Simplificando a adoção de ZTA com Motor de Políticas e Decisões Baseadas em *Logs*

Abraão Freitas[1], **Davi Guimarães**[1], **Andrey Brito**[1]

[1]Centro de Engenharia Elétrica e Informática (CEEI)
Universidade Federal de Campina Grande (UFCG)

{abraao.freitas,davi.guimaraes}@ccc.ufcg.edu.br,

andrey@computacao.ufcg.edu.br

Abstract. *The microservices architecture and the growing adoption of cloud service providers have introduced new cybersecurity challenges. Dynamic network perimeters, expanded attack surfaces, and complex communications invalidate traditional models based on static perimeters. This paper presents two access control approaches as starting points for adopting Zero Trust: one based on a policy engine with multifactor quotas and another leveraging log analysis and alerts to define penalties. Adding intelligence to standard proxies with the proposed architectures enables the Zero Trust adoption maturity level to be increased from base or intermediary to advanced in several aspects. They were also evaluated in simulated environments and the results demonstrate effectiveness in identifying irregularities and enhancing system security, blocking attacks in as little time as 4 seconds, as well as feasibility in terms of latency and ease of adoption.*

Resumo. *A arquitetura de microsserviços e a popularização de provedores de nuvem como serviço introduziram desafios de cibersegurança. Perímetros de rede dinâmicos, superfícies de ataque ampliadas e comunicações complexas invalidam modelos tradicionais baseados em perímetros estáticos. Este trabalho apresenta duas abordagens de controle de acesso como pontos de partida para a adoção de Zero Trust: uma baseada em motor de políticas com quotas multifator e outra em análise de logs e alertas para definir penalidades. A simples adição de inteligência a um proxy padrão com as arquiteturas propostas permite elevar o nível de maturidade na adoção de Zero Trust de básico ou intermediário para avançado em vários aspectos. As abordagens foram avaliadas em ambientes simulados, e os resultados demonstram eficácia na identificação de irregularidades e no aumento da segurança, bloqueando ataques em até 4 segundos, além da viabilidade em termos de latência e facilidade de adoção.*

1. Introdução

A crescente adoção de arquitetura de microsserviços, VPNs e computação em nuvem tem transformado o mercado e a infraestrutura de organizações [MarkWide 2025]. Essa evolução foi impulsionada pelo aumento do trabalho remoto, a necessidade de acesso contínuo a recursos distribuídos e a popularização de provedores de infraestrutura, plataforma e *software* como serviço (IaaS, PaaS, SaaS), como *Amazon Web Services* (AWS),

Vercel e Azure. Anteriormente, o sistema era mantido em grandes e bem protegidos *data centers*. Agora, os componentes do sistema estão distribuídos entre diferentes *clusters* mantidos por empresas terceiras [Xiao et al. 2019] e o tráfego de dados entre os serviços de uma aplicação não está mais restrito a redes internas.

Modelos tradicionais de segurança, baseados em perímetros estáticos, não conseguem atender às demandas de sistemas distribuídos modernos, tornando-se alvo de preocupação em [Athena and Sumathy 2017], haja vista o aumento na complexidade em relação à interdependência de ambientes, à maior permeabilidade e à superfície de ataque estendida. Nestes sistemas, não é trivial definir o perímetro de segurança quando os serviços estão distribuídos entre diversos provedores e, mesmo sendo feito, uma vez dentro da rede interna, o atacante pode mover lateralmente, minerar credenciais e subir seu nível de acesso [Feldman et al. 2020]. Além disso, métodos como a *Public Key Infrastructure* (PKI), baseados em credenciais estáticas, enfrentam desafios significativos, como altos custos de manutenção, falta de escalabilidade e vulnerabilidade a violações catastróficas. Em 2024, estima-se que o impacto médio de uma violação de dados causada por falhas em credenciais tenha alcançado 4,88 milhões de dólares, com um ciclo médio de 292 dias para detecção e contenção [IBM 2024].

A partir da necessidade por modelos de segurança mais sofisticados e dinâmicos que suportem a crescente complexidade dos sistemas distribuídos, o paradigma *Zero Trust* (ZT) ganha destaque. ZT é um conjunto de princípios e ideias que minimizam a incerteza através de políticas de acesso mínimas [Rose et al. 2020]. No modelo de *Zero Trust Architecture* (ZTA), é assumido que as ameaças podem estar dentro ou fora do sistema e, por isso, a confiança não é concedida, mas continuamente avaliada ao longo do tempo [Rose et al. 2020]. A adoção de uma ZTA é gradual e envolve diversas etapas, onde tecnologias como autenticação multifator, segmentação de redes e monitoramento contínuo são implementadas progressivamente e uma cultura ZT é criada [DoD 2022].

Desse modo, considerando os diferentes níveis de maturidade de uma ZTA, este artigo propõe abordagens para simplificar a adoção de práticas mais avançadas de segurança, oferecendo um ponto de partida vantajoso para as organizações em estados iniciais de maturidade. O trabalho inclui duas abordagens não excludentes: uma que exige menor compatibilidade e configurações no sistema existente, baseada em motor de políticas para decisão de acesso, a qual previne a degradação do sistema a priori; enquanto a segunda utiliza processamento de *logs* para elevar a inteligência das verificações e permitir uma degradação mais graciosa. As contribuições incluem a avaliação de ferramentas comumente usadas na implementação de ZTA para aplicar controle de acesso multifator e na proposição de abordagens para migração gradual de sistemas tradicionais para um modelo de segurança ZT com nível de maturidade alto.

A estrutura deste trabalho está organizada da seguinte forma. Na Seção 2, são apresentadas as bases teóricas sobre o conceito de ZTA, incluindo seus princípios fundamentais, níveis de maturidade e as ferramentas utilizadas. Na Seção 3, são abordados os trabalhos relacionados ao controle de acesso no contexto de ZTA, destacando suas limitações e as convergências com o presente estudo. Na Seção 4, são apresentadas e detalhadas as abordagens propostas, explicando sua arquitetura, funcionamento e os componentes envolvidos. Na Seção 5, são apresentados os métodos de avaliação e os resultados obtidos. Finalmente, na Seção 6, são discutidas as conclusões, com ênfase nas principais

contribuições do trabalho e nas sugestões para futuras pesquisas.

2. Fundamentação Teórica

Nesta seção, são introduzidos conceitos e tecnologias importantes para entender o contexto do trabalho apresentado.

2.1. Zero Trust

Zero Trust é um modelo de cibersegurança que parte da premissa de que não existe confiança inata [Rose et al. 2020]. No modelo, todo acesso deve ser verificado, independentemente de estar dentro ou fora de um perímetro. Para diminuir as incertezas, o foco deve estar voltado na autenticação, na autorização e na redução de confianças implícitas. Na Figura 1, é apresentada a arquitetura de referência de ZTA.

