061 [78]. De acordo com este documento, XACML "define uma arquitetura e uma linguagem para controle de acesso (autorização). A linguagem consiste em solicitações, respostas e políticas". Ele é usado para criar políticas de controle de acesso e pode ser aplicado com o protocolo OAuth 2.0 [67]. Nehme et al. [67] propõem um modelo com XACML junto com OAuth 2.0. Nesse caso, o OAuth 2.0 atua como um serviço de autorização e o XACML com administração de políticas e pontos de decisão. Barabanov e Marushin [47] desencorajam o uso de XAML porque ele tem uma sintaxe complicada, causando mais trabalho para os desenvolvedores, além de não haver muitas integrações de código aberto.

HMAC

O HMAC (*Hash-based Message Authentication Code*) é definido pela RFC 2104. Ele fornece uma maneira de verificar a integridade de uma informação transmitida por um meio [79]. Nas palavras de Mateus-Coelho et al., o HMAC consiste em um "código de mensagens baseado em hash para assinar o pedido". Segundo os mesmos autores, existem muitos exemplos que podem ser encontrados na internet sugerindo o uso de HMAC sobre HTTP. O algoritmo HMAC também pode ser usado para assinar um JWT [56].

SAML

O SAML (*Security Assertion Markup Language*) 2.0 é definido pela RFC 7522 e é definido como uma estrutura baseada em XML que permite que informações de identidade e segurança sejam compartilhadas entre domínios de segurança [80]. Em um ambiente de microsserviços, SAML é usado para trocar atributos de usuário armazenados em um provedor de identidade [71]. Mateus-Coelho et al. [45] afirmou que "SAML e OpenID são perfeitos para autenticação e autorização do usuário físico em um sistema, mas também

são ótimos para autenticação serviço a serviço". No entanto, eles admitiram que o SAML é complexo quando comparado a outras tecnologias, como Api Keys.

3.7.5 RQ.3.

As principais soluções/tecnologias *open source* que implementam os mecanismos e/ou estratégias de autenticação e autorização identificados na literatura são apresentadas na Tabela 3.6. É possível notar que as bibliotecas do ecossistema Spring [81] são as mais mencionadas (Spring Security, Spring Cloud, Spring Boot, Gateway Zuul e Eureka Server), totalizando 10 menções. Percebemos que Spring Boot e Eureka não são focados especificamente em segurança, mas possuem bibliotecas de segurança que podem ser utilizadas em conjunto. Por exemplo, o uso do Spring Boot permite implementar a biblioteca Spring Security e o Eureka ajuda a implementar um API Gateway, utilizando o Spring Cloud. Alguns dos estudos selecionados demonstram uma implementação prática do framework Spring utilizando mecanismos de segurança [55, 56, 58, 59].

Embora a pesquisa tenha encontrado várias referências ao ecossistema Spring, que é construído pela linguagem de programação Java, é importante mencionar que existem *frameworks* alternativos baseados em outras linguagens que implementam os mecanismos de segurança encontrados em um ambiente de microsserviços, como o GoKit (Golang) [82], Flask (Python) [83] e .NET Core (C#) [84].

A outra solução *open source* citada mais de uma vez é denominada Kong [85] (2 menções), sendo as demais mencionadas apenas uma única vez. É importante lembrar que o Spring Framework utiliza a linguagem de programação Java e possui diversas bibliotecas para implementação de estratégias ou mecanismos de segurança em microsserviços, como API Gateway, OAuth2 e OpenID Connect [81]. A aplicação Kong se faz referência ao "Kong API Gateway", ou seja, dentre todos os mecanismos e estratégias levantadas, este suporta a implementação de um API Gateway.

3.8 Discussões

Dados os problemas, mecanismos e soluções *open source* apresentadas, foram verificados quais deles poderiam ser implementados para enfrentar os desafios, de acordo com o estudo dos trabalhos relacionados nesta RSL. A Tabela 3.7 apresenta as soluções que atuam nos problemas levantados. Foi verificado que parte dos problemas não possuem uma vinculação direta sobre o mecanismo e/ou implementação *open source*. Dos 20 (vinte) problemas levantados na Tabela 3.4, foram encontrados mecanismos a serem aplicados em 09 (nove). Embora pareça um número baixo, estes problemas em específico são os mais mencionados pelos autores.

Da mesma forma em que os mecanismos foram mencionados em maior quantidade pelos autores, estes se encontram em maior quantidade para enfrentar esses problemas em específico, tendo destaque novamente a implementação do Oauth2.0, JWT, API Gateway, OpenID Connect e Single Sign ON (SSO). No entanto, isso não significa que eles sejam melhores mecanismos ou irão resolver qualquer tipo de problema de segurança relacionado à arquitetura de microsserviços. Mesmo os mecanismos menos mencionados podem ser mais apropriados, dependendo do caso. É importante saber o que é cada mecanismo individualmente e o que ele faz, para então implementar uma boa arquitetura de segurança em um sistema.

3.8.1 Limitações da RSL

O estudo foi operacionalizado com buscas nas bases DBLP, IEEE e Scopus. Para evitar que trabalhos relevantes fossem descartados, aplicou-se o processo de snowballing. Mesmo com este cuidado, é possível que, aumentando o número das bases para consultas, novos estudos sejam encontrados. Todavia, conforme *verificado* nos em alguns trabalhos coletados, há atualmente uma carência de estudos a respeito do assunto [46, 45, 55, 59, 61, 64, 9].

Também foi verificado que faltam estudos demonstrando a implementação prática de segurança em microsserviços [58, 64, 73]. Desta forma, é provável que ao longo dos próximos anos, caso aumentem as pesquisas relacionadas com o assunto em questão, se fará necessária uma nova revisão sistemática, visando complementar os conhecimentos coletados neste trabalho. Não podemos concluir que os mecanismos menos citados nos estudos sejam menos utilizados, portanto, é importante explorar todos eles, o que pode ser feito em um trabalho futuro.

Por fim, é importante destacar que este estudo está mais voltado para a identificação de respostas às questões de pesquisa, ou seja, é possível que as respostas a essas questões se tornem objeto de novos estudos que indiquem quais desafios são mais críticos em termos de vulnerabilidade, o quanto eles ocorrem em um ambiente prático, ou mesmo quais desses desafios devem ser abordados com maior prioridade. Os mecanismos também podem ser implementados e testados para descobrir em um ambiente prático quais dos desafios são mitigados com o mecanismo implementado.

3.8.2 Considerações Finais da RSL

Conforme verificado ao longo da execução deste trabalho e demonstrado na Tabela 3.4, há uma carência de estudos relacionados com segurança em arquitetura de microsserviços. A escassez aumenta quando há um direcionamento específico para autenticação e autorização, sobretudo em uma abordagem prática. É importante que o assunto seja melhor

explorado, pois, dentro de um ambiente de microsserviço, é necessária uma preocupação em aspectos de segurança em cada serviço, de maneira individual, pois a adoção dessa arquitetura pode aumentar a superfície de ataques e ainda gerar pontos de atenção na comunicação entre eles.

De todos os pontos elencados na Tabela 3.4, existem questões relacionadas à implementação de tecnologias propriamente ditas, todavia, há outros aspectos que permeiam o assunto, como por exemplo a organização de times de desenvolvimento trabalhando em microsserviços diferentes dentro de um mesmo sistema, portanto, é uma temática com vasto terreno para ser explorado.

Foram encontrados vários mecanismos e estratégias que mitigam os principais pontos de atenção observados. Todos estão listados na Tabela 3.5, e o OAuth 2.0 é o mais mencionado, juntamente com o Json Web Token (JWT) e o uso de API Gateway. A correta implementação destes pode diminuir a possibilidade de ocorrer qualquer tipo de acesso não autorizado em um ou mais microsserviços, porque o ambiente fica melhor protegido. Existem poucos estudos sobre implementações práticas, e desta forma, vislumbra-se um cenário para trabalhos futuros, sobretudo com propostas de *patterns* específicos dentro deste contexto.

Por fim, verificou-se que a literatura aponta poucas soluções *open source* que implementam os mecanismos e estratégias encontradas. Neste caso, uma alternativa viável é que a busca se expanda para novas fontes, inclusive literatura cinzenta, que é a literatura produzida em todos os níveis de governo, acadêmicos, empresariais e industriais, em formatos impressos e eletrônicos, mas que não é controlada por editoras comerciais, ou seja, onde a publicação não é atividade primária do corpo produtor [86]. Tais achados podem ser devidamente experimentados com o rigor científico e apontados como soluções técnicas que resolvam os desafios coletados ao longo deste trabalho.

Tabela 3.4: Desafios relacionados com autenticação e autorização em microsserviços

Pos	**Desafio**	**ID**	**Qtd Menções**
1º	Comunicação entre microsserviços	S1, S2, S3, S4, S7, S9, S10, S15, S16, S17, S18, S23, S24	13
2º	Confiança entre microsserviços comprometida por acesso não autorizado	S1, S2, S4, S6, S8, S12, S15, S16, S19, S21, S23, S24	12
3º	Preocupação individual para cada microsserviço	S1, S2, S5, S10, S12, S13, S15, S16, S21, S22, S23, S24	12
4º	Aumento na superfície de ataques (em comparação ao monolítico)	S1, S2, S3, S7, S8, S13, S14, S16, S17, S23	10
5º	Controle de acesso ao microsserviço	S5, S8, S10, S11, S14, S15, S19, S20, S21	9
6º	Autorização entre serviços	S1, S7, S8, S15, S16, S17, S18, S21	8
7º	Poucos estudos a respeito	S1, S2, S4, S8, S10, S13, S17	7
8º	Falta de patterns de segurança em microsserviços	S1, S13, S15, S17, S20, S21	6
9º	Times diferentes que trabalham em microsserviços diferentes devem ter o mesmo entendimento em segurança	S3, S15, S17, S20, S23	5
10º	Bypass no Api Gateway	S3, S4, S6, S12	4
11º	Detecção de intrusão/monitoramento	S1, S12, S24	3
12º	Escalada de privilégios	S2, S16, S24	3
13º	Falta de estudo demonstrando implementação prática de segurança em microsserviços	S7, S13, S23	3
14º	Coordenar servidor de autenticação com novos microsserviços	S1, S22	2
15º	Falta de atenção em reação/recuperação de ataques	S1, S13	2
16º	Validação do token a cada requisição a microsserviço	S5, S6	2
17º	Imagens Públicas podem estar comprometidas	S1	1
18º	Muitas aplicações em microsserviços comerciais sem possibilidade de avaliar código	S4	1
19º	Uso de servidor de autenticação/autorização que lide com todos os microsserviços	S3	1
20º	Possibilidade de desenvolvimento em várias tecnologias	S23	1

Tabela 3.5: mecanismos de segurança utilizados em arquitetura de microsserviços

Pos	Mecanismo	ID	Qtd menções
1º	OAuth 2.0	S1, S3, S5, S6,S7, S8, S10, S11, S12, S13, S14, S15, S16, S17, S20, S21	16
2º	JWT	S1, S3, S4, S5, S6, S9, S11, S12, S13, S14, S15, S17, S21, S22	14
3º	API Gateway	S2, S3, S4, S5, S6, S11, S12, S13, S14, S16, S17, S19, S20, S22	14
4º	Single Sign-ON SSO	S1, S2, S9, S10, S14, S19, S21, S22	8
5º	OpenID Connect	S1, S3, S8, S12, S16, S17, S21	7
6º	HTTPS	S2, S10, S14, S15, S17, S19, S20	7
7º	RBAC	S1, S3, S5, S13, S14, S17, S21	7
8º	ABAC	S1, S8, S14, S17, S20, S21	6
9º	XACML	S1, S3, S13, S15, S16, S21	6
10º	HMAC	S2, S3, S5, S14, S21	5
11º	SAML	S1, S2, S13, S14, S21	5
12º	TLS	S1, S10, S14, S15, S16	5
13º	OAuth	S1, S5, S8, S16	4
14º	Multilevel Security	S1, S3, S13	3
15º	DAC	S14, S21	2
16º	IAM	S14, S21	2
17º	RSA	S5, S11	2
18º	SASL	S1, S13	2
19º	SSL	S2, S13	2
20º	MTLS	S1, S17	2
21º	OpenID	S2, S14	2
22º	API Keys	S2	1
23º	Captcha	S19	1
24º	CAS	S8	1
25º	X509 Certificates	S1	1
26º	EAS	S3	1
27º	ECDSA	S5	1
28º	GSI	S8	1
29º	IDS	S12	1
30º	LDAP	S8	1
31º	MFA	S19	1
32º	NGAC	S3	1
33º	PBAC	S1	1

Tabela 3.6: Tecnologias *open source* que implementam segurança em arquitetura de microsserviços

Open source	**ID**
Spring Security	S1, S4, S5, S7, S8
Kong	S6, S11
Spring Boot	S6, S7
Gateway Zuul	S4, S5
Eureka Server	S5
Jadex	S15
Jarvis	S1
Lagom	S15
VertX	S15

Tabela 3.7: Vinculação dos Desafios, Mecanismos e Soluções open source

Desafio	**Mecanismo**	**Open Source**
Aumento na superfície de ataques (em comparação ao monolítico)	API Gateway (S2, S3, S4, S6, S12, S13, S16, S17, S20, S22), OAuth2 (S7, S12, S13), SSO (S14)	Spring (S4, S7), Kong (S6, S11)
Autorização entre serviços	OAuth2 (S1, S5, S6, S7, S8, S10, S12, S13, S14, S15, S16, S17, S21), SAML(S2), OpenID(S2, S14, S16, S17), JWT (S9, S12, S13, S14, S15, S17, S21)	Spring (S1, S5)
Bypass no Api Gateway	XACML (S3), NGAC (S3), JWT (S3, S6, S12), OpenID (S3, S12, S16), OAuth2 (S3, S6, S12, S16), TLS (S3), IDS (S12)	Spring (S4), Kong (S6)
Comunicação entre microsserviços	TLS (S1, S10, S14, S15), MTLS (S17), SSL (S2), HTTPS (S2, S10, S14, S15, S17, S20), SAML(S2, S14, S21), XACML (S16, S21), OpenID(S2, S3, S8, S16, S17), JWT (S4, S5, S6, S12, S13, S14,15, S16, S17, S21), OAuth2 (S5, S6, S7, S8, S10, S13, S15, S17, S21), GSI (S8)	Spring (s5), Kong (S6)
Confiança entre microsserviços comprometida por acesso não autorizado	JWT (S1, S3, S4, S5, S6, S9, S11, S12, S13, S14, S15, S17, S21, S22), OAuth 2.0 (S1, S3, S5, S6, S7, S8, S10, S11, S12, S13, S14, S15, S16, S17, S20, S21), OpenID Connect S1, S2, S3, S8, S12, S14, S16, S17, S21)	Spring (S4, S5)
Controle de Acesso ao microsserviço	OAuth2 (S1, S5, S8, S10, S11, S12, S13, S14, S15, S16, S20, S21), OpenID (S1, S2, S3, S8, S12, S14, S16), TLS(S1), MTLS(S1), SASL (S1), SSO (S1, S2), JWT (S1, S5, S11, S12, S13, S14, S15, S21), HMAC(S2, S21), ABAC (S8, S17, S20, S21), RBAC (S17, S21), CAS (S8), RSA (S11), XACML (S16), Captcha (S20), Multiplo FA (S20), DAC (S21), IAM (S21)	Spring (S1, S5, S11)
Coordenar servidor de autenticação com novos microsserviços	LDAP (S8), SSO (S1, S2, 10, S13, s14, S20, S21, S22), OAuth2 (S12, S14, S16), OpenID (S12, S14, S16),	
Preocupação individual para cada microsserviço	JWT (S1, S3, S4, S5, S6, S9, S11, S12, S13, S14, S15, S17, S21, S22), OAuth 2.0 (S1, S3, S5, S6, S7, S8, S10, S11, S12, S13, S14, S15, S16, S17, S20, S21), OpenID Connect (S1, S2, S3, S8, S12, S14, S16, S17, S21)	Spring (S4, S5)
Uso de servidor de autenticação/autorização que lide com todos os microsserviços	SSO (S1, S2, 10, S13, s14, S20, S21, S22), OAuth2 (S12, S14, S16)	

Capítulo 4

Percepção dos Profissionais de TIC

Este capítulo apresenta os detalhes da aplicação de um questionário baseado nos achados da Revisão Sistemática de Literatura, visando validar se as descobertas encontradas na literatura estão alinhadas com as práticas da indústria.

4.1 Planejamento

Nesta seção, descrevemos o procedimento utilizado para a definição das questões, seleção de participantes e de como o questionário foi conduzido.

4.1.1 Definição das Questões

As questões, no geral, foram baseadas nos achados da Revisão Sistemática de Literatura e no escopo desta pesquisa. Foram também utilizadas perguntas de nivelamento, para verificar o domínio do respondente em relação ao assunto em tela. As perguntas 1 a 6 se referem a questões de nivelamento. As demais enfocam assuntos específicos dos achados da RSL. Nas questões 7, 10 e 12 são apresentados respectivamente os desafios, protocolos e implementações *open source*. As perguntas são em geral, na forma de testes ou em escala ordinal, ainda com algumas questões abertas para complementar o que não se esgotou nas questões fechadas.

As questões definidas foram do tipo "transversal", em que participantes são questionados por informações partindo do período específico em que estão respondendo o questionário [13]. Todo o corpo do questionário está disponível no Apêndice A.

4.1.2 Definição do Processo

O questionário foi aplicado de maneira online pela plataforma Google Forms e o link foi disponibilizado para os participantes responderem quando acharem mais oportuno,

anonimamente, sem a presença do entrevistador e sem limite de tempo. O respondente ficou ciente que as informações são coletadas de modo anônimo e utilizadas na pesquisa.

4.1.3 Avaliação do Instrumento

Antes de ser enviado para o público geral, o questionário foi avaliado por 10 (dez) participantes selecionados de forma não-probabilística [13] em uma amostra escolhida por conveniência [13], pois neste caso, estavam acessíveis e disponíveis para colaboração apenas pessoas com grau de afinidade profissional e/ou acadêmica do aplicador da avaliação. Todos foram orientados para prestar feedback transparente sobre dificuldades e sugestões de melhoria. Foram apresentadas apenas observações sobre pontos gramaticais e disposição das ordens das questões, sem necessidade de mudanças estruturais.