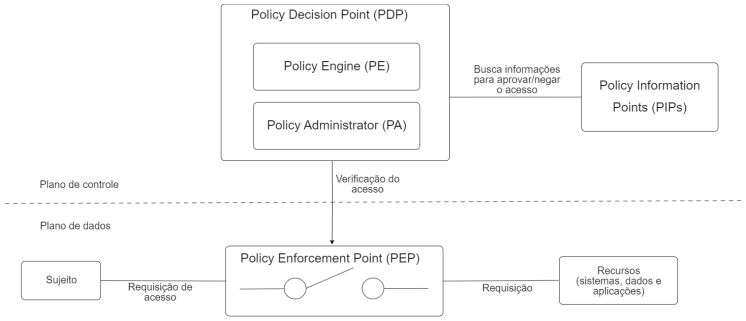

Figura 1. Arquitetura de referência de ZTA (adaptada de [Borchert et al. 2024])

A arquitetura consiste em dois componentes fundamentais trabalhando em conjunto para garantir que a decisão tomada seja a mais segura para o sistema. Estes componentes são o *Policy Enforcement Point* (PEP) e o *Policy Decision Point* (PDP). No PDP, a *Policy Engine* (PE) concede ou nega o pedido de acesso baseado no nível de confiabilidade do sujeito a partir das políticas da organização e de informações fornecidas pelos componentes de suporte, os *Policy Information Points* (PIPs). Já o *Policy Administrator* (PA) é responsável por estabelecer a comunicação entre o sujeito e o recurso através de comandos enviados ao PEP. Além disso, o PA gera credenciais de autenticação para aquela sessão. Em alguns sistemas, as funções de PE e PA são implementadas pelo mesmo componente. Em relação ao PEP, este habilita, monitora e termina a conexão entre o sujeito e o recurso; e encaminha as requisições de acesso para verificação no PDP.

2.2. Níveis de Maturidade

Transicionar para uma ZTA exige mudanças significativas em diversos aspectos do sistema, como gerenciamento de identidades, arquitetura da rede, políticas de acesso e monitoramento dos componentes do sistema. Em razão disto, são definidos níveis de maturidade que representam o grau de concordância com o modelo ZT [DoD 2022]. Esta abordagem permite que as organizações priorizem vulnerabilidades críticas e evoluam gradualmente seu modelo de ZTA. Cada nível será descrito posteriormente.

- **Descoberta e Avaliação** é o nível de preparação para a ZTA, onde as relações entre as entidades do sistema são identificadas e avaliadas. Nesta etapa, os componentes críticos do sistema são identificados, fluxos de dados são mapeados, e

atividades de acesso e autorização são analisadas. Esse diagnóstico é essencial para compreender os pontos de vulnerabilidade e estabelecer as prioridades.

- **Linha base** é o nível em que as bases dos controles de ZT são implementadas no sistema existente, com base nas informações coletadas na fase de Descoberta e Avaliação. As principais atividades incluem segmentação da rede com políticas de privilégio mínimo, classificação e encriptação de dados críticos, implementação de autenticação multifator (MFA) e a garantia de conformidade dos dispositivos com as políticas de segurança organizacional.
- **Intermediário** é o nível posterior à Linha Base, onde os princípios de ZT são expandidos para abranger todos os segmentos da organização. As políticas definidas no nível anterior são refinadas e passam a incluir mais atributos contextuais, como o dispositivo, o horário, a localização e a frequência de acesso. A partir desse ponto, mecanismos de análise comportamental começam a ser implementados para identificar anomalias e ajustar as políticas de acesso de forma proativa.
- **Avançado** é o nível com maior maturidade, com mecanismos dinâmicos de acesso em tempo real. Todos os acessos são mínimos, monitorados e avaliados com base em políticas dinâmicas ajustadas por análises robustas dos dados de monitoramento. Nesse estágio, a identificação e a autorização são realizadas de forma contínua ao longo das sessões, e a análise de ameaças é feita continuamente com alertas para que a resposta a incidentes seja rápida.

2.3. Autorização e Controle de Acesso

No fluxo de transação online envolvendo a requisição de um sujeito a um objeto/operação, a autorização é a etapa responsável por indicar quais são os recursos acessíveis por esse sujeito identificado. Com as políticas de autorização, são definidos os níveis de acesso dos indivíduos ou dispositivos. Essas políticas controlam o acesso e podem ser dinâmicas ou estáticas, a depender do modelo de controle de acesso utilizado. Dentre os modelos de controle de acesso, serão discutidos dois dos principais: RBAC e ABAC.

No modelo de controle de acesso baseado em papéis (RBAC), os usuários assumem papéis que são utilizados para determinar o nível de acesso. As permissões são atribuídas aos papéis e não aos usuários, o que dissocia a administração das políticas de controle de acesso e a administração dos usuários do sistema. Desse modo, o RBAC simplifica a definição das políticas de autorização ao agrupar entidades com o mesmo nível de acesso/papel. Entretanto, a dificuldade para definir os papéis e suas respectivas permissões, também denominada de "engenharia de papel", limita esse modelo. Da mesma forma que o objetivo é reduzir a quantidade de papéis, também objetiva-se definir políticas de controle mais restritas, o que eventualmente leva à definição de mais papéis e mais permissões, gerando conflito [de Mello et al. 2022].

Diante de tais limitações, existem variações e evoluções do RBAC, com destaque para o controle de acesso baseado em atributos (ABAC) [NIST 2012]. Nesse modelo, as permissões são baseadas em um conjunto de regras relacionadas a atributos das entidades envolvidas. A cada requisição, os atributos do sujeito, do objeto e do ambiente, como horário e localização, são verificados e, a depender das políticas definidas, o acesso é concedido ou não. Por levar em conta uma quantidade mais ampla de variáveis, é possível a criação de políticas mais flexíveis e dinâmicas [Mhetre et al. 2022]. Além disso, ao considerar papéis como atributos, o ABAC implementa o RBAC.

2.4. Motor de políticas

O *Open Policy Agent* (OPA)[1] é um motor de políticas graduado pela *Cloud Native Computing Foundation* (CNCF)[2], que concede essa certificação a projetos amplamente adotados em sistemas em produção e com comunidades consolidadas. A ferramenta se destaca pela centralização das políticas de segurança de toda a pilha de infraestrutura no formato *Policy as Code* (PaC) [Ferreira 2022], o que aumenta a transparência, além de simplificar a integração com diferentes sistemas. As políticas do OPA, escritas na linguagem nativa e declarativa Rego[3], tomam decisões com base em dados estruturados hierarquicamente, denominados documentos, com um conjunto de atributos referentes ao contexto de execução. Os dados podem ser obtidos tanto de maneira síncrona quanto assíncrona, permitindo a atualização dinâmica de políticas e avaliação de fatores externos. Quanto ao gerenciamento desses documentos, a ferramenta *Open Policy Administration Layer* (OPAL)[4], construída em cima do OPA, cria a possibilidade de administrar atualizações em tempo real (via *Websocket Pub/Sub*) tanto de políticas sincronizadas via *git*, quanto de dados vindos de diferentes fontes. Todas as decisões do OPA geram *logs*, assegurando auditabilidade e conformidade.