4.1.4 Seleção de Participantes

Assim como a Avaliação do Questionário, a seleção dos participantes foi dada de forma não-probabilística. A seleção da amostra se deu tanto por conveniência como por "snowball" [13]. O questionário foi distribuído para colaboradores com algum tipo de relacionamento social, acadêmico ou profissional, que tivessem condições para responder o questionário e ainda pudesse encaminhar para outros participantes. O questionário foi divulgado em módulos de comunicação da internet, como Whatsapp, Telegram, Linkedin e fóruns de tecnologia, todos dentro do ciclo de relacionamento dos autores desse trabalho. Importante salientar que a divulgação se limitou em grupos/comunidades específicas na área de Tecnologia da Informação e Comunicação (TIC), reforçando que o participante tivesse experiência profissional com o objeto da pesquisa. Durante o processo, foram coletadas 51 (cinquenta e uma) respostas, de participantes cujo perfil se encontra na Tabela 4.1. Percebe-se nesta tabela que há um número elevado de participantes com nível superior, compreensão sobre o conceito de microsserviços e atuação profissional com essa arquitetura.

4.2 Resultados do Questionário

Nesta seção são analisados os resultados do questionário, visando validar se os achados da RSL estão em aderência com o praticado na indústria. Primeiramente, foram descartadas as respostas dos participantes que nunca trabalharam com arquitetura de microsserviços e se declararam com um nível baixo (1 ou 2) de compreensão sobre o conceito, totalizando 05 (cinco) exclusões. Restaram ainda 46 (quarenta e seis) respostas de pessoas que já

Tabela 4.1: Número e descrição dos participantes

Formação Acadêmica	
Ensino Médio	1
Graduação	19
Especialização	12
Mestrado	16
Doutorado	2

Tempo de Serviço com TIC (anos)	
<1	**1**
1 a 3	**4**
4 a 6	**7**
7 a 10	**12**
11 a 15	**12**
16 a 20	**5**
>21	**10**

Área de atuação da organização que participa/participou com desenvolvimento de software	
Administração Pública	24
Empresa Privada	23
Área Acadêmica	4

Função na Organização	
Engenheiro de Software	17
Programadores	14
Testador	1
Gerentes de Projetos	7
Arquiteto	1
Chefe QA	1
Gestores	3
Governança	1
Cientista de Dados	1
Estagiário	1
Auditor	1
Analistas de Sistemas	2
Analista de Requisitos	1

Trabalha ou já trabalhou com microsserviços	
Sim	45
Não	6

Nivel de compreensão sobre conceito de microsserviços (escala de 1 a 5)	
1	3
2	5
3	9
4	23
5	11

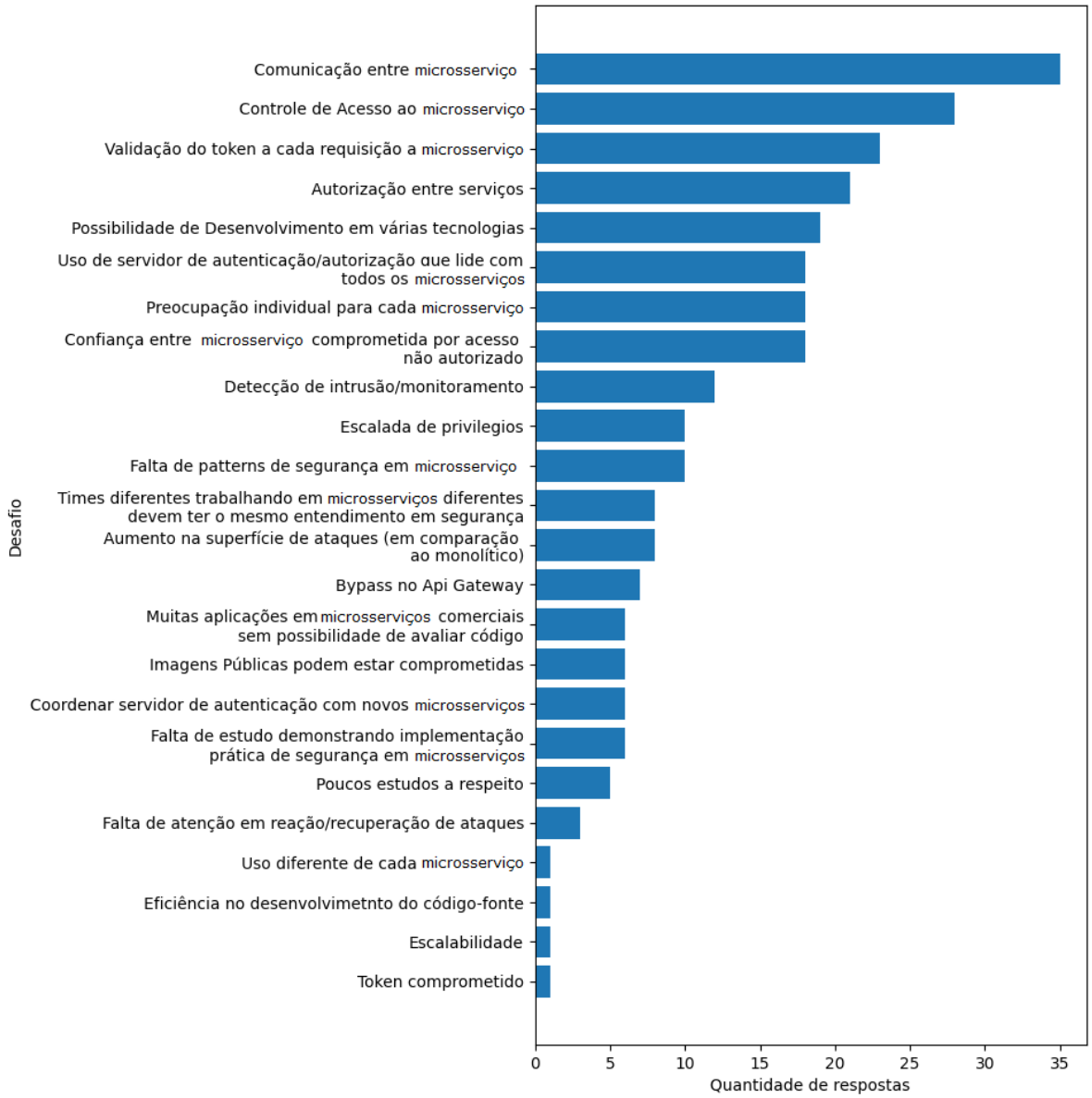
trabalharam com microsserviços e julgaram possuir um domínio médio ou elevado no assunto.

4.2.1 RQ1

Em relação aos desafios envolvendo autenticação e autorização em arquitetura de microsserviços, percebe-se que há consonância entre os achados da RSL e o questionário pois, em geral, as maiores preocupações são observadas em ambos os instrumentos. Um exemplo dessa afirmação é o desafio de comunicação entre microsserviços, que é o mais mencionado em ambos. Na RSL, de acordo com a Tabela 3.4, há 13 menções de trabalhos, e no questionário, conforme apresentado na Figura 4.1, foi observado por 35 participantes.

Outros achados da RSL com grande quantidade de menções, como confiança entre microsserviços comprometida por acesso não autorizado, preocupação individual para cada microsserviços e controle de acesso ao microsserviço, também são amplamente citados.

Figura 4.1: Desafios observados em arquitetura de microsserviços



Vale salientar que também houve elevado número de menções de desafios poucos observados na RSL, tal como Validação do Token a cada requisição (23 menções) e uso de servidor de autenticação/autorização que lide com todos os microsserviços (18 menções). Ambos os desafios possuem uma certa relação pois a utilização de um servidor de autenticação/autorização deve ser robusta o suficiente para autenticar o usuário e realizar as validações dos tokens que são feitas a cada requisição do cliente. A inobservância desses

desafios pode causar um ponto único de falha de falha (SPOF), ou seja, uma parte de um sistema que, se falhar, fará com que todo o sistema pare de funcionar [74]. Isso se justifica pois, caso o servidor de autenticação não esteja disponível, o usuário não faz login no sistema. Tampouco os microsserviços conseguem validar a requisição do cliente.

Dos entrevistados, 13 concordaram que nos projetos em que trabalham ou trabalharam, já ocorreram falhas de segurança envolvendo autenticação/autorização em microsserviço (Q.8). 17 Não concordaram, nem discordaram, e os demais entrevistados não tiveram esse tipo de problema. Na questão aberta (Q.9), foi solicitado para que o entrevistado, caso tivesse passado por algum problema, descrevesse quais dos desafios da Q.7 não foram observadas e o motivo.

A Tabela 4.2 apresenta as respostas obtidas. Os itens mais mencionados mais de uma vez foram "Uso de Servidor de Autenticação/autorização que lide com todos os microsserviços" (4 menções), "Times diferentes trabalhando em microsserviços diferentes" (4 menções), "falta de patterns de segurança entre microsserviços (3 menções) e "comunicação entre serviços" (2 menções). Foram mencionados ainda desafios que não se encontravam no rol da Q.7, como "tentativa de roubo do token", "abuso no servidor do token", "falta de documentação clara sobre os microsserviços" e "níveis e grupos de autenticação/autorização".

Importante mencionar a resposta na íntegra, de um entrevistado que descreve problemas causados com "Uso de Servidor de Autenticação/autorização que lide com todos os microsserviços":

> Basicamente, o problema que mais temos é o servidor de autenticação que controla os tokens ser um SPOF. Muitos serviços implementam sua autenticação considerando que o servidor nunca falhará e, é claro, ele falha de tempos em tempos, o que causa uma reação em cadeia, e nem todos os microsserviços da cadeia geralmente foram bem implementados o suficiente para lidar com falhas de maneira resiliente.

Observa-se ainda que há desafios complementares, pois, segundo alguns entrevistados, os times de desenvolvimento que atuam com microsserviços possuem conhecimentos distintos sobre padrões de segurança nesta arquitetura. A falta de *patterns* é mencionada como um desafio, já que ela ajudaria em tese a criar um time mais coeso e uma forma mais simples e eficiente para implementar seguranças neste modelo arquitetural.

A Q.13 dá a oportunidade para que o respondente expresse, de forma livre, algum outro desafio que ele tenha enfrentado ou enfrenta nos projetos de microsserviço. As respostas mais frequentes estão relacionadas com a falta de entendimento sobre o que de fato é um microsserviço e como ele deve ser implementado. Em complemento, é mencionado que a falta de entendimento pode desempenhar uma má modelagem de cada microsserviço, ou seja, saber se ele está fazendo mais que o necessário ou muitas poucas tarefas. Esta

Tabela 4.2: Desafios evidenciados em falhas de segurança observadas em projetos de TIC

Desafio	**Qtd**	**Motivo(s)**
Uso de servidor de autenticação/ autorização que lide com todos os microsserviços	4	Alta carga de requisições ao servidor, gerando SPOF
Times diferentes trabalhando em microsserviços diferentes	4	Os times podem ter conhecimentos distintos sobre padrões de segurança, aplicando de maneira diversa em cada projeto
Falta de patterns de segurança em microsserviços	3	Lidar com microsserviços é complexo, a adoção de modelos e patterns facilita a implementação com maior facilidade e segurança
Comunicação entre serviços	2	a falta de controle a um número elevado de microsserviços comprometeu a segurança entre a comunicação de alguns em específico
Falta de Validação no Código	1	não detalhado
Falta de validação do token	1	não detalhado
Abuso no servidor de token	1	não detalhado
Aumento na superfície de ataques	1	não detalhado
Autorização entre Serviços	1	não detalhado
Tentativa de roubo do token	1	não detalhado
Falta de documentação clara sobre os microsserviços	1	Os microsserviços são implementados sem a correta documentação e os times se perdem em relação ao nível de segurança de cada um
Níveis e grupos de autenticação/ autorização	1	Dificuldade em integrar gerenciamentos de credenciais (ex: AD) aos microsserviços
Detecção de intrusão/monitoramento	1	não detalhado
Validação do token a cada requisição	1	Alta carga no serviço/servidor de validação
Falta de estudos sobre segurança em microsserviços	1	não detalhado

falta de definições básicas podem colaborar para que conceitos mais avançados, como segurança, sejam menos observados.

A padronização, embora já tenha sido observada nos achados da RSL, também é mencionado, especificamente quanto a boas práticas no desenvolvimento, falta de código a ser reaproveitado, bibliotecas e alinhamento dos padrões de microsserviços com os sistemas legados e pouca documentação no projeto.

Por fim, mas não menos importante, é mencionada a falta de conhecimento dos desenvolvedores em relação a transações distribuídas. Mesmo no ambiente de microsserviços, é observado pelo respondente que programadores ainda implementam chamadas como se

cada serviço ainda estivesse em uma estrutura monolítica.

4.2.2 RQ2

As Q.10 e Q.14 dizem respeito aos protocolos que o respondente utiliza ou já utilizou para mitigar os desafios de autenticação e autorização em microsserviços. As respostas obtidas são apresentadas na Figura 4.2.

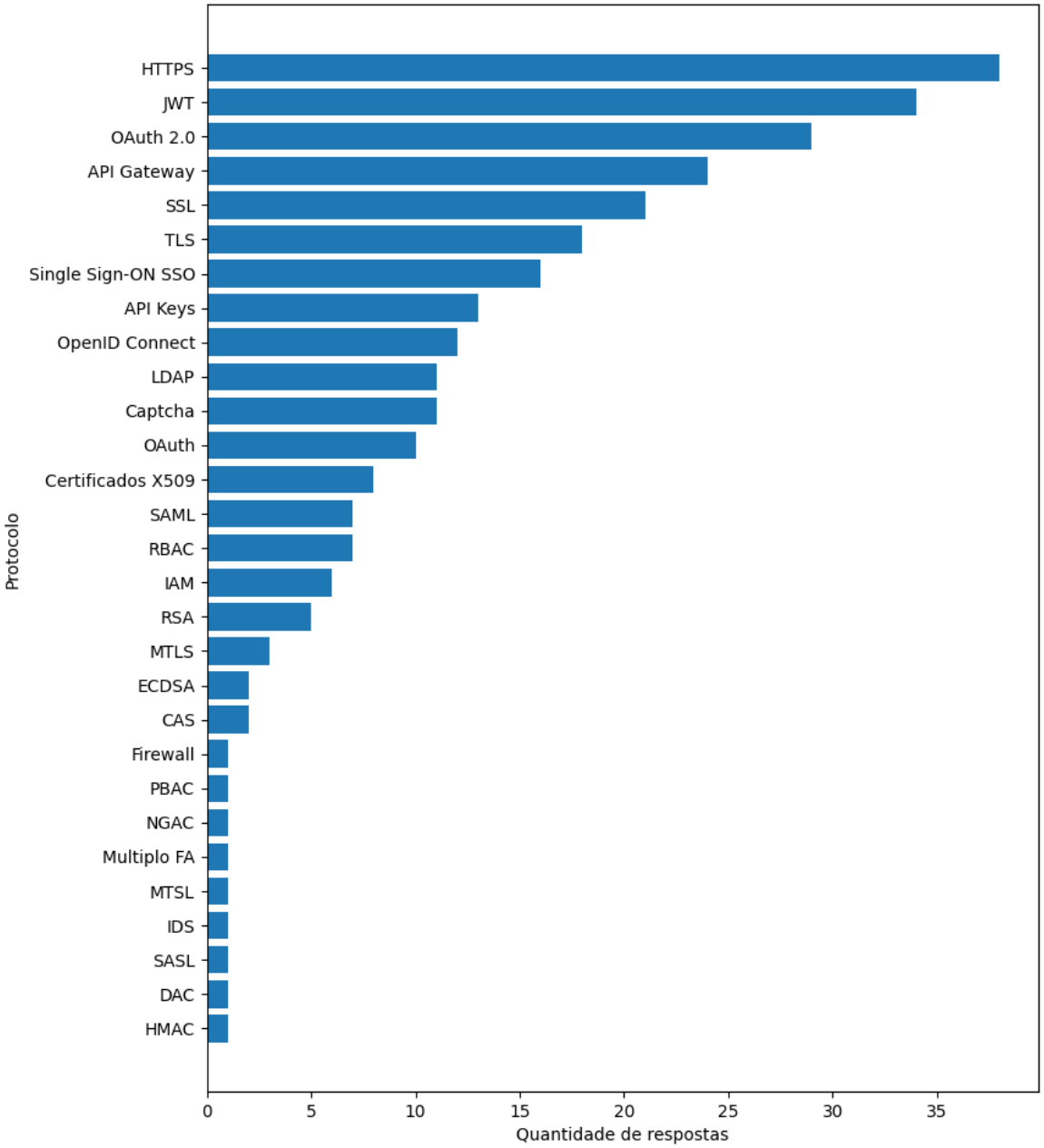


Figura 4.2: Protocolos utilizados em autenticação/autorização na arquitetura de microsserviços

É possível visualizar que os achados da RSL estão em consonância com os protocolos observados pelos respondentes, onde são mencionados em grande quantidade o uso de HTTPS (38), JWT (34), Oauth2 (29), API Gateway (24), Single Sign-ON (16) e OpenID Connect (12).

A Q.14 dá a oportunidade para que o respondente expresse, de forma livre, outras sugestões que ajudem a mitigar os desafios de segurança ao se trabalhar com arquitetura de microsserviços. Houve frequência no número de respostas que faz referência ao uso de contêineres e implementação de ferramentas automatizadas para monitoramento de infraestrutura, sobretudo no tráfego de redes durante a comunicação dos microsserviços. É sugerido ainda com certa frequência que se utilizem soluções já consolidadas e testadas, preferencialmente do tipo *open source*, ao invés de implementar uma própria biblioteca que lide com microsserviços.

4.2.3 RQ3

As Q.11, Q.12 e Q.15 dizem respeito às tecnologias *open source* que implementam os protocolos observados no estudo. 46 entrevistados afirmaram que utilizaram tecnologias *open source*. Posteriormente, foi solicitado para que eles indicassem quais tecnologias foram utilizadas. A questão elencou os achados da RSL mas foi deixada uma opção específica para que se identificasse outra solução que não está na lista. As respostas se encontram na Figura 4.3.

As tecnologias mais mencionadas foram o Spring Cloud (17), Kong (4), Jarvis (3) e VertX (3). Vale salientar que todas estas foram encontradas nos estudos primários selecionados na RSL. Houve tecnologias fora da RSL citadas com frequência, como por exemplo, o Keycloak (3) e o Istio (3). O Keycloak é um sistema *open source* para gerenciamento de identidades [87]. Istio é "uma plataforma de *service mesh open source* que permite controlar a maneira como os microsserviços compartilham dados entre si" [88].

A Q.15 permitiu que o respondente relatasse, de forma livre, outras soluções *open source* sugeridas para implementar a segurança em arquitetura de microsserviços. O Keycloak é o que aparece com maior frequência, prova de que ele pode implementar o OAuth2, OpenID e outros protocolos, além de poder ser integrado com o Spring Cloud. O Spring Cloud, embora já identificado na RSL e citado no questionário, foi reforçado novamente pelos respondentes, que mencionaram que tal ecossistema pode implementar os principais protocolos mostrados como por exemplo, o OAuth2.

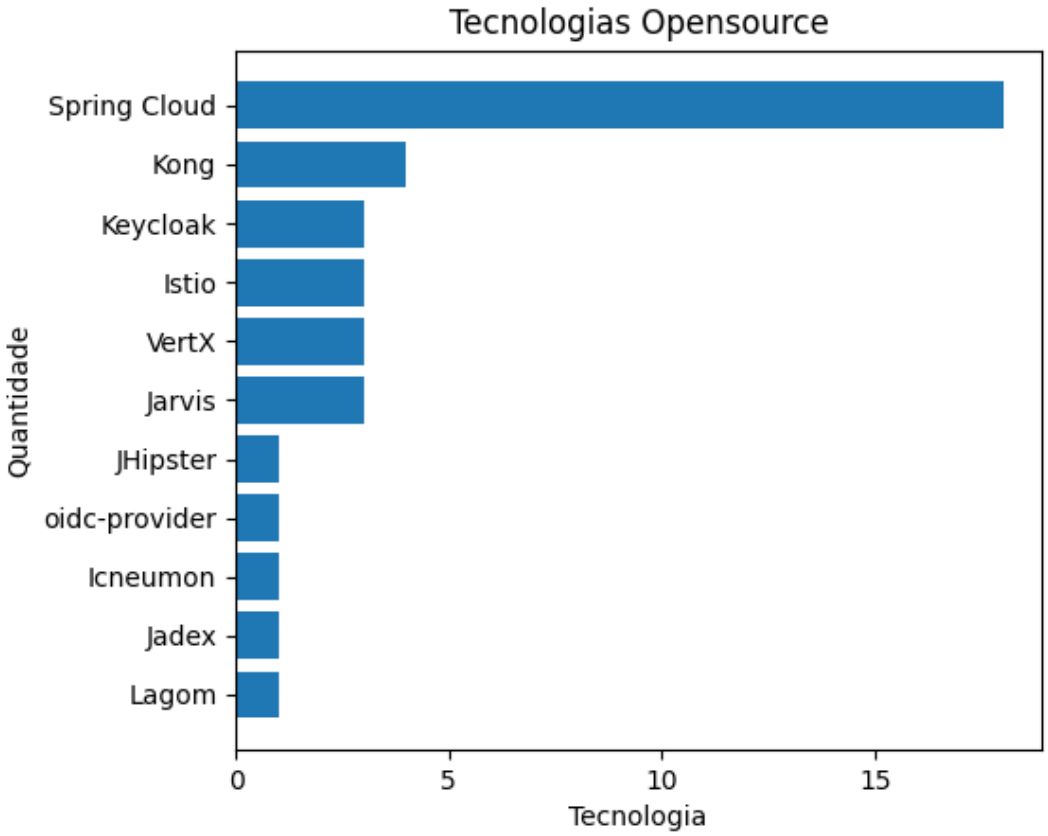


Figura 4.3: Tecnologias *open source* que implementam os protocolos utilizados em autenticação/autorização na arquitetura de microsserviços

4.3 Síntese do Capítulo

Este capítulo reforça os achados da RSL pois, após checagem através de *survey*, escolhendo como entrevistados profissionais de TIC que trabalham e possuem conhecimento sobre arquitetura de microsserviços, descobriu-se que em geral, os desafios, protocolos e implementações também são observados em ordem prática. A execução deste instrumento colaborou ainda a perceber que existem implementações *open source* que estão totalmente aderentes à realidade dos sistemas do Centro de Inteligência da Polícia Militar do Estado de São Paulo, como por exemplo as tecnologias Spring Cloud e Keycloak.

Capítulo 5

Proposta

De acordo com os nossos achados na execução da RSL e do Survey, onde foram verificados os desafios, protocolos e implementações open-source que lidam com autenticação e autorização de microsserviços, foi realizada uma comparação dos achados mais mencionados na literatura e nas respostas com a atual situação observada nos sistemas do Centro de Inteligência da PMESP, conforme apresentado na Tabela 5.1. Assim, propõe-se que sejam implementados os protocolos ausentes na arquitetura atual do Centro de Inteligência da PM (CIPM), com a utilização de uma tecnologia open-source.

Tabela 5.1: Protocolos e Tecnologias no Sistema de microsserviços do CIPM/PMESP

Protocolo	**Implementado no CIPM?**
OAuth2	Não
JWT	Sim
API Gateway	Sim
OpenID Connect	Não
Single Sign-ON	Não
Tecnologia	
Spring Cloud	Sim
Keycloak	Não
Kong	Não
Jarvis	Não
Istio	Não

Os sistemas implementados no Centro de Inteligência da PMESP já utilizam o ecossistema Spring, sobretudo o Spring Cloud, para desenvolvimento em arquitetura de microsserviços. Realizando buscas na página oficial do projeto (https://spring.io), é possível verificar guias para a implementação do OAuth2.0, OpenID e Single Sign On (SSO) utilizando a biblioteca "Spring Cloud Security" [89]. Há também na mesma página soluções para integração com o OpenID Connect. Desta forma, a utilização das bibliotecas de segurança do Spring Cloud poderia em tese implementar os protocolos ausentes.

O Keycloak, por sua vez, segundo a sua documentação oficial [90], possui a capacidade de adicionar autenticação em aplicativos e serviços seguros de uma forma simples, podendo aplicar protocolos padrões como OpenID Connect, OAuth2, SAML 2.0, Single-Sign On, além de prover integração com outros serviços como LDAP e Active Directory (AD). Foi discutido durante este trabalho, que as credenciais de usuário e senha são validadas em um serviço de AD, portanto, a utilização do Keycloak poderia tanto implementar os protocolos faltantes como providenciar novas integrações. O código-fonte do projeto está disponível no Github[91] e se encontra com elevado grau de engajamento além de constantes atualizações.

Conforme apresentado nas Figuras 1.1 e 1.2, os Sistemas do Centro de Inteligência utilizam um microsserviço de autenticação que checa as credenciais do usuário no Active Directory, se possui alguma permissão para utilizar o sistema e, caso positivo, realiza a montagem de um JWT, contendo informações básicas do usuário, as permissões que o usuário possui e o tempo de expiração. O processo de montagem do token é realizado na classe *TokenAuthenticationService.class*, demonstrada no Apêndice B. A criação do token é vislumbrada no método *addAuthentication*. O tempo de expiração do token é atualmente de 12 horas. Uma percepção desse tipo de implementação é que, caso o cliente estiver com o token gerado, ele terá as autorizações que foram dadas embutidas neste JWT, durante as 12 horas, ou seja, mesmo se ele não possuir mais tais permissões durante o período de validade do token. Também é percebido que, caso alguma nova regra seja adicionada a este usuário, ela só terá efeito após uma nova tentativa de login pelo usuário. É comum observar que, sempre após alguma alteração na aplicação, tanto no *front-end* quanto *back-end*, solicitamos aos usuários que realizem logoff e façam o login novamente, para absorverem essas modificações no novo token.

Utilizando-se um servidor próprio de autenticação e autorização com OAuth2 e OpenID, essas credenciais poderiam ser checadas em todas as requisições do cliente, portanto, caso houvesse qualquer tipo de mudança nas regras que o usuário possui, ou mesmo for desabilitado, a aplicação perceberia de imediato assim que fosse realizada alguma requisição, pois haveria checagem pelo servidor de autorização. A Figura 5.1 ilustra todo o fluxo que envolve o processo tanto de autenticação como autorização. Neste caso, o microsserviço não será mais utilizado para realizar a validação do token como na arquitetura atual, mas apenas irá enviar para que o servidor de autorização assim faça. Além dos benefícios já supracitados, não haverá mais a necessidade do armazenamento do "secret" responsável por decodificar o token, que estava presente em cada microsserviço.

Propõe-se assim, implementar as tecnologias *open source* supramencionadas, seja o Spring Cloud ou Keycloak de forma isolada ou conjunta, para que os protocolos observados sejam integrados à aplicação de inteligência e ajude a mitigar os principais desafios

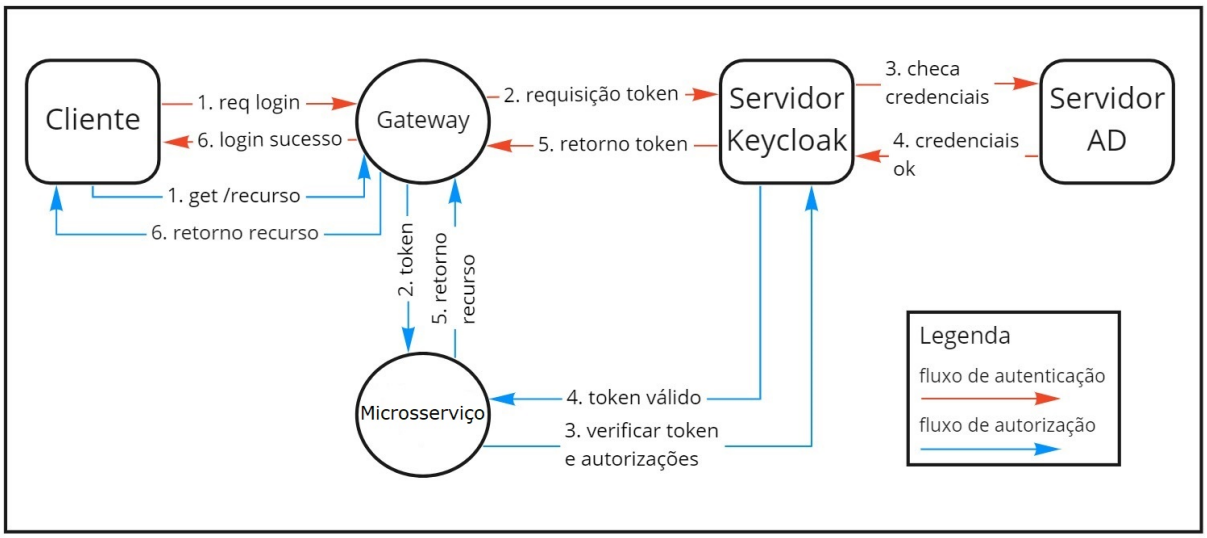


Figura 5.1: Sistemas e bases de dados isoladas, desenvolvidos no Centro de Inteligência da PMESP

envolvendo autenticação e autorização em uma arquitetura de microsserviços.

Capítulo 6

Implementação

Este capítulo demonstra o processo de implantação da solução e indica as bibliotecas utilizadas, o passo a passo e a demonstração do código-fonte.

6.1 Tecnologias e Bibliotecas Adotadas

Foi realizado o download do Keycloak Server versão 15.0.1 no site oficial da solução (https://www.keycloak.org/downloads). Preferiu-se, neste caso, baixar uma imagem de contêiner para uso em Docker. Ao subir a imagem do Keycloak Server, verificamos que o sistema possui uma interface web amigável, em que se pede para criar o "Realm"[90], que gerencia um grupo de usuários, credenciais, funções e grupos, com configurações gerais de segurança para ser utilizado na aplicação em questão.

O sistema possui ainda uma tela de configuração dos clients que irão consumir o Keycloak, em que podemos adicionar por exemplo o *front-end* web e o *back-end*.

Um recurso importante do Keycloak para o projeto de inteligência da PMESP, é a de "Federação de Usuários", utilizando-se o recurso LDAP disponível no ambiente da PMESP. Desta forma, pode ser implementado o mesmo serviço de autenticação que já era utilizado no sistema passado. Antes da implementação do Keycloak, foi escrito a classe *ADAuthenticationManager.class* visando a configuração e utilização do LDAP, conforme demonstrado no Apêndice B. Desta forma, não existe mais a necessidade de utilizar tal classe, o que reduz a quantidade de código-fonte e a complexidade de manutenção.

Outro recurso aderente à atual arquitetura do sistema é o de *roles* e grupos. As roles são as permissões para acessar os recursos no sistema, já o grupo é um conjunto de roles, de acordo com o perfil do usuário. Na aplicação, tal conceito já era utilizado. Ao usuário se logar, o sistema verificava no banco de dados a que grupo pertence, suas respectivas roles, para assim realizar a montagem do token de acesso. Com o keycloak, é possível

realizar essa customização na própria aplicação, embora seja ainda possível a integração com o banco de dados.

Na aplicação propriamente dita, aproveitou-se para atualizar a biblioteca do serviço de Gateway, que até então era o Spring Cloud Zuul, que se tornou descontinuado no ecossistema Spring. Implementou-se então a biblioteca "Spring Cloud Gateway" (org.springframework.cloud:spring-cloud-starter-gateway), além da OAuth2 Resource Server (org.springframework.boot:spring-boot-starter-oauth2-resource-server), esta última possuindo os recursos necessários para se comunicar com o servidor Keycloak e utilizar o protocolo OAuth2. É possível verificar no Apêndice C, como o serviço de Gateway ficou enxuto em relação ao que era observado na aplicação antiga. Nas configurações de propriedade, basta apontar em qual o endereço IP será validado o token quando o gateway for acionado. Na classe *WebSecurityConfig.class*, basta apenas apontar que as requisições para este serviço devem estar autenticadas e será utilizado um recurso Oauth2 para realizar a validação do JWT.

No *front-end* web, também foi necessária a implementação de uma biblioteca específica para integração com o OAuth2. Neste caso, foram criadas as bibliotecas "keycloak-angular"[92] e "keycloak-js"[93]. Tais bibliotecas foram necessárias para a checagem da existência ou validade do token do client, para assim redirecionar até a página de login, caso necessário. O Apêndice D demonstra as classes necessárias para instalação das bibliotecas.

6.2 Vantagens e desafios percebidos

Conforme verificado neste capítulo, a implementação do keycloak ajudou a reduzir linhas de código do projeto original, pois já possui uma série de recursos usados de maneira nativa, não sendo mais necessária a codificação deles, como por exemplo a configuração para conexão com o Active Directory e utilização do serviço LDAP para validar a autenticação, pois tudo isso é feito pelo próprio Keycloak, conforme verificado na figura 6.1. A tabela 6.1 demonstra quão significativa foi a redução em número de classes Java e linhas de código. Ele também avoca a responsabilidade para a montagem do token que será enviado ao client e não há mais necessidade da implementação de "filtros" de token. Basta apenas importar a biblioteca OAuth2 Resource Server (org.springframework.boot:spring-boot-starter-oauth2-resource-server) e indicar que a aplicação deve validar os tokens com um recurso Oauth2 e apontar no arquivo application.properties os endereços para realizar tal validação. Ainda sobre o token, é possível revogá-lo a qualquer instante, diferente da versão atual, onde o token tem a duração de 12 horas e pode ser utilizado por todo esse período mesmo que o usuário tenha sido eliminado do registro da plataforma.

Tabela 6.1: Comparativo de número de classes e linhas de código

Medida	Solução Atual	Solução Proposta
Número de classes	21	3
Linhas de Código	1139	50

Não houve dificuldades em implementar as bibliotecas do OAuth2. Bastou importar do próprio repositório oficial do ecossistema Spring. Na verdade, o Spring Cloud sequer sabe que o servidor é keycloak, pois está apenas preocupado em utilizar tal protocolo.

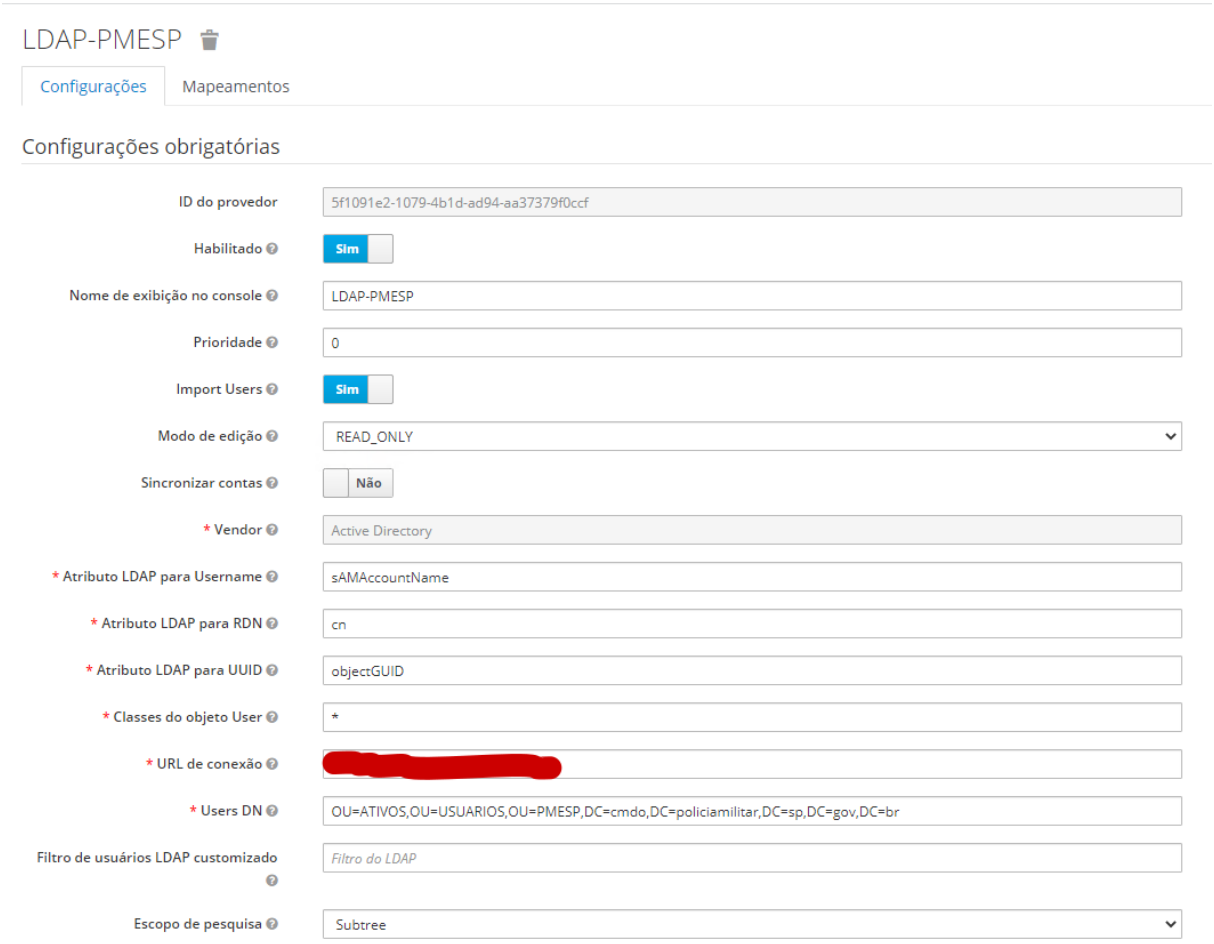


Figura 6.1: Configuração de Serviço LDAP para autenticação no Keycloak

O keycloak também implementa uma tela própria de login, conforme verificado na Figura 6.2 que pode ser customizada com a identidade visual do sistema. Mostramos na mesma figura a url do servidor de autenticação, fazendo menção ao uso do protocolo OAuth2 e OpenID Connect. Esta mesma tela de login pode ser reaproveitada para diversos outros sistemas que estejam integrados ao keycloak, o que não é atualmente o caso, mas pode ser configurado no futuro, fazendo a estratégia Single SignOn atingir o seu objetivo, ou seja, diversos sistemas utilizados com um único recurso de autenticação ao usuário.