2.5. Análise de Logs

OpenSearch (OS) é um motor de busca e análise de dados de código aberto criado a partir do *ElasticSearch*[5]. Oferecendo flexibilidade e alta escalabilidade, o OS permite que os usuários realizem consultas e visualizem seus dados em tempo real. Sua estrutura é baseada em documentos e índices: os documentos são unidades básicas que armazenam informações no formato *JSON*; já os índices estruturam esses documentos, otimizando o desempenho das consultas [14]. Além das capacidades básicas, o OS possui uma vasta gama de *plugins* que adicionam funcionalidades à ferramenta. Dentre os *plugins*, cabe destacar o *OpenSearch Alerting*[6]. Esse *plugin* permite a criação de monitores que realizam consultas periódicas utilizando os índices e, quando certas condições são atendidas, notificações de alertas são geradas. Cada alerta possui um nível de gravidade em uma escala de 1 a 5, onde 1 representa um alerta de alta gravidade e 5 representa um alerta de baixa gravidade. Diante disso, este trabalho explora a capacidade do OS de processar grandes volumes de dados e identificar irregularidades por meio do suporte à auditoria e ao monitoramento, atividade indispensável para uma ZTA.

3. Trabalhos Relacionados

A aplicação de *Zero Trust* (ZT) e controle de acesso em ambientes distribuídos e dinâmicos é um tema recorrente na literatura. Diversos trabalhos propuseram soluções de baixo custo que buscam mitigar o acesso aos recursos por usuários maliciosos, enquanto garantem o acesso dos usuários legítimos. Em arquiteturas de microsserviços, um dos pilares para a disponibilidade é justamente a adoção de *rate limiting* para reduzir *throttling* em clientes com comportamento esperado e no sistema, protegendo de sobrecargas

[1] `https://www.openpolicyagent.org`
[2] `https://www.cncf.io/`
[3] `https://www.openpolicyagent.org/docs/latest/policy-language/`
[4] `https://docs.opal.ac`
[5] `https://opensearch.org/`
[6] `https://github.com/opensearch-project/alerting`

causadas por mau uso ou ataques de Negação de Serviço [Chandramouli 2020]. Nesta seção, alguns desses trabalhos serão apresentados e sua relação com o presente trabalho será discutida.

Em Freitas et al. (2024), um modelo de controle de acesso ABAC dinâmico com ZT foi apresentado para ambientes de *e-health* [Freitas et al. 2024]. No modelo, cada par *(ação, recurso)* possui uma sensibilidade de 0 a 100, onde 0 é não sensível e 100 é altamente sensível. Para realizar uma ação sobre um recurso, o usuário precisa ter um nível de confiança pré-determinado ou se reautenticar, caso esse nível não atenda aos requisitos mínimos do recurso. A análise de confiança do usuário é feita através de um mecanismo de penalidades que avalia o comportamento do usuário a partir de atributos e de risco em três perspectivas: contexto, dispositivo e histórico. O sistema demonstrou resultados satisfatórios na detecção de anomalias e no controle de acesso; entretanto, em um dos cenários avaliados, quando há roubo de credenciais, o atacante ainda possui tanto acesso a rotas de baixa sensibilidade quanto a se reautenticar múltiplas vezes, havendo brecha para múltiplas requisições e, assim, um ataque de negação de serviço, aumentando a latência de usuários legítimos.

Por outro lado, a proposta de Dimitrakos et al. (2020) traz uma abordagem inspirada em ZTA para sistemas de microsserviços voltados a *Internet of Things* (IoT) em ambientes multissensoriais, denominada UCON+ [Dimitrakos et al. 2020]. Este trabalho leva em consideração dispositivos como reconhecimento de voz, reconhecimento facial, digitais, entre outros. A partir de uma determinada operação, como a requisição de desbloqueio de porta, a arquitetura apresentada utiliza algoritmos de ABAC, dados do componente PIP, expressões de confiança desenvolvidas na pesquisa e monitoramento contínuo, para então gerar uma pontuação de confiabilidade que é utilizada para permitir ou negar a ação.

Por fim, o trabalho de Lukaseder et al. (2020) investiga o controle de acesso ABAC voltado para ZT em redes acadêmicas. Essa pesquisa desenvolve a arquitetura Alekto[7], cujo plano de controle é composto por um motor de políticas associado a um motor de confiança e uma fonte de dados para responder ao acesso a um *proxy* que controla o fluxo de rede, diferenciando usuários e dispositivos na análise. No entanto, o modelo de políticas utilizado baseia-se apenas nas informações de tentativas de autenticações realizadas, considerando somente ataques de força bruta no âmbito da autenticação, como sucessivas falhas de login de um mesmo usuário ou de um dispositivo.

Apesar do trabalho Freitas et al. (2024) apresentar uma solução eficaz que implementa os princípios do ZT em ambientes de *e-health*, existem oportunidades de maior cobertura de segurança, como a implementação de *rate limiting* para proteção via *throttling*, além de que, ao adaptar para contextos mais mutáveis e gerais, agregaria valor ao sistema a alteração dinâmica e em tempo de execução dos níveis de sensibilidade dos recursos. Em relação ao trabalho de Dimitrakos et al. (2020), este traz inovações e aplica princípios muito semelhantes de monitoramento e análise contínua a partir das informações de ambiente aos que serão apresentados, porém tem sua aplicação e contexto voltados para casas inteligentes, com parâmetros e complexidade diretamente relacionados à proposta de IoT, além de ter maior foco na escalada de privilégios. Finalmente, o trabalho de Lukase-

[7]`https://github.com/vs-uulm/alekto/tree/master`

der et al. (2020) não considera uso indevido do sistema enquanto autenticado e, além de não trabalhar com ferramentas consolidadas, o estudo não se aprofunda no controle de acesso e demonstra-se preliminar em sua aplicação. Diante disso, o presente trabalho visa apresentar duas abordagens de fácil integração de controle de acesso dinâmico ABAC em concordância com os princípios do ZT, que possam simplificar a adoção de princípios de segurança de forma ampla, robusta e generalizada para aplicações, possibilitando maior flexibilidade para os seus administradores.

4. Abordagens Propostas

As abordagens propostas basearam-se na ZTA de referência da Figura 1. Nas próximas subseções será descrito, primeiramente, o modelo de penalidades utilizado por ambas as abordagens e, em seguida, a proposta baseada em um motor de políticas e a baseada em análise de *logs*. As propostas consideram tanto usuários finais que acessam um serviço alvo, como também serviços clientes acessando este alvo. As seções abaixo descrevem a arquitetura das abordagens propostas, assim como potenciais implementações de um sistema que as implementa com base em ferramentas populares de código aberto.

4.1. Abordagem com Motor de Políticas

A proposta baseada em motor de políticas tira proveito da capacidade de centralização e transparência na definição de regras de um motor típico para a tomada de decisão a partir de dados dinâmicos e contextuais. Após a definição granular de controle de acesso e a utilização de uma política de quotas que leva em consideração a heterogeneidade dos recursos, o motor irá definir o resultado da solicitação de autorização feita pelo *proxy*. Neste trabalho, escolhemos o OPA como motor de políticas.

4.1.1. Arquitetura

A Figura 2 ilustra a arquitetura da proposta, o motor OPA é considerado como o único ator no PDP, o Processador de Quotas funciona como fonte de verdade no PIP e o Envoy atua como o PEP. Primeiramente, o sistema recebe a chamada do cliente no Envoy, que encaminha os dados da requisição para o OPA e espera a definição da autorização. Então, o motor de políticas primeiro avalia o RBAC, podendo negar imediatamente o pedido caso as políticas definidas não sejam atendidas, retornando o status 403 *Forbidden*, ou prosseguir na execução. Após ser bem-sucedido no RBAC, o OPA avaliará outros atributos com o modelo de penalidades e quotas gastas da Seção 4.1, consultando as quotas disponíveis para o usuário na janela de tempo de avaliação no seu registro de dados. Estes dados estão sendo constantemente atualizados pela estrutura do OPAL, que, por sua vez, recebe as informações mais recentes de um agente intermediário, o Processador de Quotas na figura, alimentado pelos *logs* de acesso enviados diretamente do OPA na função de *webhook* e computa as quotas utilizadas. Caso algum mecanismo falhe na atualização de quotas, as políticas checam se os dados tiverem sido injetados há mais de 5 minutos e o motor aceita as requisições, registrando nos *logs* o ocorrido, sempre evitando bloquear usuários legítimos. A saída será 429 *Too Many Requests* caso a quantidade de quotas do usuário seja insuficiente e, do contrário, a requisição será aprovada e encaminhada para o serviço desejado.

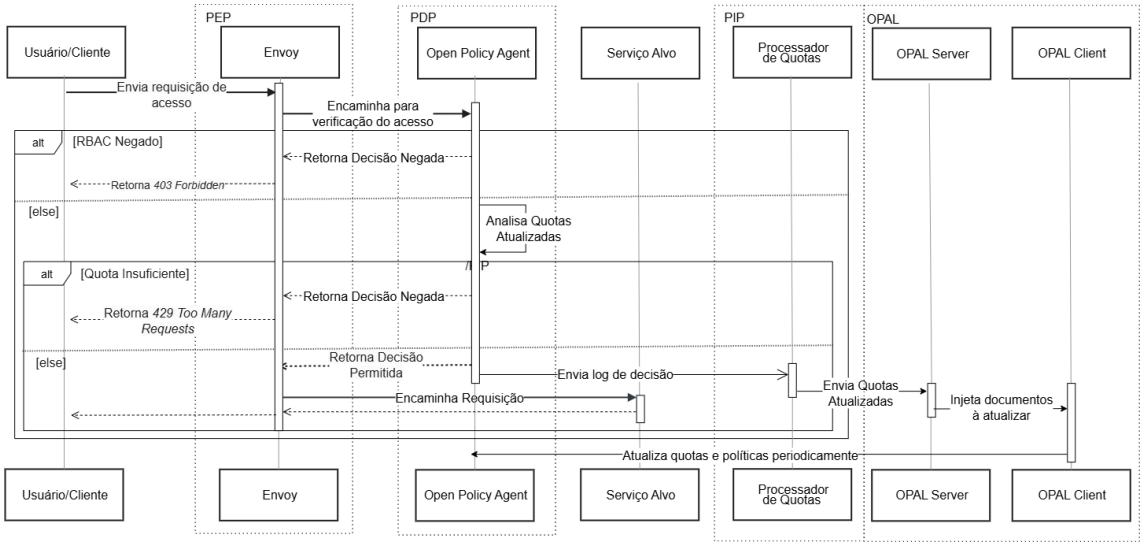

Figura 2. Diagrama de sequência da solução com motor de políticas.

4.1.2. Políticas de Acesso

Políticas de acesso bem definidas e transparentes são fundamentais para a manutenção da segurança e operação eficiente do sistema. A utilização de um motor de políticas centraliza a gestão de regras em um único componente, permitindo maior controle e flexibilidade. Neste trabalho, além da aplicação de RBAC com base no principal ID da requisição, foram implementadas regras que utilizam atributos de ambiente, configurando um sistema ABAC, alinhado aos princípios do modelo ZT. As políticas baseadas em atributos desenvolvidas estão organizadas nos seguintes tópicos:

- ***Rate Limiting* com Custos por *Endpoint* e Quota por Usuário.** Esta abordagem reconhece a heterogeneidade dos recursos computacionais, considerando que diferentes *endpoints* possuem demandas distintas de CPU e memória. Para proteger o sistema contra uso indevido por agentes maliciosos, foi implementada uma estratégia de quotas a partir de dois conceitos principais:
  I. **Custos por *Endpoint*.** Cada *endpoint* possui um custo associado em quotas, definido pelo administrador do sistema ou gerenciado dinamicamente por um serviço que injeta definições no documento de políticas do OPA. O custo reflete o impacto computacional das operações e é utilizado como métrica para priorização de recursos.
  II. **Quotas.** Cada usuário ou serviço recebe uma quantidade limitada de quotas, configuradas pelo administrador, para serem consumidas dentro de uma janela de tempo. Sempre que um usuário acessa um *endpoint*, a quota disponível é reduzida proporcionalmente ao custo do *endpoint*, garantindo que os recursos computacionais não sejam excedidos dentro do período estipulado. Todas essas operações são registradas no componente Processador de Quotas.
- **Análise de Data e Hora.** A gestão de quotas é adaptável, podendo levar em consideração sazonalidades, como horários comerciais ou períodos noturnos. Configurações de quotas específicas para horários de baixa demanda podem ser

ajustadas dinamicamente por meio de documentos JSON, otimizando o uso de recursos e prevenindo comportamentos anômalos em clientes fora do expediente.

- **Detecção e Resposta a Anomalias.** Para melhorar a resposta a situações anômalas, o OPA expõe uma API que permite a atualização em tempo de execução das políticas de acesso. Isso inclui, por exemplo, a adição de IDs de usuários a uma lista de bloqueio ou o ajuste dinâmico dos custos dos *endpoints* com base em métricas de uso. Embora este trabalho não tenha implementado o serviço integrado para modificar os valores de forma autônoma, foi configurado suporte para essas operações e interação com uma lista de bloqueio que pode ser editada externamente. A flexibilidade para definição de anomalias e métricas adaptáveis reforça a capacidade de resposta dinâmica do sistema.