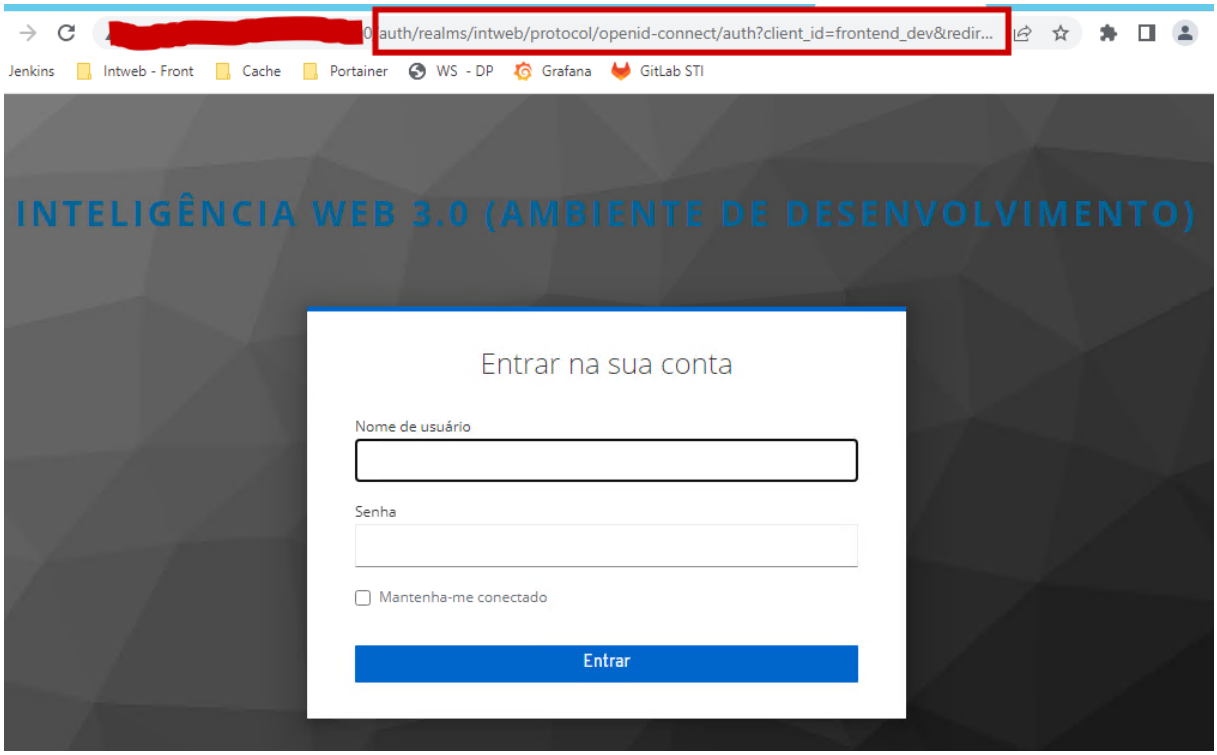


Figura 6.2: Tela de login com implementação do Keycloak

O Keycloak também possui um painel administrativo com tela amigável e intuitiva para uso, ajudando os profissionais menos experientes a entender com mais facilidade os recursos deste servidor.

A implementação de biblioteca para checagem do JWT no *front-end* traz o benefício de possibilitar a revogação do token durante a própria navegação do usuário. No moedlo anterior, um JWT teria a validade de 12 horas e só seria destruído caso o usuário fizesse logout. Neste novo cenário, é possível revogar este token e o usuário não poderá mais o utilizar, já que em todas as requisições terá uma validação prévia sobre a sua validade.

Os desafios encontrados serão de explorar os diversos recursos oferecidos pela solução, que vão além do proposto neste projeto. Embora esta seja uma solução *open source* que implemente ou facilite a implementação dos mecanismos apontados no Capítulo 5, há uma série de outros mecanismos que podem ser implantados, muitos deles verificados na Tabela 3.5

Capítulo 7

Testes de Segurança

Este capítulo pretende fornecer uma análise comparativa das diferentes soluções discutidas neste trabalho, ou seja, a solução atual implementada no CIPM com a nova solução que utiliza a solução proposta neste trabalho. A segurança e o desempenho dos modelos foram analisados para determinar qual opção produziu os melhores resultados.

Foram realizados testes com o SonarQube [94] para apurar possíveis defeitos nas soluções e verificar a cobertura de testes nos modelos. Com isso, é possível analisar quais sistemas têm maior probabilidade de conter uma possível vulnerabilidade crítica no código. Também se realizou ataques automatizados com as ferramentas OWASP Zap [95] e Nikton [96], soluções que realizam diversos testes de vulnerabilidade na aplicação e servidor web. Todas as ferramentas utilizadas neste capítulo foram sugestionadas pela equipe do Laboratório de Operações Cibernéticas da Diretoria de Operações Integradas do Ministério da Justiça e Segurança Pública (MJSP) [97].

7.1 Execução de Testes com Sonarqube

Segundo a documentação oficial [94], o O SonarQube® é uma ferramenta automática de revisão de código para detectar bugs, vulnerabilidades, *Security Hotspots* e *codesmells* em seu código. Ele pode ser integrado ao fluxo de trabalho existente para permitir a inspeção contínua de código em todas as ramificações do projeto. Ao realizar os testes usando a ferramenta SonarQube, o próprio sistema gera um relatório em que é possível identificar bugs, vulnerabilidades, pontos críticos de segurança (segundo o sistema, são trechos de código que necessitam de uma revisão manual para verificar se realmente se configura como uma vulnerabilidade) e *code smells*, um termo utilizado para descrever potenciais problemas no design do software [98]. Em suma, o *code smell* é um indício de uma programação ruim ou equivocada, que poderia resultar em um problema futuro.

A figura 7.1 demonstra os resultados dos testes da ferramenta SonarQube na camada de segurança da atual aplicação. É possível verificar que foram encontrados 8 (oito) bugs, 8 (oito) vulnerabilidades, 3 (três) pontos de segurança e 210 (duzentos e dez) *code smells*.

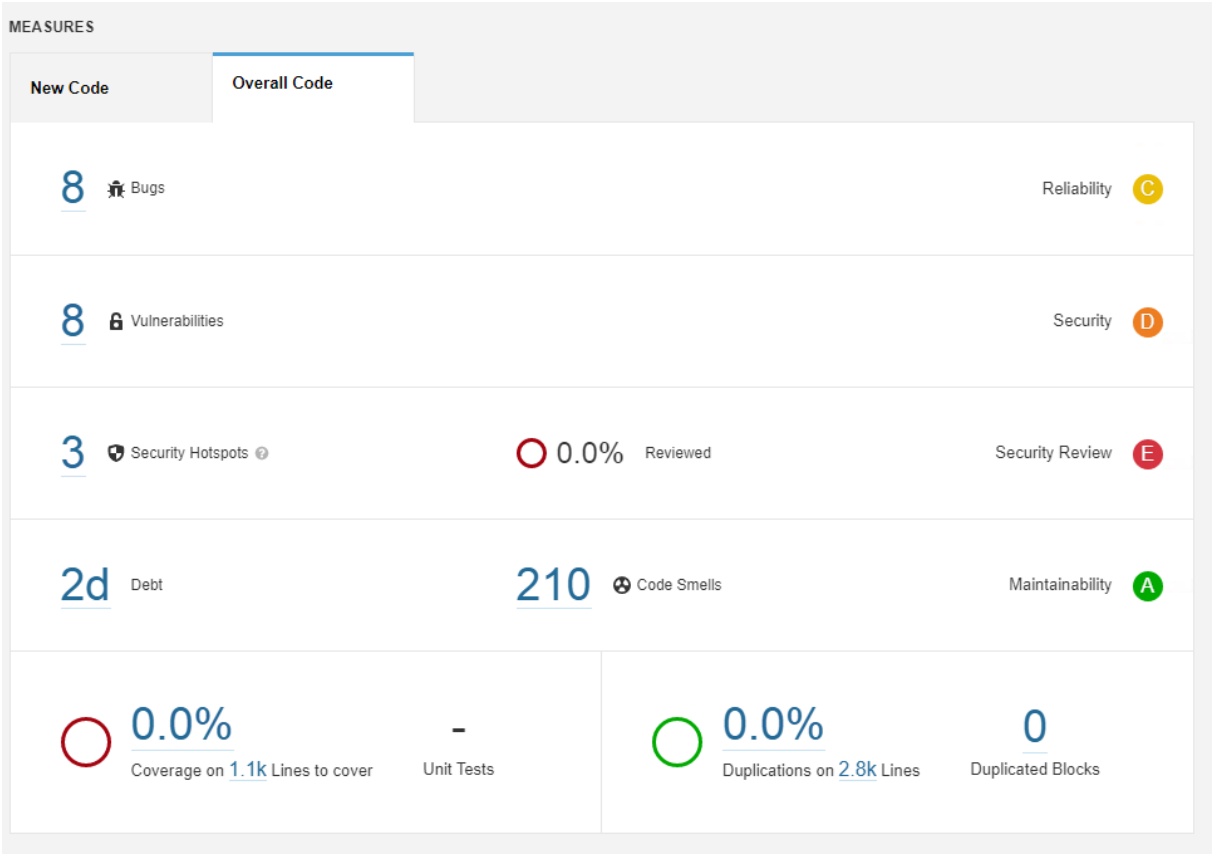


Figura 7.1: Resultado dos testes realizados no SonarQube no ambiente atual.

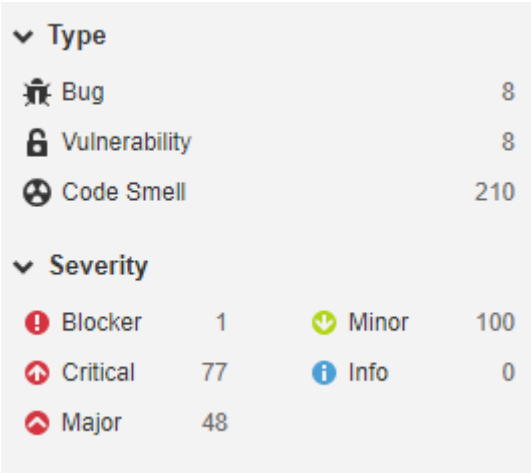


Figura 7.2: Categorias das severidades encontradas no ambiente atual.

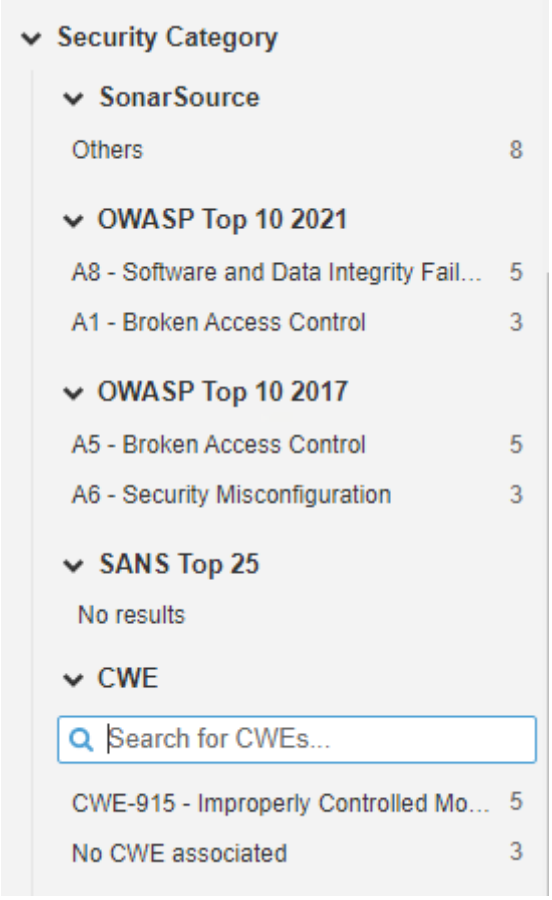


Figura 7.3: Categorias de segurança encontradas no ambiente atual.

A figura 7.4 demonstra os resultados dos testes da ferramenta SonarQube na camada de segurança da aplicação proposta, utilizando as novas bibliotecas do Spring Cloud, Keycloak e implementação dos protocolos OAuth2 e OpenID, além dos mecanismos propostos no Capítulo 5. É possível verificar que neste novo cenário, foram encontrados 0 (zero) bugs, 0 (zero) vulnerabilidades, 0 (zero) pontos de segurança e 3 (três) *Code Smells*, um número relativamente baixo de incidências comparado com a solução atual. A tabela 7.1 demonstra os dados dos dois cenários.

7.2 Execução de testes com OWASP Zazp

Segundo documentação oficial [95], O Zed Attack Proxy (ZAP) é uma ferramenta gratuita de teste de penetração de código aberto, mantida sob a égide do Open Web Application Security Project (OWASP). O ZAP foi projetado especificamente para testar aplicativos da web.

Ainda segundo o site oficial, o ZAP pode ser conhecido como um *proxy man-in-the-middle*. Ele fica entre o navegador do testador e o aplicativo da web para que possa

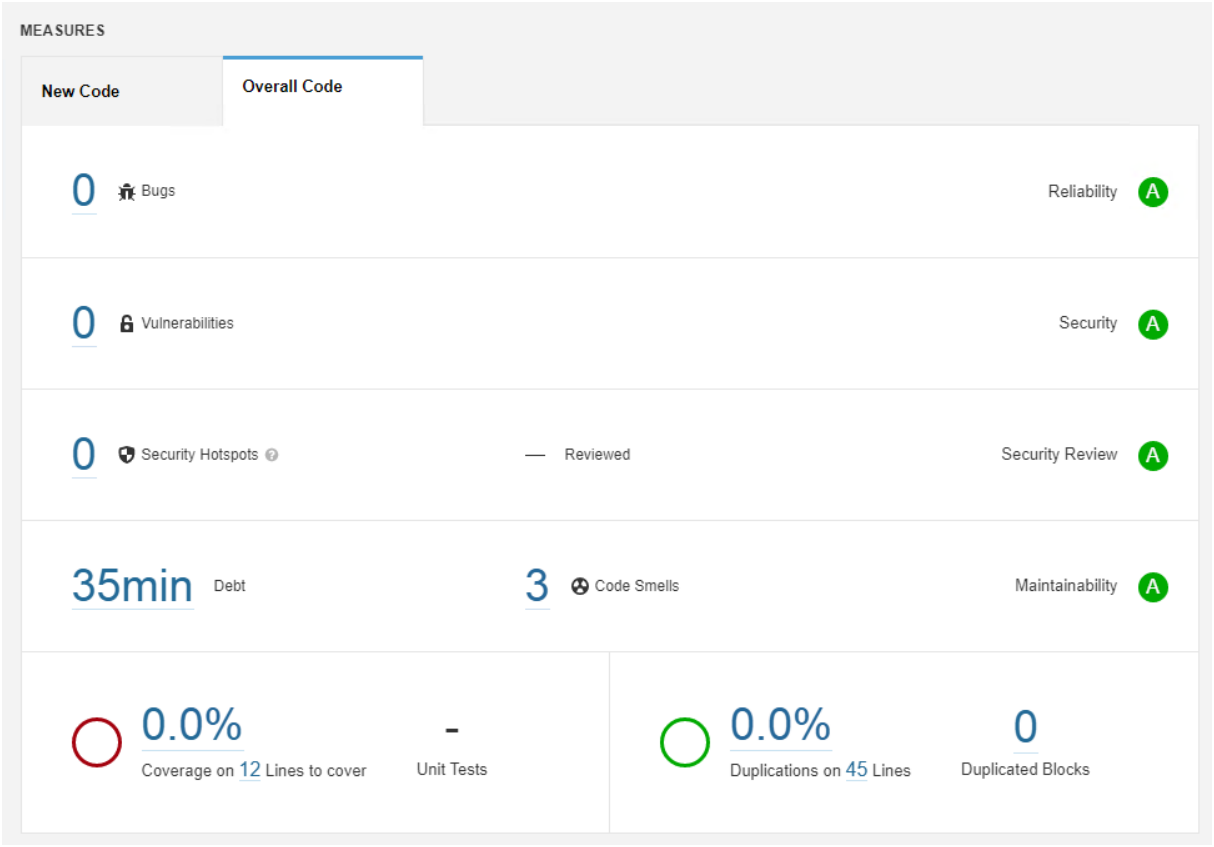


Figura 7.4: Resultado dos testes realizados no SonarQube no novo ambiente.

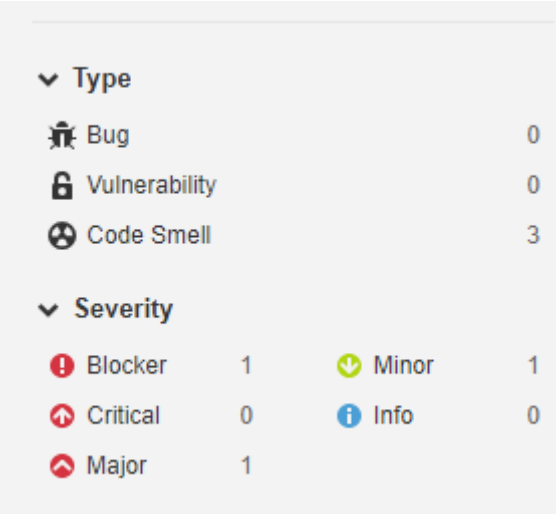


Figura 7.5: Categorias das severidades encontradas no ambiente novo.

interceptar e inspecionar as mensagens enviadas entre o navegador e o aplicativo da web, modificar o conteúdo se necessário, e encaminhar esses pacotes para o destino.

Para realização do teste, foi utilizado o ZAP como ferramenta de proxy, em que foram abertos os dois ambientes (atual e novo), sendo executados os mesmos comandos durante

Tabela 7.1: Comparação de quantitativo de problemas e nota atribuída pela ferramenta SonarQube

Medida	Solução Atual (Quantidade / Nota)	Solução Proposta (Quantidade / Nota)
Bugs	8 / C	0 / A
Vulnerabilidades	8 / C	0 / A
Pontos de Segurança	3 / E	0 / A
Code Smell	210 / A	3 / A

a navegação. Basicamente, foi efetuado um login na aplicação e usados seus recursos internos, como pesquisas, cadastros e visualizações. Seguiu-se exatamente a mesma sequência de ações em ambos os cenários. Após os registros das rotas, executou-se o ataque automatizado da própria ferramenta.

Verificou-se que durante os testes no atual ambiente, a ferramenta mostrou alertas em alguns ataques em específico, conforme Figura 7.6. Os alertas foram relacionados a *cross site scripting weakness (reflected in JSON response)* (3 alertas), *buffer overflow* (4 alertas) e *format string error*. Segue descritivo dos erros encontrados, definido pelo próprio relatório da ferramenta:

- **Cross Site Scripting Weakness[99]:** Cross-site scripting (XSS) é uma técnica de ataque que envolve injetar código malicioso fornecido pelo invasor na instância do navegador de um usuário. O código é geralmente escrito em HTML/JavaScript. Quando um invasor faz com que o navegador de um usuário execute seu código malicioso, é possível que o código seja executado em um contexto onde haja possibilidade de leitura, modificação e envio de dados sensíveis transmitidos pelo browser.
- **Buffer Overflow[100]:** Os problemas de *buffer overflow* são caracterizados pela substituição de espaços de memória do processo da web em segundo plano, que nunca deveriam ter sido modificados, intencionalmente ou não.
- **Format String Error[101]:** Um erro de *format string* ocorre quando os dados enviados de uma *string* de entrada são avaliados como um comando pelo aplicativo.

Utilizando o mesmo teste, agora executando no novo ambiente proposto no Capítulo 5, não foram identificados alertas, conforme verificado na Figura 7.7.

7.3 Execução de testes com Nikto

Segundo documentação oficial [96], o Nikto é um scanner de servidor web de código aberto (GPL) que executa testes abrangentes em servidores web para vários itens, incluindo mais

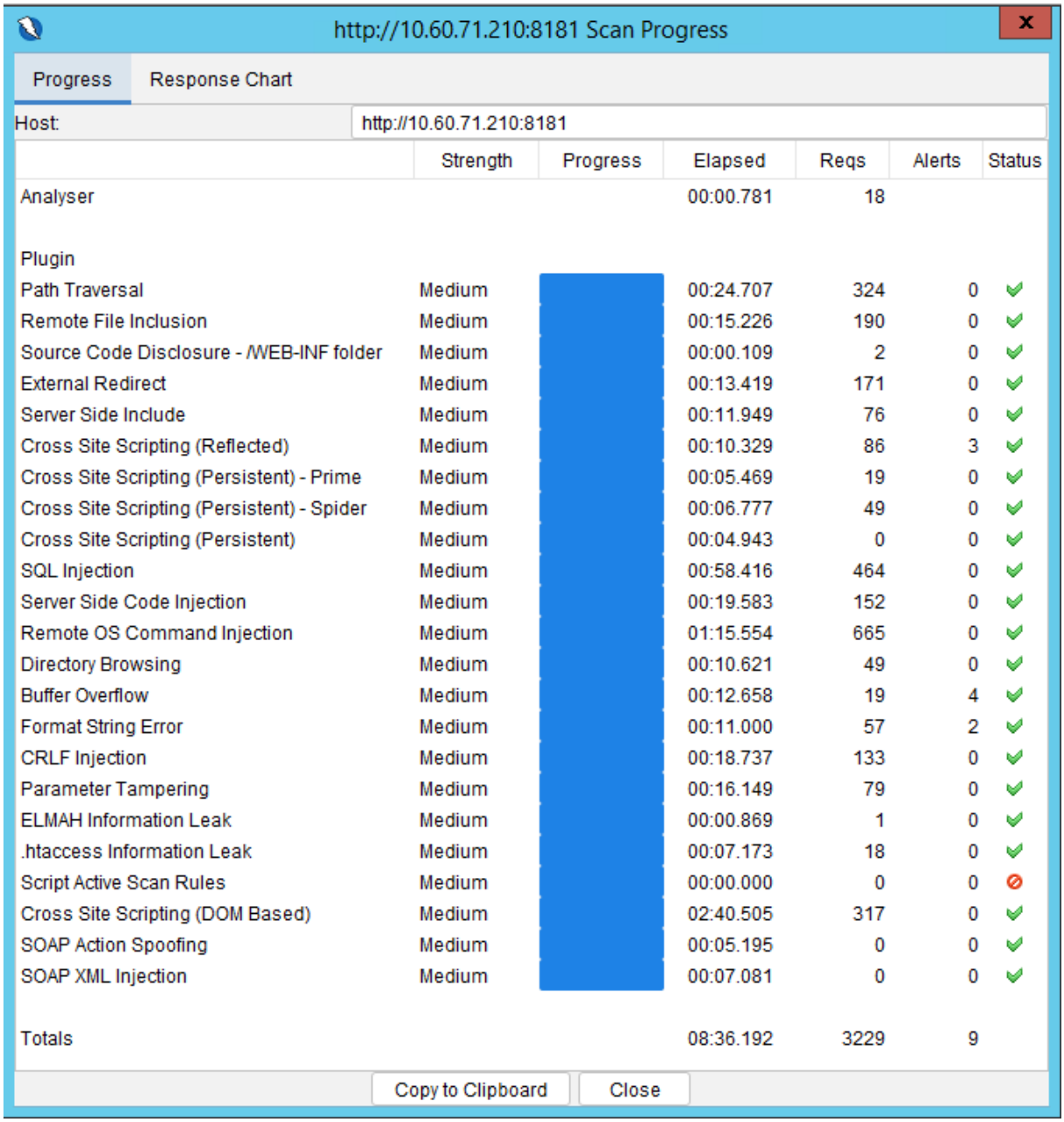


Figura 7.6: Resultado dos testes realizados no OWASP Zap no antigo ambiente.

de 6700 arquivos ou programas potencialmente perigosos, versões desatualizadas de cerca de 1250 servidores e problemas específicos encontrados em 270 deles. Ele também verifica itens de configuração do servidor, como a presença de arquivos de índice e requisições HTTP, tentando e softwares de web instalados e realizando testes de penetração neles. O objetivo do projeto é examinar um servidor web para encontrar possíveis problemas e vulnerabilidades de segurança, incluindo:

- Configurações incorretas de servidor e software

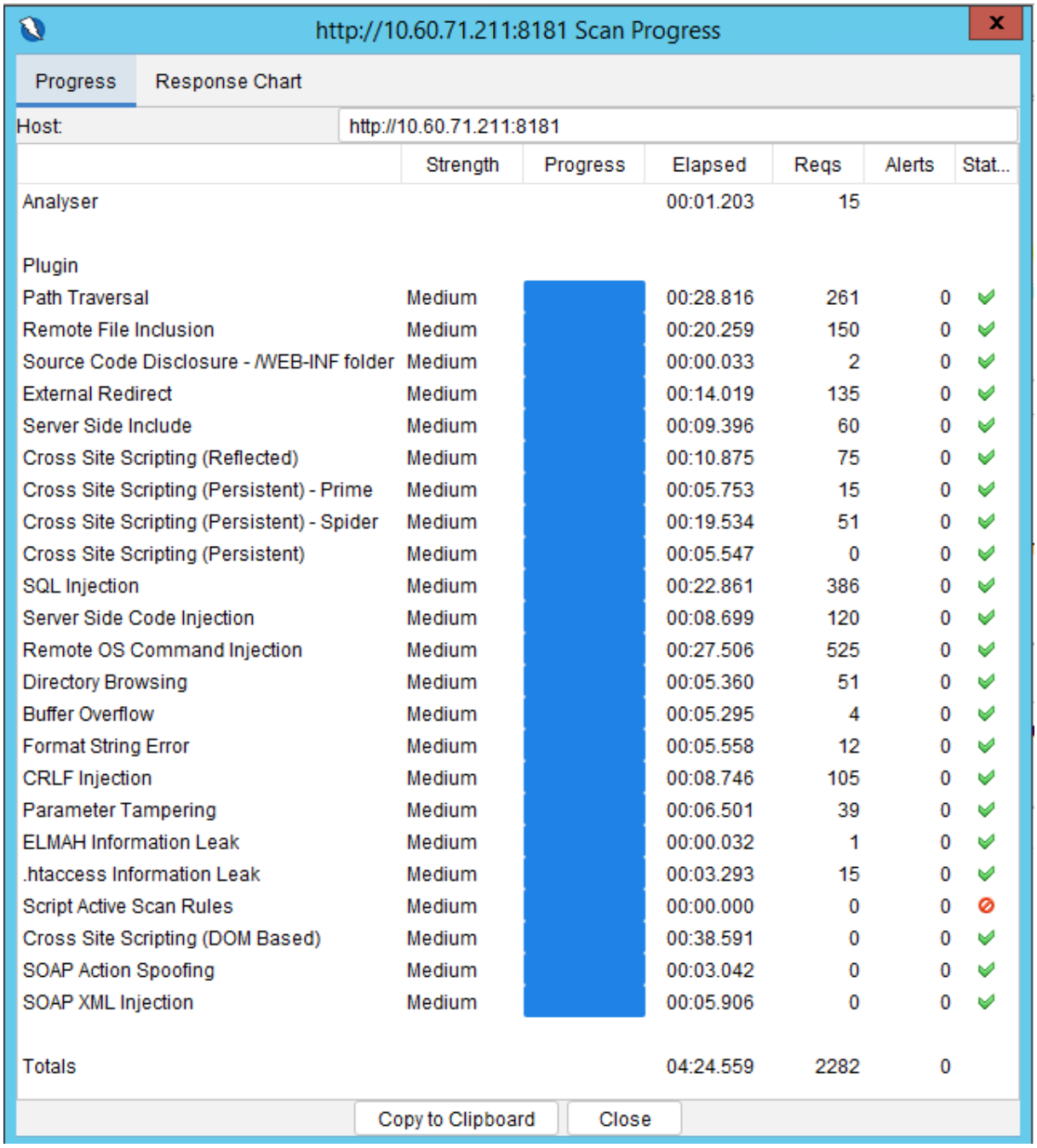


Figura 7.7: Resultado dos testes realizados no OWASP Zap no novo ambiente.

- Arquivos e programas padrão
- Arquivos e programas inseguros
- Servidores e programas desatualizados
- Ponteiros para levar um testador humano a melhores testes manuais

Foi utilizada a distribuição Kali Linux, que possui o Nikto instalado de forma padrão.

Os testes foram executados diretamente nos endereços IP e portas das aplicações hospedadas, seja da solução atual como a proposta. Verificou-se novamente que a aplicação atual apresenta mais potenciais problemas do que a proposta. A Figura 7.8 demonstra a execução dos testes e os respectivos resultados no ambiente atual (5 descobertas), sendo a Figura 7.9 executando os mesmos testes e demonstrando o resultado no ambiente proposto (1 descoberta). Das descobertas encontradas, as que exigem mais atenção são duas, conforme obervado pela própria ferramenta: *HTTP method (allow header)*: "O método PUT pode permitir que os clientes salvem arquivos no servidor web"; *HTTP method (Allow Header)*: "DELETE pode permitir que clientes removam arquivos no servidor web". Tais questões não foram encontradas no ambiente novo, inclusive há uma identificação que o novo ambiente exige autenticação para realm, conforme apresentado na Figura 7.9, o que pode ter prejudicado a própria execução da ferramenta de intrusão.

```
┌──(kali㉿kali)-[~]
└─$ nikto -h 10.60.71.210 -p 8181 -o reportold.html
- Nikto v2.1.6
---------------------------------------------------------------------------
+ Target IP:          10.60.71.210
+ Target Hostname:    10.60.71.210
+ Target Port:        8181
+ Start Time:         2022-07-15 21:57:51 (GMT-4)
---------------------------------------------------------------------------
+ Server: No banner retrieved
+ Uncommon header 'content-disposition' found, with contents: inline;filename=f.txt
+ All CGI directories 'found', use '-C none' to test none
+ Retrieved access-control-allow-origin header: nikto.example.com
+ Allowed HTTP Methods: GET, HEAD, POST, PUT, DELETE, OPTIONS
+ OSVDB-397: HTTP method ('Allow' Header): 'PUT' method could allow clients to save files on the web server.
+ OSVDB-5646: HTTP method ('Allow' Header): 'DELETE' may allow clients to remove files on the web server.
+ 26547 requests: 0 error(s) and 5 item(s) reported on remote host
+ End Time:           2022-07-15 22:14:42 (GMT-4) (1011 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
```

Figura 7.8: Resultado dos testes realizados no Nikto no ambiente atual.

```
┌──(kali㉿kali)-[~]
└─$ nikto -h 10.60.71.211 -p 8181 -o report.html
- Nikto v2.1.6
---------------------------------------------------------------------------
+ Target IP:          10.60.71.211
+ Target Hostname:    10.60.71.211
+ Target Port:        8181
+ Start Time:         2022-07-15 21:37:54 (GMT-4)
---------------------------------------------------------------------------
+ Server: No banner retrieved
+ / - Requires Authentication for realm ''
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ Retrieved access-control-allow-origin header: *
+ 8068 requests: 0 error(s) and 1 item(s) reported on remote host
+ End Time:           2022-07-15 21:42:21 (GMT-4) (267 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
```

Figura 7.9: Resultado dos testes realizados no Nikto no novo ambiente.

Capítulo 8

Conclusões e Trabalhos Futuros

Durante este trabalho foi apresentado o contexto dos sistemas desenvolvidos no Centro de Inteligência da Polícia Militar do Estado de São Paulo e o que motivou a adoção de determinadas tecnologias e arquiteturas. Com a modernização dos sistemas legados existentes, alcançou-se uma arquitetura de microsserviços, justificada por necessidades e amadurecimento percebidos durante o processo de evolução das aplicações. Todavia, não se esgotou a preocupação com questões de segurança neste novo ambiente arquitetural, sobretudo relacionado com autenticação e autorização dos microsserviços, por conta da natureza sensível do conhecimento produzido em uma atividade de inteligência policial.

Para melhor entendimento dos os desafios identificados nesta arquitetura, tal como formas de mitigar através do uso de mecanismos implementados em tecnologias open-source, realizou-se uma Revisão Sistemática de Literatura que encontrou uma série de desafios, mecanismos e tecnologias. Com estes achados, buscou-se dar maior robustez em tais evidências com aplicação de survey a profissionais que possuem experiência nesta área, e resultou em grande aderência ao que foi encontrado na Revisão Sistemática de Literatura.

Junto com o material coletado na execução da metodologia supramencionada, realizou-se uma comparação com as implementações existentes, em que foi possível perceber que tanto alguns mecanismos como tecnologias já estavam implementadas, mas ainda seria possível aplicar outros mecanismos. Neste caso, o sistema já estava desenvolvido dentro do ecossistema Spring, o que facilitaria o uso de novas bibliotecas do Spring Cloud ou mesmo integração com outro tipo de sistema, como o Keycloak.

A implementação dos novos mecanismos ajudou a modernizar a camada de segurança da aplicação, além de reduzir a complexidade do código fonte do projeto, pois muitas configurações que eram implementadas manualmente por linhas de código, agora poderia ser gerenciadas pela própria ferramenta visual do Keycloak, inclusive a configuração de vários mecanismos como o OAuth2 e OpenID. Vislumbra-se ainda um impacto no time de

desenvolvimento que será afetado com a simplicidade do projeto, já que há uma perceptível redução de código e simplicidade para utilização da ferramenta visual do Keycloak para se usar mecanismos de segurança.

Verificou-se ainda, ao aplicar testes automatizados, que houve ganhos em relação ao projeto antigo com a diminuição de bugs, vulnerabilidades e erros no código, além de se mostrar mais resiliente a uma série de ataques catalogados pela OWASP. Isso reduziu substancialmente a possibilidade de sucesso em possíveis ataques cibernéticos que a aplicação possa sofrer.

O estudo possui limitações em relação ao número de bases digitais pesquisadas para realização da RSL. Pode ser ampliado em novas fontes para pesquisas futuras. O número de entrevistados no survey foi o possível dentro do ciclo pessoal/profissional dos pesquisadores, também podendo ser estendido como está ou mudado o critério de seleção para maiores buscas. Por fim, ainda há a possibilidade de executar testes de vulnerabilidade com diversas ferramentas ou mesmo de forma manual, para reforçar ainda mais a segurança da solução aplicada.

Vislumbra-se a aplicação, em trabalhos futuros, da avaliação em relação a outros aspectos além da segurança propriamente dita, como a percepção da equipe em relação às novas arquiteturas e tecnologias implementadas, avaliando se houve algum tipo de melhoria no processo de entendimento dos mecanismos de segurança e como tal implementação impactou o processo de construção de software, além de avaliar os ganhos em relação a interoperabilidade e capacidade integrativa da solução, já que pode ser utilizada em outros sistemas a serem introduzidas futuramente na instituição.