É importante notar que a flexibilidade do sistema não implica em complexidade. Ajustar o valor dos custos do *endpoint* para refletir sempre um custo unitário, implementaria um sistema com *rate-limiting* convencional. Ao mesmo tempo, o isolamento dessas configurações, junto com as configurações de horário de acesso e adição de usuários bloqueados, facilita o refinamento dos parâmetros pelo operador.

4.2. Abordagem com Análise de *Logs*

O segundo sistema avalia o comportamento do sujeito para determinar a decisão de acesso. Padrões anômalos ou que não seguem boas práticas no uso dos recursos são identificados por meio da análise de *logs* com um motor de busca, o OS. Ao detectar uma violação de política de acesso, um alerta é gerado e uma penalidade é aplicada ao sujeito. Além disso, atributos dos recursos também são monitorados e podem resultar em penalidades. Dessa forma, antes de permitir ou negar uma requisição de acesso, é verificado o nível de penalidade do sujeito e do recurso a ser acessado. Isso garante que a confiança dos componentes seja continuamente avaliada antes da autorização ser concedida, conforme definido no modelo ZTA descrito em Borchert et al. (2024).

4.2.1. Arquitetura

Na arquitetura do sistema com análise de *logs*, todos os acessos são centralizados em uma única entidade, o Envoy, que possui um *listener* atuando como PEP para aplicar as restrições de acesso. Para verificar e validar cada acesso no PDP, um filtro Lua[8] do Envoy busca informações sobre o sujeito e o recurso no *State Storage* (SS) e define a decisão. O SS e o OS compõem os PIPs armazenando informações sobre as penalidades e os acessos, respectivamente. Assim, o sistema implementa os três principais módulos de uma ZTA.

O ciclo de vida de uma requisição, descrito na Figura 3, começa com o usuário enviando sua requisição ao *listener* do serviço no Envoy. O listener encaminha o pedido para o filtro Lua que irá verificar e determinar a decisão de acesso. Para isso, informações sobre as penalidades do sujeito e do recurso são buscadas no SS. Nesta etapa, as penalidades estão armazenadas em janelas de tempo acumulativas de tamanho $window\_size$, onde as do sujeito são identificadas pelo *Uniform Resource Identifier* (URI) apresentado

[8] `https://www.lua.org`

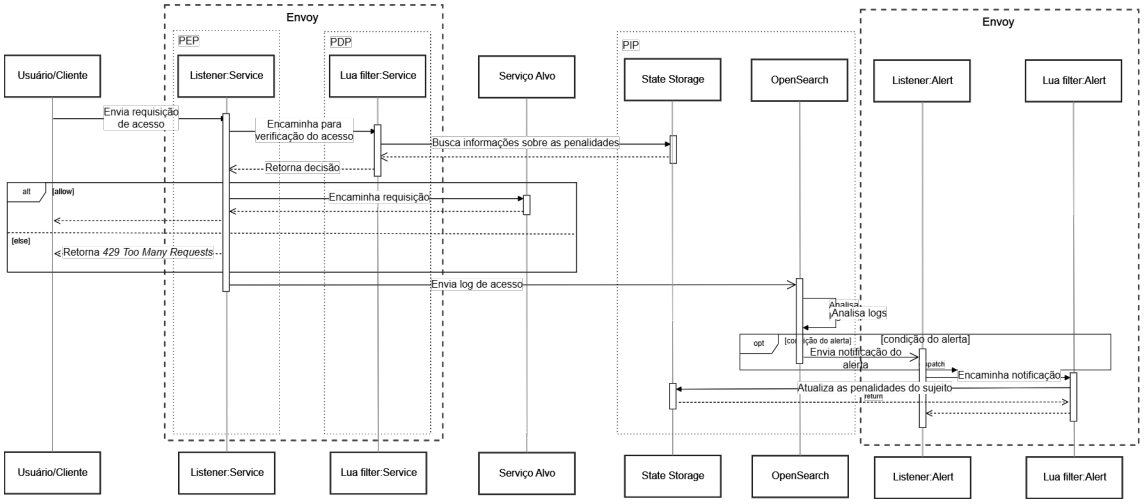

Figura 3. Diagrama de sequência da solução com análise de logs.

no certificado e as do recurso são identificadas por uma chave global[9]. O tamanho da janela de tempo, $window\_size$, definido para o sistema, foi de cinco minutos. Daí, se algumas das penalidades ultrapassarem o limite de penalidades $penalty\_limit$, o filtro Lua nega a requisição e o listener retorna 429 *Too Many Requests*. Caso contrário, a requisição é encaminhada normalmente para o serviço. Ao final da requisição, o *log* com informações do acesso, como horário, duração, código da resposta e URI, é enviado para o OS.

Paralelamente, o OS executa consultas periódicas nos *logs* para encontrar anomalias ou comportamentos que violem algumas das políticas de acesso definidas. Se alguma política for violada, o OS envia uma notificação de alerta com o tipo de política e a severidade ao *listener* de alertas do Envoy. Em seguida, a notificação é encaminhada a um filtro Lua que atualiza as penalidades no SS do sujeito com base no URI ou do recurso com base na chave global. As penalidades são inversamente proporcionais à severidade do alerta, de modo que, na configuração padrão, alertas com severidade um têm cinco pontos de penalidade e alertas com severidade cinco têm um ponto de penalidade.

4.2.2. Modelo de Penalidades

As penalidades indicam que uma ou mais políticas de acesso foram violadas. Todas as penalidades são armazenadas no SS. O SS é um armazenamento chave-valor, implementado com base em um sistema REDIS[10] e armazena informações que precisam ser persistidas entre duas chamadas. Uma penalidade é um par $\{time, penalty\}$, onde $time$ indica o tempo do alerta e $penalty$ indica o valor da penalidade associada à gravidade do alerta. As penalidades são armazenadas em um conjunto ordenado, permitindo que operações em intervalos sejam aplicadas com eficiência. Assim, quando dois alertas A e B possuem penalidades $\{time_A, penalty_A\}$ e $\{time_B, penalty_B\}$, respectivamente, onde $time_A = time_B = time$, apenas uma entrada é armazenada no conjunto com o valor

[9]Neste trabalho, utilizamos o padrão SPIFFE para identidades. Neste padrão, a identidade é armazenada em um campo URI na extensão *Subject Alternative Name* (SAN) [SPIFFE Project 2025].

[10]`https://redis.io/`

$\{time, penalty_A + penalty_B\}$. Dessa forma, o cálculo de penalidade Pen_{key} para uma chave key é dado pela fórmula:

$$Pen_{key} = \sum_{\{time,penalty\} \in Penalties_{key}} penalty \quad (1)$$

Onde $Penalties_{key}$ é o conjunto de todas as penalidades da chave key. A soma das penalidades será usada para a avaliação de uso de uma quota, definida pelo operador da aplicação. Note que usar essa fórmula pode deixar o sistema em um estado de bloqueio, onde nenhuma operação será mais realizada pelo usuário, pois as penalidades são cumulativas. Por isso, foi implementada uma solução com janela de tempo. Antes de calcular a penalidade, são removidas as entradas $\{time, penalty\}$ do $Penalties_{key}$ cujo $time + window\_size < start\_time$, onde $window\_size$ é o tamanho da janela de tempo, e $start\_time$ é o tempo em que a requisição foi realizada. Em seguida, as penalidades são calculadas a partir da fórmula (1) normalmente. Isso tudo é feito com uma única chamada ao SS através do comando *EVAL*[11]. Através dessa estratégia simples, uma janela de tempo é implementada. Assim, a fórmula para o cálculo de penalidades pode ser reescrita como:

$$Pen_{key}(start\_time, window\_size) = \sum_{\substack{\{time,penalty\} \in Penalties_{key} \ \& \\ time+window\_size > start\_time}} penalty \quad (2)$$

Esta solução, além de prevenir bloqueios, também diminui a quantidade de entradas no SS, reduzindo o custo e melhorando o desempenho do sistema de penalidades.

4.2.3. Políticas de Acesso

As políticas de acesso definem quais comportamentos não seguem boas práticas e estão fora do padrão. Ao usar o OS para processar os *logs* e gerir as políticas de acesso, políticas podem ser adicionadas, removidas ou ajustadas em tempo real, o que é esperado de uma ZTA avançada. Cada política de acesso está associada a um monitor e um alerta. Os monitores aplicam consultas periódicas e geram notificações para os alertas acionados. Os alertas são configurados através do limiar, da janela de tempo avaliada na consulta e da severidade que pode assumir um valor na escala de um a cinco, inversamente proporcional à gravidade. Dessa forma, foram definidas quatro políticas do sujeito e duas políticas do recurso, conforme descritas na Tabela 1.

A primeira política limita a quantidade de requisições para aplicar *Rate Limiting* e garantir equidade. Da mesma forma, a segunda política de duração total somadas das requisições busca garantir equidade, uma vez que a duração representa o tempo de processamento da requisição e está associada ao uso de recursos computacionais. A terceira aplica penalidades aos sujeitos que desrespeitam decisões negadas, enquanto a quarta pune usuários que geram alertas com frequência. As duas últimas são referentes aos recursos e monitoram o uso de recursos do serviço, permitindo uma degradação graciosa do mesmo.

[11] `https://redis.io/docs/latest/commands/eval/`

Política	Tipo	Limiar	Janela de Tempo	Severidade
Quantidade de requisições	Sujeito	>1000	10 minutos	4
Duração das requisições	Sujeito	>10000 ms	10 minutos	3
Quantidade de requisições negadas	Sujeito	>100	60 minutos	2
Quantidade de alertas	Sujeito	>10	240 minutos	1
Consumo de CPU	Global	>80%	10 minutos	3
Consumo de memória	Global	>70%	10 minutos	2

Tabela 1. Políticas de acesso e alertas para o sistema com análise de logs.

5. Avaliação e Resultados

O sistema que implementa as abordagens propostas foi avaliado em três cenários: caso base, apenas com o *proxy* de entrada; caso de estudo com motor de políticas; e caso de estudo com motor de *logs*. A arquitetura de testes descrita na Figura 4 conta com seis máquinas virtuais, com suas especificações presentes na Tabela 2, utilizadas para isolar cada componente e facilitar as análises.

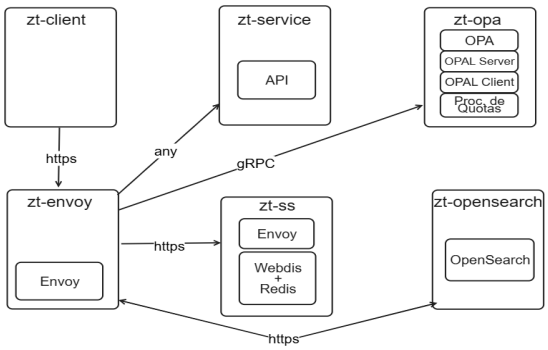

Figura 4. Infraestrutura de testes.

VM	CPU	Memória
zt-client	2c	4 GB
zt-service	2c	4 GB
zt-ss	2c	4 GB
zt-envoy	4c	8 GB
zt-opa	4c	8 GB
zt-opensearch	4c	8 GB

Tabela 2. Configurações das máquinas virtuais usadas na avaliação.

O teste simula um roubo de credenciais de um usuário legítimo autenticado, no qual um invasor realiza um ataque de força bruta. A carga de acessos foi gerada por meio da ferramenta de código aberto Vegeta[12], configurada para uma taxa constante de requisições. Durante os testes, as conexões são mutuamente autenticadas via mTLS, e os usuários (ou serviços clientes) apresentam certificados X.509 contendo suas identidades no campo URI, conforme o padrão SPIFFE.

Na simulação, apresentada na Figura 5, o usuário legítimo inicia o envio de requisições por 120 segundos a uma taxa de 1 requisição por segundo. Em seguida, o mesmo usuário, identificado pelo URI, passa a apresentar comportamento anômalo, com uma frequência de requisições de 40 por segundo, caracterizando um possível ataque.

No cenário em que apenas o *proxy* é utilizado, a latência média ficou em 6,86 milissegundos e não há mecanismo de resposta à anomalia, permitindo que todas as requisições sejam processadas indiscriminadamente. Em contraste, as demais soluções detectam corretamente o comportamento anômalo e passam a negar as requisições com o código 429 (*Too Many Requests*), conforme esperado.

[12] `https://github.com/tsenart/vegeta`

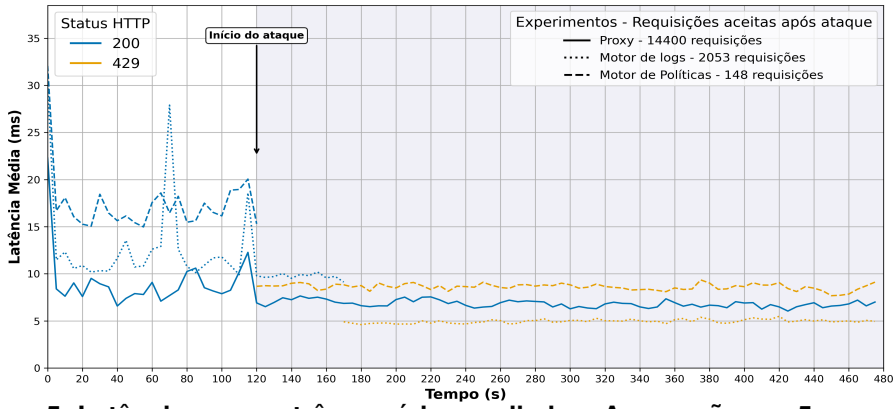

Figura 5. Latência com os três cenários avaliados. Agregação por 5 segundos.