Referências

[1] Brasil: *Constituição da república federativa do brasil*, 1988. http://www.planalto.gov.br/ccivil_03/constituicao/constituicao.htm, acesso em 21 jun. 2021. 1

[2] Brasil: *Decreto nº 3.695 de 21 de dezembro de 2020. criação do subsistema de inteligência de segurança pública*, 2020. http://www.planalto.gov.br/ccivil_03/decreto/d3695.htm, acesso em 21 jun. 2021. 1

[3] Lewis, James e Martin Fowler: *Microservices - a definition of this new archtectural term*, 2014. https://martinfowler.com/articles/microservices.html, acesso em 23 out. 2021. 2, 24

[4] Halonen, Teppo: *Authentication and authorization in mobile environment*. Em *Tik-110.501 Seminar on Network Security*. Citeseer, 2000. 3, 24

[5] Jones, M, J Bradley e N Sakimura: *Rfc 7519: Json web token (jwt)*. IETF. May, 2015. 3, 38

[6] São Paulo, Policia Militar do Estado de: *Sistema de Gestão da Polícia Militar do Estado de São Paulo - GESPOL*. Imprensa Oficial, 2010. 5

[7] Brasil: *Lei no 9.883, de 7 de dezembro de 1999*, 1999. http://www.planalto.gov.br/ccivil_03/leis/l9883.htm, acesso em 21 jun. 2021. 5

[8] Cruz, Juliana Cristina da *et al.*: *A atividade de inteligência de segurança pública para o fortalecimento da cidadania*, 2013. 5

[9] Yarygina, Tetiana e Anya Helene Bagge: *Overcoming security challenges in microservice architectures*. Proceedings - 12th IEEE International Symposium on Service-Oriented System Engineering, SOSE 2018 and 9th International Workshop on Joint Cloud Computing, JCC 2018, páginas 11–20, maio 2018. 5, 25, 29, 32, 36, 39, 40, 41, 43

[10] Pippal, Sanjeev Kumar, Aruna Kumari e Dharmender Singh Kushwaha: *Ctes based secure approach for authentication and authorization of resource and service in clouds*. Em *2011 2nd International Conference on Computer and Communication Technology (ICCCT-2011)*, páginas 444–449. IEEE, 2011. 5, 24

[11] Kitchenham, B. e S Charters: *Guidelines for performing systematic literature reviews in software engineering*, 2007. 7, 24

[12] Kitchenham, Barbara, O. Pearl Brereton, David Budgen, Mark Turner, John Bailey e Stephen Linkman: *Systematic literature reviews in software engineering - a systematic literature review.* Information and Software Technology, 51:7–15, janeiro 2009, ISSN 09505849. 7, 24

[13] Kitchenham, Barbara Ann, David Budgen e Pearl Brereton: *Evidence-Based Software Engineering and Systematic Reviews.* Chapman; Hall/CRC, 2015, ISBN 1482228653. 7, 49, 50

[14] Bennett, Keith: *Legacy systems: Coping with success.* IEEE software, 12(1):19–23, 1995. 9

[15] Bisbal, J., D. Lawless, Bing Wu, J. Grimson, V. Wade, R. Richardson e D. O'Sullivan: *An overview of legacy information system migration.* Em *Proceedings of Joint 4th International Computer Science Conference and 4th Asia Pacific Software Engineering Conference*, páginas 529–530, 10.1109/APSEC.1997.640219, 1997. IEEE. 9

[16] De Lucia, A., G.A. Di Lucca, A.R. Fasolino, P. Guerra e S. Petruzzelli: *Migrating legacy systems towards object-oriented platforms.* Em *1997 Proceedings International Conference on Software Maintenance*, páginas 122–129, 10.1109/ICSM.1997.624238, 1997. IEEE. 9

[17] Knoche, Holger e Wilhelm Hasselbring: *Using microservices for legacy software modernization.* IEEE Software, 35(3):44–49, 2018. 9

[18] Abrams, Charles e W. Roy Schulte: *Service-oriented architecture overview and guide to soa research.* Relatório Técnico G00154463, Gartner, January 2008. `https://www.gartner.com/doc/575608/serviceoriented-architecture-overview-guide-soa`. 9

[19] Josuttis, N.M.: *SOA in Practice: The Art of Distributed System Design.* O'Reilly Media, USA, 2007, ISBN 9780596551551. 10

[20] Erl, Thomas: *Service-Oriented Architecture: Concepts, Technology, and Design.* Prentice Hall PTR, USA, 2005, ISBN 0131858580. 10

[21] De Lauretis, Lorenzo: *From monolithic architecture to microservices architecture.* Em *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, páginas 93–96, DOI: 10.1109/ICSM.1997.624238, 2019. IEEE, IEEE. 10

[22] Bucchiarone, Antonio, Nicola Dragoni, Schahram Dustdar, Stephan T Larsen e Manuel Mazzara: *From monolithic to microservices: An experience report from the banking domain.* Ieee Software, 35(3):50–55, 2018. 10, 11

[23] Luz, Welder, Everton Agilar, Marcos César de Oliveira, Carlos Eduardo R. de Melo, Gustavo Pinto e Rodrigo Bonifácio: *An experience report on the adoption of microservices in three brazilian government institutions.* Em *Proceedings of the XXXII Brazilian Symposium on Software Engineering*, SBES '18, página 32–41, New

York, NY, USA, 2018. Association for Computing Machinery, ISBN 9781450365031. https://doi.org/10.1145/3266237.3266262. 10, 19

[24] Merson, Paulo: *Microservices beyond the hype: What you gain and what you lose*, 2015. https://insights.sei.cmu.edu/blog/microservices-beyond-the-hype-what-you-gain-and-what-you-lose, acesso em 06 out. 2021. 11, 24

[25] Salleh, Syahanim Mohd, Zarina Shukur e Hairulliza Mohamad Judi: *Scaffolding model for efficient programming learning based on cognitive load theory.* Int. J. Pure Appl. Math, 118(7):77–83, 2018. 12

[26] Desmond, Brian, Joe Richards, Robbie Allen e Alistair G Lowe-Norris: *Active Directory: Designing, Deploying, and Running Active Directory.* " O'Reilly Media, Inc.", USA, 2008. 13

[27] Richardson, L. e S. Ruby: *RESTful Web Services.* O'Reilly Media, USA, 2008, ISBN 9780596554606. 15

[28] Fielding, Roy Thomas: *Architectural styles and the design of network-based software architectures.* University of California, Irvine, USA, 2000. 15

[29] Blischak, John D, Emily R Davenport e Greg Wilson: *A quick introduction to version control with git and github.* PLoS computational biology, 12(1):e1004668, 2016. 15

[30] Johnson, Rod, Juergen Hoeller, Keith Donald, Colin Sampaleanu, Rob Harrop, Thomas Risberg, Alef Arendsen, Darren Davison, Dmitriy Kopylenko, Mark Pollack *et al.*: *The spring framework–reference documentation.* interface, 21:27, 2004. 16

[31] Davis, Adam L: *Spring cloud.* Em *Spring Quick Reference Guide*, páginas 231–246. Springer, USA, 2020. 16

[32] Boettiger, Carl: *An introduction to docker for reproducible research.* ACM SIGOPS Operating Systems Review, 49(1):71–79, 2015. 17

[33] Smart, J.: *Jenkins: The Definitive Guide.* Oreilly and Associate Series. O'Reilly Media, USA, 2011, ISBN 9781449305352. 18

[34] Dwaraki, Abhishek, Srini Seetharaman, Sriram Natarajan e Tilman Wolf: *Gitflow: Flow revision management for software-defined networks.* Em *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, páginas 1–6, DOI: 10.1145/2774993.2775064, 2015. ACM. 18

[35] Apache: *Apache kafka*, 2021. https://kafka.apache.org/documentation/gettingStarted, acesso em 14 dez. 2021. 19

[36] Vargas Agilar, Everton de: *A service oriented approach to the modernization of unb's legacy systems.* Tese de Mestrado, University of Brasília, Brasília, Brazil, 2016. 20

[37] Carvalho, Luiz, Alessandro Garcia, Wesley K. G. Assunção, Rodrigo Bonifácio, Leonardo P. Tizzei e Thelma Elita Colanzi: *Extraction of configurable and reusable microservices from legacy systems: An exploratory study.* Em *Proceedings of the 23rd International Systems and Software Product Line Conference - Volume A*, SPLC '19, página 26–31, New York, NY, USA, 2019. Association for Computing Machinery, ISBN 9781450371384. https://doi.org/10.1145/3336294.3336319. 20

[38] Czarnecki, Krzysztof: *Variability in software: State of the art and future directions.* Em Cortellessa, Vittorio e Dániel Varró (editores): *Fundamental Approaches to Software Engineering*, páginas 1–5, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg, ISBN 978-3-642-37057-1. 20

[39] Agilar, Everton, Rodrigo Almeida e Edna Dias Canedo: *A systematic mapping study on legacy system modernization.* Em *The 28th International Conference on Software Engineering & Knowledge Engineering*, páginas 345–350, USA, 2016. SEKE. 20

[40] Laigner, Rodrigo, Marcos Kalinowski, Pedro Diniz, Leonardo Barros, Carlos Cassino, Melissa Lemos, Darlan Arruda, Sérgio Lifschitz e Yongluan Zhou: *From a monolithic big data system to a microservices event-driven architecture.* Em *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, páginas 213–220, DOI: 10.1109/SEAA51224.2020.00045, 2020. IEEE. 21

[41] Mazzara, Manuel, Nicola Dragoni, Antonio Bucchiarone, Alberto Giaretta, Stephan T. Larsen e Schahram Dustdar: *Microservices: Migration of a mission critical system.* IEEE Transactions on Services Computing, 14(5):1464–1477, 2021. 21

[42] Fritzsch, Jonas, Justus Bogner, Stefan Wagner e Alfred Zimmermann: *Microservices migration in industry: Intentions, strategies, and challenges.* Em *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, páginas 481–490, 2019. 21

[43] Fowler, Martin: *Monolithfirst*, 2015. https://martinfowler.com/bliki/MonolithFirst.html, acesso em 21 jun. 2021. 21

[44] Freitas, Vitor: *Parsifal*, 2021. https://parsif.al/, acesso em 17 out. 2021. 24

[45] Mateus-Coelho, Nuno, Manuela Cruz-Cunha e Luis Gonzaga Ferreira: *Security in microservices architectures.* Procedia Computer Science, 181:1225–1236, 2021, ISSN 18770509. 24, 25, 29, 33, 36, 39, 40, 41, 43

[46] Pereira-Vale, Anelis, Eduardo B. Fernandez, Raúl Monge, Hernán Astudillo e Gastón Márquez: *Security in microservice-based systems: A multivocal literature review.* Computers and Security, 103, abril 2021, ISSN 01674048. 24, 29, 32, 33, 36, 38, 39, 40, 43

[47] Barabanov, Alexander e Denis Makrushin: *Authentication and authorization in microservice-based systems: Survey of architecture patterns.* Voprosy Kiberbezopasnosti, 2020. 25, 29, 34, 35, 36, 39, 41

[48] Sun, Yuqiong, Susanta Nanda e Trent Jaeger: *Security-as-a-service for microservices-based cloud applications*. Proceedings - IEEE 7th International Conference on Cloud Computing Technology and Science, CloudCom 2015, páginas 50–57, fevereiro 2016. 25, 29, 33

[49] Mehraj, Saima e M. Tariq Banday: *Establishing a zero trust strategy in cloud computing environment*. Em *2020 International Conference on Computer Communication and Informatics (ICCCI)*, páginas 1–6, 2020. 25

[50] Rose, Scott, Oliver Borchert, Stu Mitchell e Sean Connelly: *Zero trust architecture*. Relatório Técnico, National Institute of Standards and Technology, 2020. 25

[51] Rossi, Bruno, Barbara Russo e Giancarlo Succi: *Adoption of free/libre open source software in public organizations: factors of impact*. Information Technology & People, 25(2):156–187, Jan 2012, ISSN 0959-3845. https://doi.org/10.1108/09593841211232677. 25

[52] Hippel, Eric von e Georg von Krogh: *Open source software and the "private-collective" innovation model: Issues for organization science*. Organization science, 14(2):209–223, 2003. 25

[53] Lavazza, Luigi: *Beyond total cost of ownership: Applying balanced scorecards to open-source software*. Em *International Conference on Software Engineering Advances (ICSEA 2007)*, páginas 74–74, 2007. 25

[54] Wohlin, Claes: *Guidelines for snowballing in systematic literature studies and a replication in software engineering*. Em *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, EASE '14, New York, NY, USA, 2014. Association for Computing Machinery, ISBN 9781450324762. https://doi.org/10.1145/2601248.2601268. 26

[55] Safaryan, Olga, Elena Pinevich, Evgenia Roshchina, Larissa Cherckesova e Nadezhda Kolennikova: *Information system development for restricting access to software tool built on microservice architecture*. E3S Web of Conferences, 224, dezembro 2020, ISSN 22671242. 29, 34, 42, 43

[56] Shulin, Yang e Hu Jieping: *Research on unified authentication and authorization in microservice architecture*. International Conference on Communication Technology Proceedings, ICCT, 2020-October:1169–1173, outubro 2020. 29, 35, 36, 38, 40, 41, 42

[57] Jin, Wenquan, Rongxu Xu, Taewan You, Yong Geun Hong e Dohyeun Kim: *Secure edge computing management based on independent microservices providers for gateway-centric iot networks*. IEEE Access, 8:187975–187990, 2020, ISSN 21693536. 29, 34, 36, 38, 39

[58] Nguyen, Quy e Oras Baker: *Applying spring security framework and oauth2 to protect microservice architecture api*. Journal of Software vol 14, 2019. 29, 34, 35, 36, 38, 42, 43

[59] Yu, Dongjin, Yike Jin, Yuqun Zhang e Xi Zheng: *A survey on security issues in services communication of microservices-enabled fog applications*. Concurrency and Computation: Practice and Experience, 31, novembro 2019, ISSN 15320634. 29, 33, 34, 35, 38, 39, 41, 42, 43

[60] Bhutada, Sunil e Kanuga Karuna Jyothi: *Enhancing security to the microservice (ms) architecture by implementing authentication and authorization service using docker and kubernetes*. International Journal of Innovative Technology and Exploring Engineering, 8, 2019, ISSN 2278-3075. 29, 38

[61] Krämer, Michel, Sven Frese e Arjan Kuijper: *Implementing secure applications in smart city clouds using microservices*. Future Generation Computer Systems, 99:308–320, outubro 2019, ISSN 0167739X. 29, 34, 40, 43

[62] Xu, Rongxu, Wenquan Jin e Dohyeun Kim: *Microservice security agent based on api gateway in edge computing*. Sensors (Switzerland), 19, novembro 2019, ISSN 14248220. 29, 34, 38, 39

[63] Nehme, Antonio, Vitor Jesus, Khaled Mahbub e Ali Abdallah: *Securing microservices*. IT Professional, 21:42–49, janeiro 2019, ISSN 1941045X. 29, 32, 34, 36, 38, 39

[64] Pereira-Vale, Anelis, Gaston Marquez, Hernan Astudillo e Eduardo B. Fernandez: *Security mechanisms used in microservices-based systems: A systematic mapping*. Proceedings - 2019 45th Latin American Computing Conference, CLEI 2019, setembro 2019. 29, 32, 37, 38, 43

[65] Banati, A, E Kail, K Karoczkai e M Kozlovszky: *Authentication and authorization orchestrator for microservice-based software architectures; authentication and authorization orchestrator for microservice-based software architectures*. 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2018. 29, 33, 34, 36, 38, 39, 40

[66] Jander, Kai, Lars Braubach e Alexander Pokahr: *Defense-in-depth and role authentication for microservice systems*. Procedia Computer Science, 130:456–463, 2018, ISSN 18770509. 29, 33, 34, 35, 40

[67] Nehme, Antonio, Vitor Jesus, Khaled Mahbub e Ali Abdallah: *Fine-grained access control for microservices*. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 11358 LNCS:285–300, 2019, ISSN 16113349. 29, 33, 36, 38, 39, 41

[68] Richter, Daniel, Tim Neumann e Andreas Polze: *Security considerations for microservice architectures*. CLOSER 2018 - Proceedings of the 8th International Conference on Cloud Computing and Services Science, 2018-January:608–615, 2018. 29

[69] Cao, Rongqiang, Shasha Lu, Xiaoning Wang, Haili Xiao e Xuebin Chi: *Unified account management for high performance computing as a service with microservice architecture*. PoS, 2018. `http://pos.sissa.it/`. 29, 33, 40

[70] Lu, Duo, Dijiang Huang, Andrew Walenstein e Deep Medhi: *A secure microservice framework for iot*. Proceedings - 11th IEEE International Symposium on Service-Oriented System Engineering, SOSE 2017, páginas 9–18, junho 2017. 29, 34, 35, 39

[71] Preuveneers, Davy e Wouter Joosen: *Access control with delegated authorization policy evaluation for data-driven microserviceworkflows*. Future Internet, 9, setembro 2017, ISSN 19995903. 29, 36, 38, 39, 40, 41

[72] He, Xiuyu e Xudong Yang: *Authentication and authorization of end user in microservice architecture*. Journal of Physics: Conference Series, 910, novembro 2017, ISSN 17426596. 29, 35, 38, 39, 40

[73] Torkura, Kennedy A., Muhammad I.H. Sukmana e Christoph Meinel: *Integrating continuous security assessments in microservices and cloud native applications*. UCC 2017 - Proceedings of the10th International Conference on Utility and Cloud Computing, páginas 171–180, dezembro 2017. 29, 34, 35, 39, 43

[74] Dooley, K.: *Designing Large Scale Lans: Help for Network Designers*. O'Reilly Media, 2001, ISBN 9780596551896. 36, 53

[75] Hardt, Dick: *Rfc 6749: The oauth 2.0 authorization framework*. Internet Engineering Task Force (IETF), 10:2070–1721, 2012. 38

[76] Foundation, OpenID: *How is openid connect different than openid 2.0?*, 2022. `https://openid.net/connect/`, acesso em 01 mar. 2022. 39, 40

[77] Rescorla, Eric: *Rfc 2818: Http over tls*. Internet Engineering Task Force (IETF), 2000. 40

[78] Sinnema, R e E Wilde: *Rfc 7061: extensible access control markup language (xacml) xml media type*. Internet Engineering Task Force (IETF), 2013. 41

[79] Krawczyk, H, M Bellare e R Canetti: *Mac: Keyed-hashing for message authenticatio*. Internet Engineering Task Force (IETF), 1997. 41

[80] Campbell, B, C Mortimore e M Jones: *Security assertion markup language (saml) 2.0 profile for oauth 2.0 client authentication and authorization grants*. Internet Engineering Task Force (IETF), 2015. 41