No motor de políticas, a restrição foi aplicada eficazmente de modo quase instantâneo, em menos de 4 segundos, considerando limite máximo de 100 quotas por usuário (onde o custo de cada chamada foi configurado como 1), dentro da janela de avaliação de 60 segundos, obtendo a latência média de 8,77 milissegundos e permitindo apenas 148 requisições a partir do momento em que a taxa de acesso aumentou. Já no motor de *logs*, a mitigação ocorre assim que o usuário ultrapassa os limites definidos na Tabela 1 e os alertas de quantidade de requisições e de duração das requisições com severidades 4 e 3, respectivamente, são acionados. Esse mecanismo também resulta no bloqueio das requisições em menos de um minuto, entretanto é notado maior atraso, no qual foi permitido o acesso de 2053 requisições. Vale ressaltar ainda que ambas as abordagens demonstram impacto positivo na latência ao negar rapidamente as requisições maliciosas, com destaque para o motor de *logs* que obteve a média de 5,67 milissegundos considerando apenas o período de ataque.

Em paralelo, durante o experimento, foi simulada a atividade de um usuário com comportamento regular de 1 requisição por segundo para medir o impacto do ataque em sua latência, como se vê nas Figuras 6 e 7. Nessa análise, não foi observado impacto relevante, apesar de uma diminuição de cerca de 12 a 20% nas latências, uma vez que essa diferença também foi notada no sistema sem nenhuma restrição de anomalia, com apenas o *proxy* de intermediário, podendo ser causada por fatores não relacionados (ver Figura 8).

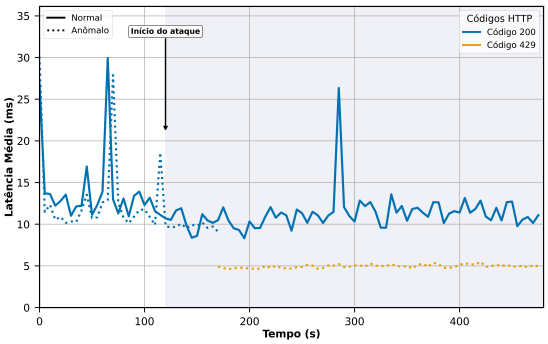

Figura 6. Motor de *Logs* - Impacto na latência do usuário normal durante ataque. Agregação por 5 segundos.

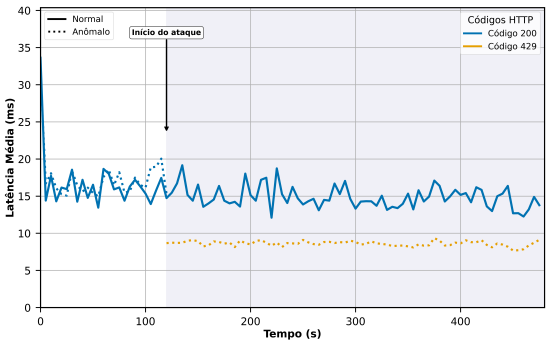

Figura 7. Motor de Políticas - Impacto na latência do usuário normal durante ataque. Agregação por 5 segundos.

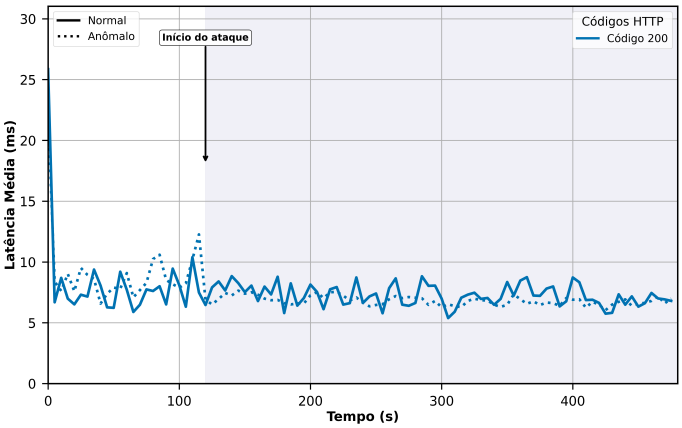

Figura 8. Proxy Puro - Impacto na latência do usuário normal durante ataque. Agregação por 5 segundos.

Entretanto, analisando a média de uso da CPU apresentada na Figura 9, observamos a maneira como cada experimento lida com a anomalia e o impacto no sistema. No cenário com apenas o *proxy*, todas as requisições são permitidas sem restrições, resultando em um aumento significativo no consumo isolado de CPU do serviço acessado em comparação com o mesmo componente no cenário com processamento *logs*, cerca de 4 vezes maior, e em mais de 23 vezes do terceiro cenário, sobrecarregando muito mais o serviço final, o que é um problema dependendo do consumo de recursos deste. O sistema completo utilizou cerca de 18,43% de um núcleo. No motor de *logs* e no motor de políticas, por outro lado, a utilização de recursos é distribuída entre os elementos da arquitetura, devido à execução paralela das computações entre os componentes, balanceando a carga de CPU consumida, totalizando aproximadamente 19,68% de utilização de um núcleo para o primeiro sistema e 32,67% para o segundo, onde apenas uma parcela de 3,22% e 0,63%, respectivamente, é do serviço final.

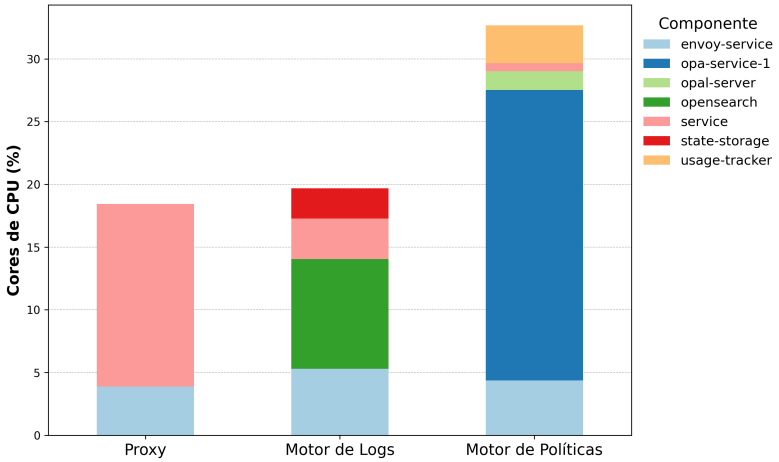

Figura 9. Consumo médio de CPU com os três cenários avaliados.

Em suma, ambas as soluções demonstram viabilidade e corretude, proporcionando detecção eficiente de anomalias, com possibilidade de ajuste para maior ou menor sensibilidade a comportamentos irregulares, conforme a necessidade do sistema. No caso do cenário com o motor de políticas (OPA/OPAL), a identificação e mitigação das ameaças

ocorrem rapidamente — em menos de 5 segundos —, protegendo o serviço de acesso irrestrito e evitando o desperdício de CPU no serviço final. Em contrapartida, o impacto na latência das requisições é levemente mais significativo, com aumento no uso de CPU na arquitetura do motor. Apesar disso, a solução mantém a escalabilidade e eficiência, sendo ideal para ambientes que priorizam respostas rápidas e políticas simples e eficazes, além da facilidade do OPAL de administrar as políticas via *git* e permitir paralelização de diferentes motores de políticas para escalar o sistema, mantendo a consistência dos dados.

Já no cenário com o motor de *logs* (OpenSearch), o impacto na latência das requisições é mínimo para o usuário normal, o que favorece a experiência geral. No entanto, a detecção e o bloqueio de ataques levaram perto de 1 minuto, demonstrando uma resposta mais lenta em comparação ao motor de políticas. Ainda assim, essa abordagem é extremamente flexível e adequada para sistemas maiores, que demandam análises mais elaboradas e integradas. Além disso, os componentes envolvidos podem ser paralelizados, otimizados e evoluídos continuamente, garantindo adaptação às necessidades do ambiente sem comprometer a eficiência global das arquiteturas propostas.

6. Considerações Finais

Este trabalho propôs duas abordagens de arquiteturas baseadas em atributos para auxiliar na migração progressiva para um modelo de segurança ZT com alto nível de maturidade. Essas arquiteturas não são excludentes, pelo contrário, podem ser utilizadas de forma complementar. Assim, através de simulações de comportamentos anômalos, as duas propostas demonstraram-se eficazes na identificação de irregularidades no acesso e na proteção do serviço final contra sobrecargas causadas por ataques. Os resultados demonstraram a viabilidade das abordagens que têm como pontos fortes a simplicidade de adoção e eficácia, sendo pontos de partida úteis à migração para uma ZTA. A proposta com motor de políticas já contempla pontos de maturidade intermediária, por meio da inclusão de atributos contextuais, e também atinge um nível avançado, ao viabilizar autorização contínua e resposta ágil a anomalias, com ações preventivas integradas. Esta ainda pode evoluir com motores distintos para tópicos e rotas distintas. Em paralelo, a arquitetura com motor de *logs* abrange ainda mais tópicos avançados, com monitoramento constante e análises contínuas de ameaças com alertas e menor consumo de CPU, permitindo uma degradação mais graciosa a partir do estado do sistema. É importante destacar que evitar o ataque não somente evita o uso de recursos, mas também bloqueia tentativas de invasão por força bruta e a recuperação de dados para vazamento.

As configurações e dados resultantes deste trabalho estão disponíveis publicamente em um repositório GitHub[13] para referência. Como trabalho futuro, é proposta a aplicação de gatilhos de reautenticação em ambas as arquiteturas apresentadas, preservando o sistema após a identificação de anomalias. Por fim, a integração das abordagens propostas pode agregar as qualidades da análise de *logs* do sistema e dos serviços, com a facilidade no versionamento das regras e a escalabilidade na tomada de decisão dos motores de políticas, quando coordenados por uma camada de administração.

[13]https://github.com/AbraaoCF/zero-trust

Agradecimentos

Agradecemos à FAPESQ-PB pelo apoio ao SmartCampus UFCG através do projeto "Ecossistema baseado em IoT para gerência de água e energia" (edital 09/2021, Demanda Universal), uma parceria entre a UFCG e a FAPESQ-PB.

Referências

Athena, J. and Sumathy, V. (2017). Survey on public key cryptography scheme for securing data in cloud computing. *Circuits and Systems*, 8(3).

Borchert, O., Howell, G., Kerman, A., Rose, S., Souppaya, M., Ajmo, J., Fashina, Y., Grayeli, P., Hunt, J., Hurlburt, J., Irrechukwu, N., Klosterman, O., Slivina, O., Symington, S., Tan, A., Scarfone, K., Barker, W., Gallagher, P., Palermo, A., Balaji, M., Cerini, A., Barosin, J., et al. (2024). Implementing a zero trust architecture: High-level document. Special Publication 1800-35, National Institute of Standards and Technology. CODEN: NSPUE2.

Chandramouli, R. (2020). Security strategies for microservices-based application systems. Technical Report NIST SP 800-204, US Department of Commerce, National Institute of Standards and Technology.

de Mello, E. R., de Chaves, S. A., da Silva, C., Wangham, M. S., Brito, A., and Henriques, M. A. A. H. (2022). *Autenticação e Autorização: antigas demandas, novos desafios e tecnologias emergentes*, pages 1–50. Sociedade Brasileira de Computação.

Dimitrakos, T., Dilshener, T., Kravtsov, A., La Marra, A., Martinelli, F., Rizos, A., Rosetti, A., and Saracino, A. (2020). Trust aware continuous authorization for zero trust in consumer internet of things. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 1801–1812.

DoD (2022). Zero trust reference architecture. Technical report, Defense Information Systems Agency (DISA) and National Security Agency (NSA). Prepared by the Zero Trust Engineering Team.

Feldman, D., Fox, E., Gilman, E., et al. (2020). *Solving the Bottom Turtle — a SPIFFE Way to Establish Trust in Your Infrastructure via Universal Identity*. SPIFFE, 1 edition. Disponível em: `https://thebottomturtle.io`. Acesso em: 12 mar. 2025.

Ferreira, R. (2022). *Policy Design in the Age of Digital Adoption: Explore how PolicyOps can drive Policy as Code adoption in an organization's digital transformation*, chapter 3. Packt Publishing Ltd.

Freitas, L., Coelho, K., Nogueira, M., Vieira, A., Nacif, J., and Silva, E. (2024). Controle de acesso sensível ao contexto e zero trust para a segurança em e-health. In *Anais do XLII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 770–783, Porto Alegre, RS, Brasil. SBC.

IBM (2024). Cost of a data breach report 2024. `https://www.ibm.com/security/data-breach`. Acesso em: 01/04/2025.

MarkWide (2025). Cloud microservices market analysis- industry size, share, research report, insights, covid-19 impact, statistics, trends, growth and forecast 2025-2034. Accesso em: 01/04/2025.

Mhetre, N. A., Deshpande, A. V., and Mahalle, P. N. (2022). Experience-based access control in ubicomp: A new paradigm. *Journal of Computer and Communications*, 10(1).

NIST (2012). Guide to attribute based access control (abac) definition and considerations. NIST Special Publication 800-162. Disponível em: `https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-162.pdf`. Acesso em: 11/04/2025.

Rose, S., Borchert, O., Mitchell, S., and Connelly, S. (2020). Nist special publication 800-207: Zero trust architecture. Technical report, National Institute of Standards and Technology (NIST), Gaithersburg, MD. Acesso em: 10/4/2025.

SPIFFE Project (2025). The x.509 spiffe verifiable identity document. GitHub repository: standards/X509-SVID.md. Accesso em: 4/05/2025.

Xiao, Y., Jia, Y., Liu, C., Cheng, X., Yu, J., and Lv, W. (2019). Edge computing security: State of the art and challenges. *Proceedings of the IEEE*, 107(8):1608–1631.