[81] Pivotal: *Spring cloud*, 2021. `https://spring.io/projects/spring-cloud`, acesso em 14 out. 2021. 42

[82] kit, Go: *Go kit - a toolkit for microservices*, 2022. `https://gokit.io/`, acesso em 02 mar. 2022. 42

[83] Projects, Pallets: *Flask web development*, 2022. `https://flask.palletsprojects.com/`, acesso em 02 mar. 2022. 42

[84] Microsoft: *.net core*, 2022. `https://dotnet.microsoft.com/`, acesso em 02 mar. 2022. 42

[85] Kong: *Kong api gateway*, 2021. https://konghq.com/kong, acesso em 14 out. 2021. 42

[86] Garousi, Vahid, Michael Felderer e Mika V. Mäntylä: *Guidelines for including grey literature and conducting multivocal literature reviews in software engineering*. Information and Software Technology, 106:101–121, 2019, ISSN 0950-5849. https://www.sciencedirect.com/science/article/pii/S0950584918301939. 44

[87] Christie, Marcus A, Anuj Bhandar, Supun Nakandala, Suresh Marru, Eroma Abeysinghe, Sudhakar Pamidighantam e Marlon E Pierce: *Using keycloak for gateway authentication and authorization*. Future Generation Computer Systems, 2017. 56

[88] Kong: *Istio*, 2022. https://www.redhat.com/pt-br/topics/microservices/what-is-istio, acesso em 06 jan. 2022. 56

[89] Spring: *Spring security overview*, 2022. https://spring.io/projects/spring-cloud-security, acesso em 06 jan. 2022. 58

[90] Hat, Red: *Keycloak*, 2022. https://www.keycloak.org/documentation, acesso em 06 jun. 2022. 59, 61

[91] Hat, Red: *Keycloak github*, 2022. https://github.com/keycloak/keycloak, acesso em 21 jan. 2022. 59

[92] Vigiolo, Mauricio: *Keycloak angular*, 2022. https://www.npmjs.com/package/keycloak-angular, acesso em 15 jul. 2022. 62

[93] Stianst: *Keycloak js*, 2022. https://www.npmjs.com/package/keycloak-js, acesso em 15 jul. 2022. 62

[94] SonarSource: *Sonarqube documentation*, 2022. https://docs.sonarqube.org/latest/, acesso em 14 jul. 2022. 65

[95] OWASP: *Owasp zap - getting started*, 2022. https://www.zaproxy.org/getting-started/, acesso em 15 jul. 2022. 65, 67

[96] CIRT.NET: *Nikto - overview & description*, 2022. https://github.com/sullo/nikto/wiki/Overview-&-Description, acesso em 17 jul. 2022. 65, 69

[97] Justiça, Ministério da: *Diretoria de operações*, 2022. https://www.gov.br/mj/pt-br/acesso-a-informacao/institucional/sumario/quemequem/diretoria-de-operacoes-diop, acesso em 17 jul. 2022. 65

[98] Santos, José Amancio M., João B. Rocha-Junior, Luciana Carla Lins Prates, Rogeres Santos do Nascimento, Mydiã Falcão Freitas e Manoel Gomes de Mendonça: *A systematic review on the code smell effect*. Journal of Systems and Software, 144:450–477, 2018, ISSN 0164-1212. https://www.sciencedirect.com/science/article/pii/S0164121218301444. 65

[99] OWASP: *Cross site scripting (reflected)*, 2022. https://www.zaproxy.org/docs/alerts/40012/, acesso em 15 jul. 2022. 69

[100] OWASP: *Buffer overflow*, 2022. https://www.zaproxy.org/docs/alerts/30001/, acesso em 15 jul. 2022. 69

[101] OWASP: *Format string error*, 2022. https://www.zaproxy.org/docs/alerts/30002/, acesso em 15 jul. 2022. 69

Apêndice A

Questionário aplicado

Protocolos de Autenticação e Autorização em Arquitetura de microsserviços

Olá, sou Murilo Góes de Almeida, mestrando em Computação Aplicada pela Universidade de Brasília (UnB), sob orientação da Profa. Dra. Edna Dias Canedo.

Estamos investigando quais são os principais desafios de segurança envolvendo autenticação e autorização em uma arquitetura de microsserviços. Dentro dos desafios encontrados, estudamos quais os principais protocolos que ajudam a mitigar ou resolver tais desafios, além de propor soluções open-source que implementam tais protocolos.

Desta feita, solicito a gentileza para responder o questionário, que tem como objetivo verificar se os desafios, protocolos e implementações *open source* são observados na indústria.

DECLARAÇÃO DE CONSENTIMENTO INFORMADO

Ao responder a esta pesquisa, você permite que o pesquisador obtenha, use e divulgue as informações anônimas fornecidas conforme descrito abaixo.

CONDIÇÕES E ESTIPULAÇÕES

1. Eu entendo que todas as informações são confidenciais. Não serei identificado pessoalmente. Concordo em preencher a pesquisa para fins de pesquisa e que os dados derivados desta pesquisa anônima podem ser publicados em periódicos, conferências e postagens de blog.

2. Eu entendo que minha participação nesta pesquisa é totalmente voluntária e que a recusa em participar não implicará em nenhuma penalidade ou perda de benefícios. Se eu quiser, posso cancelar minha participação a qualquer momento. Também entendo que, se decidir participar, posso recusar-me a responder a qualquer pergunta para a qual não me sinta confortável em responder.

3. Entendo que posso entrar em contato com o pesquisador se tiver alguma dúvida sobre a pesquisa. Estou ciente de que meu consentimento não me beneficiará diretamente. Também estou ciente de que o autor manterá os dados coletados em perpetuidade e poderá utilizar os dados para trabalhos acadêmicos futuros.

4. Ao clicar no botão abaixo, eu livremente dou consentimento e reconheço meus direitos como um participante voluntário da pesquisa, conforme descrito acima, e dou consentimento ao pesquisador para usar minhas informações na realização de pesquisas nas áreas mencionadas acima.

*** ESTE FORMULÁRIO LEVA CERCA DE 5 MINUTOS PARA SER PREENCHIDO.**

[**Q1.**] Qual o seu nível de formação acadêmica?

- Ensino Médio
- Estudante de Graduação
- Graduação
- Especialização

- Mestrado

- Doutorado

- Pós Doutorado

[**Q2.**] Há quanto tempo trabalha na área de TIC?

- Menos de 1 ano

- 1 a 3 anos

- 4 a 6 anos

- 7 a 10 anos

- 11 a 15 anos

- 16 a 20 anos

- mais de 21 anos

[**Q3.**] Qual é a área de atuação da organização em que você participa/-participou por mais tempo em um projetos de desenvolvimento de software?

- Administração pública

- Empresa privada de desenvolvimento de software

- Projetos de colaboração/pesquisa

- Projetos open-source

- Cursos técnicos

- Cursos acadêmicos

[**Q4.**] Qual é a sua função na organização?

- Analista de requisitos

- Projetista de dados

- Engenheiro de software

- Programador/Desenvolvedor de software

- Testador de Software

- Gerente de projeto

- Outro:

[**Q5.**] Você trabalha ou já trabalhou com projetos de TIC que implementa ou implementou arquitetura de microsserviços?

- Sim

- Não

[**Q6.**] Qual o seu nível de compreensão em relação aos conceitos de arquitetura de microsserviços?

- Escala Ordinal (1 a 5)

[**Q7.**] Quais desafios abaixo relacionados são/foram levados em consideração nos projetos de microsserviços que você atua/atuou?

- Comunicação entre microsserviços

- Confiança entre microsserviços comprometida por acesso não autorizado

- Preocupação individual para cada microsserviço

- Aumento na superfície de ataques (em comparação ao monolítico)

- Controle de acesso ao microsserviço

- Autorização entre serviços

- Poucos estudos a respeito

- Falta de patterns de segurança em microserivços

- Times diferentes trabalhando em microsserviços diferentes devem ter o mesmo entendimento em segurança

- Bypass no Api Gateway

- Detecção de intrusão/monitoramento

- Escalada de privilegios

- Falta de estudo demonstrando implementação prática de segurança em microsserviços Coordenar servidor de autenticação com novos microsserviços

- Falta de atenção em reação/recuperação de ataques

- Validação do token a cada requisição a microsserviço

- Imagens públicas podem estar comprometidas

- Muitas aplicações em microsserviços comerciais sem possibilidade de avaliar código

- Uso de servidor de autenticação/autorização que lide com todos os microsserviços

- Possibilidade de Desenvolvimento em várias tecnologias

- Outro:

[**Q8.**] Nos projetos de microsserviços em que você trabalha ou trabalhou, já ocorreram falhas de segurança envolvendo autenticação/autorização em microsserviço?

- Discordo totalmente

- Discordo

- Não concordo e nem discorto

- Concordo

- Concordo totalmente

[**Q9.**] Qual dos desafios da questão 7 colaboraram com a falha e Por que?

- Resposta Aberta

[**Q10.**] Quais dos protocolos abaixo são/foram implementados para mitigar os desafios identificados nos projetos em que você atua/atuou.

- OAuth 2.0

- JWT

- API Gateway
- OpenID Connect
- Single Sign-ON SSO
- HTTPS
- RBAC
- ABAC
- XACML
- HMAC
- SAML
- TLS
- OAuth
- Multilevel Security
- DAC
- IAM
- RSA
- SASL
- SSL
- API Keys
- Captcha
- CAS
- Certificados X509
- EAS
- ECDSA
- GSI
- IDS

- LDAP

- MTLS

- MTSL

- Multiplo FA

- NGAC

- PBAC

- Outro:

[**Q11.**] Nos projetos que você atua/atuou com arquitetura de microsserviços, houve a utilização de soluções open-source para implementar os protocolos elencados na questão anterior?

- Sim

- Não

[**Q12.**] Dentro dos projetos que você atua ou atuou, em arquitetura de microsserviços, caso tenha sido implementadas soluções open-source que implementem algum dos protocolos da questão anterior, quais foram utilizadas?

- Jarvis

- Kong

- Spring Cloud

- VertX

- Lagom

- Jadex

- Outro:

[**Q13.**] Você poderia descrever algum outro desafio que você já enfrentou ou enfrenta nos projetos de microsserviços que atua ou atuou?

- Resposta Aberta

[**Q14.**] Quais são as suas sugestões para mitigar os desafios de segurança ao se trabalhar com arquitetura de microsserviços?

- Resposta Aberta

[**Q15.**] Quais são as soluções open-source que você sugere para implementar a segurança em arquitetura de microsserviços?

- Resposta Aberta

Apêndice B

Descrição das principais classes utilizadas na camada de Segurança do Sistema

B.1 Classes utilizadas no microsserviço auth (Responsável pelo processo de autenticação)

B.1.1 WebSecurityConfig.class

```
1  @Configuration
2  @EnableWebSecurity
3  @EnableGlobalMethodSecurity(securedEnabled = true)
4  // A classe mais importante do Spring Security que extende
       WebSecurityConfigurerAdapter e realiza as configuracoes basicas no
        metodo configure
5  public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

6      @Autowired
7      private ADAuthenticationManager authProvider;
8
9      @Autowired
10     TokenAuthenticationService tokenAuthenticationService;
11
12 // metodo de configuracao do Spring Security, aqui pede para passar
       direto em alguns endpoints em especifico e determina que em todas
       as outras requisicoes o usuario deve estar autenticado.
```

```
13      @Override
14      protected void configure(HttpSecurity http) throws Exception {
15          http.csrf().disable()
16              .cors().and()
17               .authorizeRequests().antMatchers("/*", "/assets/**", "/
                    favicon.ico", "/construcao", "/contato").permitAll()
18              .antMatchers(HttpMethod.POST, "/login").permitAll()
19              .antMatchers(HttpMethod.GET, "/heart-check").permitAll()
20              .anyRequest().authenticated()
21              .and()
22  // filtra requisicoes de login
23              .addFilterBefore(new JWTLoginFilter("/login",
                    authenticationManager(), tokenAuthenticationService),
24              UsernamePasswordAuthenticationFilter.class)
25  // filtra outras requisicoes para verificar a presenca do jwt no
        header
26              .addFilterBefore(new JWTAuthenticationFilter(),
27                      UsernamePasswordAuthenticationFilter.class)
28              .sessionManagement().sessionCreationPolicy(
                    SessionCreationPolicy.STATELESS);
29      }
30  //o usuario e senha sera validado pela classe que configura a
        requisicao ao Active Directory
31      @Override
32      protected void configure(AuthenticationManagerBuilder auth)
            throws Exception {
33          auth.authenticationProvider(authProvider);
34      }
35  }
```

B.1.2 ADAuthenticationManager.class

```
36  @Slf4j
37  @Component
38  // Esta classe vai verificar no Active Directory se o usuario e senha
        estao validos e depois vamos verificar se ele tem autorizacao
        para usar o sistema.
39  public class ADAuthenticationManager implements
        AuthenticationProvider {
```

```
40      @Autowired
41      private KafkaService kafkaService;
42      @Autowired
43      UsuarioRepository usuarioRepository;
44      @Autowired
45      PerfilRepository perfilRepository;
46      @Autowired
47      private Tracer tracer;
48      @Value("${serverAD}")
49      private String serverAD;
50      @Value("${serverADName}")
51      private String serverADName;
52      static String perfil;
53
54      @Override
55      public Authentication authenticate(Authentication auth) throws
            AuthenticationException {
56          Span newSpan = tracer.nextSpan().name("Autenticacao Login");
57          try (Tracer.SpanInScope ws = this.tracer.withSpan(newSpan.
                start())) {
58              newSpan.tag("USER", auth.getName());
59
60  // ===================================================================
61  // pegando o username e senha digitado pelo usuario na hora do login
62  // ===================================================================
63              String username = auth.getName();
64              String password = auth.getCredentials().toString();
65              AuxAD ldapContxCrtn = new AuxAD();
66
67  // ===================================================================
68  // tentando se conectar no AD
69  // ===================================================================
70              log.info("Verificando a existencia do usuario " +
                    username + " na Base de usuarios da PMESP");
71              newSpan.event("Chamando autenticacao do AD");
72              RetornoAD retornoAD = ldapContxCrtn.validaUsuarioAD(
                    serverAD, username, password, serverADName);
73
74  // se for sucesso, vamos procurar o usuario dentro da propria
    aplicacao
```

```
75              if (retornoAD.getMensagem() == AuxAD.SUCESSO) {
76                  log.info("O usuario " + username + " e existente na
                        Base de usuarios da PMESP");
77                  newSpan.event("usuario encontrado na base AD");
78  // ================================================================
79  // aqui estou vendo se o CPF que autenticou, esta autorizado a usar a
        aplicacao
80  // ================================================================
81                  log.info("Verificando a existencia do usuario " +
                        username + " na Base de usuarios do INTWEB");
82                  newSpan.event("Chamando autenticacao INTWEB");
83                  Usuario usuario = usuarioRepository.findByUserName(
                        username);
84
85  // ================================================================
86  // se deu nulo, quer dizer que ele nao esta na nossa base (mesmo que
        tenha uma
87  // conta valida no AD)
88  // ================================================================
89                  if (usuario == null) {
90                      log.info("O usuario " + username + " nao esta
                            existente na Base de usuarios do INTWEB");
91                      newSpan.event("Usuario nao encontrado na base
                            INTWEB");
92
93  // ================================================================
94  // ENVIANDO LOG AO KAFKA
95  // ================================================================
96                      LogModel logModel = new LogModel();
97                      logModel.setIdAcao(Acoes.TENTATIVA_LOGIN.ID());
98                      logModel.setDetalhes("Tentativa frustrada de
                            login do usuario " + username
99                                  + ", o qual nao foi localizado na Base de
                                        usuarios do INTWEB");
100                     kafkaService.sendLog(logModel, username);
101                     throw new
                            AuthenticationCredentialsNotFoundException("
                            usuario nao cadastrado.");
102                 } else {
```

```
103                     newSpan.event("usuario encontrado na base INTWEB"
                            );
104                     log.info("O usuario " + username
105                             + " foi localizado na Base de usuarios do
                                    INTWEB. Prosseguindo no login.");
106
107 // ================================================================
108 // agora estou vendo quais permissoes esse usuario tem
109 // ================================================================
110                     List<Permissao> listaPermissoes = usuario.
                            getPerfil().getPermissoes();
111
112 // ================================================================
113 // adicionando as permissoes para o usuario
114 // ================================================================
115                     List<GrantedAuthority> grantedAuths = new
                            ArrayList<>();
116
117                     for (Permissao perm : listaPermissoes)
118                         grantedAuths.add(new SimpleGrantedAuthority(
                                perm.getPermissao()));
119                     return new UsernamePasswordAuthenticationToken(
                            username, password, grantedAuths);
120                 }
121             } else {
122                 log.info("Tentativa frustrada de login do usuario " +
                        username
123                         + " na Base de usuarios da PMESP. Detalhes do
                                erro : " + retornoAD.getMensagem());
124                 newSpan.event("Falha no login AD");
125                 newSpan.tag("AD_ERROR", retornoAD.getMensagem());
126
127 // ================================================================
128 // ENVIANDO LOG AO KAFKA
129 // ================================================================
130                 LogModel logModel = new LogModel();
131                 logModel.setIdAcao(Acoes.TENTATIVA_LOGIN.ID());
132                 logModel.setDetalhes("Tentativa frustrada de de login
                        do usuario " + username
```

```
133                         + " na Base de usuarios da PMESP. Detalhes do
                                erro : " + retornoAD.getMensagem());
134                 kafkaService.sendLog(logModel, username);
135                 throw new AuthenticationCredentialsNotFoundException(
                       retornoAD.getMensagem());
136             }
137         } catch (Exception e) {
138             newSpan.tag("error-main", e.getMessage());
139             log.info("Excecao lancada durante a opearacao principal:
                   " +  e.getMessage());
140             return null;
141         } finally {
142             newSpan.end();
143         }
144     }
145
146     @Override
147     public boolean supports(Class<?> authentication) {
148         return authentication.equals(
               UsernamePasswordAuthenticationToken.class);
149     }
150 }
```

B.1.3 TokenAuthenticationService.class

```
152 @Slf4j
153 @Component
154 // Caso o AD aprovar o login e senha, esta classe vai checar se o
       policial possui permissao para usar o sistema, buscar os dados
       basicos do policial e caso ele tiver permissao, sera montado o
       token para devolver ao cliente
155 public class TokenAuthenticationService {
156     @Autowired
157     private Tracer tracer;
158
159     @Autowired
160     private KafkaService kafkaService;
161
162     @Autowired
```

```
163     PolicialMilitarService consultaPM;
164
165     static long EXPIRATION_TIME;
166     static String SECRET;
167     final static String TOKEN_PREFIX = "Bearer";
168     final static String HEADER = "Authorization";
169
170     @Value("${token.secret}")
171     public void setSecret(String secret) {
172         SECRET = secret;
173     }
174
175     @Value("${token.expiration}")
176     public void setExpiration_time(long expiration) {
177         EXPIRATION_TIME = expiration;
178     }
179
180 // ================================================================
181 // neste metodo vamos montar o token para ser retornado ao front-end,
        colocando as permissoes e dados basicos do usuario
182 // ================================================================
183     void addAuthentication(HttpServletResponse response,
           Authentication auth) {
184         Span newSpan = tracer.nextSpan().name("Criacao do JWT");
185         try (Tracer.SpanInScope ws = this.tracer.withSpan(newSpan.
               start())) {
186             newSpan.event("Chamando consulta de PM por CPF");
187             SaidaPmSimplificadoCipmWs pm = consultaPM.procuraPmPorCpf
                   (auth.getName());
188             newSpan.event("Iniciando a criacao do JWT");
189             Claims claims = Jwts.claims().setSubject(auth.getName());
190             claims.put("scopes", auth.getAuthorities().stream().map(s
                    -> s.toString()).collect(Collectors.toList()));
191             claims.put("opm", pm.getOpmCod());
192             claims.put("opmDes", FormatHelper.arrumarDescricaoOPM(pm.
                   getOpmDes()));
193             claims.put("posto", FormatHelper.
                   arrumarDescricaoPostoGrad(pm.getPosto()));
194             claims.put("nome", pm.getNome());
195
```

```
196 // ================================================================
197 // Montando o token de fato com os dados colhidos na aplicacao
198 // ================================================================
199             String JWT = Jwts.builder().setClaims(claims).
                   setExpiration(new Date(System.currentTimeMillis() +
                   EXPIRATION_TIME)).signWith(HS512, SECRET).compact();

200             response.addHeader(HEADER, TOKEN_PREFIX + " " + JWT);
201             response.addHeader("Content-Type", "application/json");
202
203 //================================================================
204 // Montando o objeto com os dados colhidos na aplicacao para
       utilizacao do Frontend
205 //================================================================

206             response.getOutputStream().write((
207                             "{\"username\":\""
208                             + auth.getName()
209                             + "\",\"token\":\""
210                             + JWT
211                             + "\",\"nome\":\""
212                             + pm.getNome()
213                             + "\",\"re\":\""
214                             + pm.getRe()
215                             + "\",\"opmDes\":\""
216                             + FormatHelper.arrumarDescricaoOPM(pm.
                                   getOpmDes())
217                             + "\",\"foto\":\""
218                             + pm.getFoto()
219                             + "\",\"opm\":\""
220                             + pm.getOpmCod()
221                             + "\",\"posto\":\""
222                             + FormatHelper.arrumarDescricaoPostoGrad(
                                   pm.getPosto())

223                             + "\"}").getBytes());
224 // ================================================================
225 // ENVIANDO MENSAGEM DE LOG
226 // ================================================================
227             log.info("Realizando autenticacao de usuario");
```

```
228             LogModel logModel = new LogModel();
229             logModel.setIdAcao(Acoes.LOGIN.ID());
230             logModel.setUsuario(auth.getName());
231             logModel.setDetalhes(
232                     "Usuario "
233                     + pm.getPosto() + " "
234                     + pm.getNome() + " do "
235                     + pm.getOpmDes()
236                     + " autenticado com sucesso.");
237             kafkaService.sendLog(logModel, auth.getName());
238 
239         } catch (Exception e) {
240             newSpan.tag("JWT_CREATE", e.getMessage());
241             try {
242                 Claims claims = Jwts.claims().setSubject(auth.getName
                        ());
243                 claims.put("scopes", auth.getAuthorities().stream().
                        map(s -> s.toString()).collect(Collectors.toList()
                        ));
244                 String JWT = Jwts.builder().setClaims(claims).
                        setExpiration(new Date(System.currentTimeMillis()
                        + EXPIRATION_TIME)).signWith(HS512, SECRET).
                        compact();
245                 response.addHeader(HEADER, TOKEN_PREFIX + " " + JWT);
246                 response.addHeader("Content-Type", "application/json"
                        );
247 
248                 response.getOutputStream().write((
249                         "{\"username\":\""
250                         + auth.getName()
251                         + "\",\"token\":\""
252                         + JWT + "\",\"nome\":\""
253                         + "Erro ao buscar informacao"
254                         + "\",\"foto\":\""
255                         + ""
256                         + "\",\"opm\":\""
257                         + "Erro ao buscar informacao"
258                         + "\",\"posto\":\"Erro ao buscar informacao"
259                         + "\"}").getBytes());
260 
```

```
261 //=====================================================================
262 // ENVIANDO MENSAGEM DE LOG
263 //=====================================================================
264                 log.info("Realizando autenticacao de usuario - Fluxo
                        com excecao");
265                 LogModel logModel = new LogModel();
266                 logModel.setIdAcao(Acoes.LOGIN.ID());
267                 logModel.setUsuario(auth.getName());
268                 logModel.setDetalhes("Usuario " + logModel.getUsuario
                        () + " autenticado com sucesso.");
269                 kafkaService.sendLog(logModel, auth.getName());
270             } catch (Exception e1) {
271                 newSpan.tag("JWT_CREATE_WITH_EXCEPTION", e.getMessage
                        ());
272                 log.info("Excecao lancada durante a operacao", e1.
                        getMessage());
273             } finally {
274                 newSpan.end();
275             }
276         }
277     }
278
279     @SuppressWarnings("unchecked")
280     public static Authentication getAuthentication(HttpServletRequest
            request) {
281         String token = request.getHeader(HEADER);
282         if (token != null) {
283 // ====================================================================
284 // faz parse do token
285 // ====================================================================
286             String user = null;
287             ArrayList<String> permissoes = null;
288             ArrayList<SimpleGrantedAuthority> autorities = new
                    ArrayList<SimpleGrantedAuthority>();
289             try {
290                 user = Jwts.parser().setSigningKey(SECRET).
                        parseClaimsJws(token.replace(TOKEN_PREFIX, "")).
                        getBody().getSubject();
```

```
291                 permissoes = Jwts.parser().setSigningKey(SECRET).
                        parseClaimsJws(token.replace(TOKEN_PREFIX, "")).
                        getBody().get("scopes", ArrayList.class);
292             } catch (Exception e) {
293                 log.info(e.getMessage());
294                 return null;
295             }
296             for (String str : permissoes)
297                 autorities.add(new SimpleGrantedAuthority(str));
298 
299             if (user != null) {
300                 return new UsernamePasswordAuthenticationToken(user,
                        null, autorities);
301             } else
302                 return null;
303         }
304         return null;
305     }
306 }
```

B.1.4 JWTLoginFilter.class

```
308 // Classe responsavel por filtrar as requisicoes de login
309 public class JWTLoginFilter extends
       AbstractAuthenticationProcessingFilter {
310 
311         TokenAuthenticationService tokenAuthenticationService;
312 
313          protected JWTLoginFilter(String url, AuthenticationManager
               authManager, TokenAuthenticationService
               _tokenAuthenticationService) {
314              super(new AntPathRequestMatcher(url));
315              setAuthenticationManager(authManager);
316              this.tokenAuthenticationService =
                    _tokenAuthenticationService;
317         }
318 
319         @Override
```

```
320         public Authentication attemptAuthentication(
               HttpServletRequest request, HttpServletResponse response)
321                 throws AuthenticationException, IOException,
                       ServletException {
322             if (!HttpMethod.POST.name().equals(request.getMethod()) )
                    {
323                 if(logger.isDebugEnabled()) {
324                     logger.debug("Authentication method not supported
                           . Request method: " + request.getMethod());
325                 }
326                 throw new AuthenticationServiceException("Metodo nao
                       suportado");
327             }
328             AccountCredentials credentials;
329             try {
330                 credentials = new ObjectMapper()
331                         .readValue(request.getInputStream(),
                               AccountCredentials.class);
332             }
333             catch (Exception e){
334                 credentials = new AccountCredentials("","");
335             }
336             return getAuthenticationManager().authenticate(
337                     new UsernamePasswordAuthenticationToken(
338                             credentials.getUsername(),
339                             credentials.getPassword(),
340                             Collections.emptyList()
341                     )
342             );
343         }
344
345         @Override
346         protected void successfulAuthentication(
347                 HttpServletRequest request,
348                 HttpServletResponse response,
349                 FilterChain filterChain,
350                 Authentication auth) throws IOException,
                       ServletException {
351                 tokenAuthenticationService.addAuthentication(response
                       , auth);
```

```
352         }
353 }
```

B.1.5 JWTAuthenticationFilter.class

```
355 // Filtra as requsicoes que ja contenha um jwt e adiciona ao contexto
        de seguranca para uso durante a requisicao
356 public class JWTAuthenticationFilter extends GenericFilterBean {
357     @Override
358     public void doFilter(ServletRequest request, ServletResponse
            response, FilterChain filterChain)
359             throws IOException, ServletException {
360         Authentication authentication = TokenAuthenticationService.
                getAuthentication((HttpServletRequest) request);
361         SecurityContextHolder.getContext().setAuthentication(
                authentication);
362         filterChain.doFilter(request, response);
363     }
364 }
```

B.2 Classes utilizadas no microsserviço Gateway para checagem de autorização

B.2.1 WebSecurityConfig.class

```
365
366 @Configuration
367 @EnableWebSecurity
368 @EnableGlobalMethodSecurity(securedEnabled = true)
369 public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
370     @Autowired
371     JWTAuthenticationFilter jWTAuthenticationFilter;
372
373     @Value("${list.sites}")
374     private String sites;
375
376     @Override
```

```
377     protected void configure(HttpSecurity http) throws Exception {
378         http.csrf().disable()
379             .cors().and()
380             .authorizeRequests()
381             .antMatchers(HttpMethod.POST, "/auth/login").permitAll()
382             .antMatchers(HttpMethod.GET, "/**/heart-check").permitAll
                    ()
383             .anyRequest().authenticated()
384             .and()
385 
386             // filtra outras requisicoes para verificar a presenca do
                    jwt no header
387             .addFilterBefore(jWTAuthenticationFilter,
388                     UsernamePasswordAuthenticationFilter.class) //;
389             .sessionManagement().sessionCreationPolicy(
                    SessionCreationPolicy.STATELESS);
390     }
391 
392     @Bean
393     CorsConfigurationSource corsConfigurationSource() {
394         CorsConfiguration configuration = new CorsConfiguration();
395         configuration.setAllowedOrigins(Arrays.asList(sites));
396         configuration.setAllowedMethods(Arrays.asList("GET", "PUT", "
                POST","OPTIONS", "DELETE"));
397         configuration.setAllowedHeaders(Arrays.asList("authorization"
                ,"content-type"));
398         UrlBasedCorsConfigurationSource source = new
                UrlBasedCorsConfigurationSource();
399         source.registerCorsConfiguration("/**", configuration);
400         return source;
401     }
402 }
```

B.2.2 JWTAuthenticationFilter.class

```
403 @Component
404 public class JWTAuthenticationFilter extends GenericFilterBean {
405 
406     @Value("${token.secret}")
```

```
407      private String SECRET;
408
409      private String TOKEN_PREFIX = "Bearer";
410
411      private String HEADER_STRING = "Authorization";
412
413     @Override
414     public void doFilter(ServletRequest request, ServletResponse
            response, FilterChain filterChain)
415              throws IOException, ServletException {
416
417
418          Authentication authentication = getAuthentication((
                HttpServletRequest) request);
419
420          SecurityContextHolder.getContext().setAuthentication(
                authentication);
421          filterChain.doFilter(request, response);
422     }
423
424     @SuppressWarnings("unchecked")
425     public Authentication getAuthentication(HttpServletRequest
            request) {
426
427          String token = request.getHeader(HEADER_STRING);
428
429
430           if (token != null) {
431               // faz parse do token
432               String user = null;
433               ArrayList<String> permissoes = null;
434               ArrayList<SimpleGrantedAuthority> autorities = new
                     ArrayList<SimpleGrantedAuthority>();
435               try {
436                    user = Jwts.parser()
437                            .setSigningKey(SECRET)
438                            .parseClaimsJws(token.replace(
                                 TOKEN_PREFIX, ""))
439                            .getBody()
440                            .getSubject();
```

```
441                         permissoes = Jwts.parser()
442                                 .setSigningKey(SECRET)
443                                 .parseClaimsJws(token.replace(
                                        TOKEN_PREFIX, ""))
444                                 .getBody()
445                                 .get("scopes", ArrayList.class);
446                     } catch (Exception e) {
447                         return null;
448                     }
449                     for(String str : permissoes){
450                         autorities.add(new SimpleGrantedAuthority(str));
451                     }
452 
453                     if (user != null) {
454                         return new UsernamePasswordAuthenticationToken(
                                    user, null, autorities);
455                     }
456                     else
457                     {
458                         return null;
459                     }
460                 }
461                 return null;
462     }
463 
464 }
```

Apêndice C

Descrição das principais classes utilizadas na nova Camada de Segurança

C.1 Classes utilizadas no novo serviço de Gateway

C.1.1 application.properties

Por questões de segurança, os endereço IPs e portas foram suprimidos.

```
466
467 spring:
468   application:
469     name: gateway
470   security:
471     oauth2:
472       resourceserver:
473         jwt:
474           issuer-uri: http://0.0.0.0:0000/auth/realms/intweb
475           jwk-set-uri: http://0.0.0.0:0000/auth/realms/intweb/
                protocol/openid-connect/certs
476   cloud:
477     gateway:
478       discovery:
479         locator:
480           lower-case-service-id: true
481           enabled: true
```

```
482
483 server:
484   port: 8181
485
486 eureka:
487   client:
488     should-unregister-on-shutdown: true
489     serviceUrl:
490       defaultZone: http://0.0.0.0:0000/eureka/
```

C.1.2 WebSecurityConfig

```
493
494 @EnableWebFluxSecurity
495 @EnableReactiveMethodSecurity
496 public class WebSecurityConfig {
497
498     @Bean
499     SecurityWebFilterChain springSecurityFilterChain(
        ServerHttpSecurity http) throws Exception {
500          http.authorizeExchange().anyExchange().authenticated().and().
                oauth2ResourceServer().jwt();
501          return http.build();
502     }
503
504 }
```

Apêndice D

Descrição das principais classes utilizadas na nova Camada de Segurança aplicado ao *front-end*

D.1 Trechos da implementação do Keycloak no *front-end*

Por questões de segurança, algumas informações dessas classes foram suprimidas.

D.1.1 shared.module.ts

A classe shared.module.ts importa os módulos das bibliotecas necessárias para o desenvolvimento do projeto. Esta classe também possui o método initializeKeycloak que foi adicionado ao atributo providers. Isso fará com que a biblioteca do keycloak seja inicializada antes de qualquer carregamento da página, permitindo assim uma verificação prévia de segurança.

```
509
510 // por razoes de seguranca, foram suprimidos os outros imports e
       configuracoes
511
512 @NgModule({
513   imports: [
514     KeycloakAngularModule
```

```
515     ],
516     exports: [
517       KeycloakAngularModule
518     ],
519     providers: [
520       {
521         provide: APP_INITIALIZER,
522         useFactory: initializeKeycloak,
523         multi: true,
524         deps: [KeycloakService],
525       }
526     ],
527 })
528
529 export class SharedModule { }
530
531 // ESSA FUNCAO VAI SER CHAMADA NO PROVIDERS, A APLICACAO SO VAI
       INICIAR ANTES DE INICIAR O KEYCLOAK PARA FAZER TODAS VERIFICACOES
       DE SEGURANCA
532  function initializeKeycloak(keycloak: KeycloakService) {
533   return async () =>
534   await keycloak.init({
535       config: {
536         url: environment.keycloakUrlBase + '/auth',
537         realm: environment.realm,
538         clientId: environment.clientId,
539       },
540       initOptions: {
541         onLoad: 'login-required'
542       },
543       loadUserProfileAtStartUp: true
544     });
545 }
```

D.1.2 auth.guard.ts

A classe auth.guard.ts verifica em cada requisição HTTP feita pelo *front-end*, se o usuário está autenticado, se o token é válido e se ele possui a "role"

requirida para a requisição. As checagens são feitas automaticamente pela biblioteca keycloak-angular

```
547
548 import { Injectable } from '@angular/core';
549 import {
550   ActivatedRouteSnapshot,
551   Router,
552   RouterStateSnapshot
553 } from '@angular/router';
554 import { KeycloakAuthGuard, KeycloakService } from 'keycloak-angular'
      ;
555
556 @Injectable({
557   providedIn: 'root'
558 })
559 export class AuthGuard extends KeycloakAuthGuard {
560   constructor(
561     protected readonly router: Router,
562     protected readonly keycloak: KeycloakService) {
563     super(router, keycloak);
564   }
565
566   public async isAccessAllowed(
567     route: ActivatedRouteSnapshot,
568     state: RouterStateSnapshot) {
569
570     // ================================================
571     // SE NAO ESTIVER AUTENTICADO FORCAMOS O LOGIN
572     // ================================================
573     if (!this.authenticated) {
574       await this.keycloak.login({
575         redirectUri: window.location.origin + state.url
576       });
577       return false
578     }
579     else {
580     // ================================================
581     // OBTENDO AS ROLES REQUERIDAS DA ROTA
582     // ================================================
```

```
583       const requiredRoles: string[] = route.data.expectedRole;
584         return this.checkRoles(requiredRoles)
585       }
586     }
587
588     // ======================================================
589     // VERIFICANDO SE O USUARIO POSSUI A ROLE REQUERIDA
590     // ======================================================
591     checkRoles(roles: string[]) {
592       // ======================================================
593       // SE NAO EXISTIR ROLE REQUERIDA IGNORE ESSA OPERACAO
594       // ======================================================
595       if (!(roles instanceof Array) || roles.length === 0) return true;
596       else {
597         let passed = false;
598         this.roles.forEach((roleUsuario: string) => {
599           roles.forEach((roleRequerida) => {
600             if (roleRequerida == roleUsuario)
601               passed = true;
602             else console.log("Sem permissao !")
603           });
604         });
605         return passed;
606       }
607     }
608 }
```