Aprimorando o controle de acesso Zero Trust: Um mecanismo de penalidades em tempo real com Envoy e OpenSearch para decisões baseadas em logs

Davi Henrique Silva Guimarães
Universidade Federal de Campina Grande
davi.guimaraes@ccc.ufcg.edu.br

Andrey Elísio Monteiro Brito
Universidade Federal de Campina Grande
andrey@computacao.ufcg.edu.br

RESUMO

O surgimento da arquitetura de microsserviços trouxe novos desafios para a engenharia de software, em especial para a segurança. Nessa arquitetura, onde os microsserviços estão distribuídos entre diferentes clusters mantidos por empresas terceiras, a superfície de ataque é maior e o perímetro não pode ser definido com facilidade. E, mesmo que definido, uma vez dentro do perímetro, o atacante pode se mover lateralmente e violar a integridade de diversos componentes. Nesse contexto, surge o modelo *Zero Trust* que visa minimizar os privilégios e assegurar um controle de acesso granular, onde o nível de confiança das partes seja validado antes do acesso ser concedido. Este trabalho propõe um sistema de controle de acesso baseado em *Zero Trust* que implementa um mecanismo de penalidades alimentado por informações de logs de acesso dos usuários; e de atributos dos recursos. Os resultados mostram eficácia na identificação de irregularidades nos acessos e na segurança do sistema.

Keywords

Zero Trust, Security, Access Control, Logging, Proxying.

1. INTRODUÇÃO

O aumento de popularidade da arquitetura com microsserviços e o surgimento de empresas que fornecem infraestrutura, plataforma, software como serviço (IaaS, PaaS, SaaS), como AWS, Vercel e Azure, mudaram completamente a arquitetura típica dos sistemas. Antes, 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 [1]. Essa mudança de panorama trouxe novos desafios para a ciência da computação, em especial às áreas de segurança e privacidade [2].

Nesse cenário, modelos tradicionais de segurança são inviáveis ou não conseguem proteger o sistema contra ataques mais sofisticados. 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 [3]. Ainda, abordagens como *Public Key Infrastructure* (PKI), onde o acesso é baseado em credenciais estáticas, possuem uma manutenção extremamente custosa e não escalam bem à medida que o sistema ganha complexidade e tamanho, já que a gestão das credenciais é manual [3]. Outro problema dessa abordagem é que credenciais podem ser roubadas ou comprometidas, e o seu impacto pode ser desastroso. Segundo [4], em 2024, o impacto médio de violação de dados por falhas desse tipo foi de 4,81 milhões de dólares, tendo um ciclo de vida médio de 292 dias para identificar e conter a violação.

A partir da necessidade por modelos de segurança mais sofisticados e dinâmicos que consigam ligar com a crescente complexidade dos sistemas, surge, então, o paradigma *Zero Trust* (ZT). ZT é um conjunto de princípios e ideias que minimizam a incerteza através de políticas de acesso mínimo. ZT assume que as ameaças podem estar dentro ou fora do sistema e, por isso, são necessárias políticas granulares de monitoramento contínuo, de privilégios mínimos e de avaliação de risco adaptáveis. Em um ambiente ZT, a confiança não é concedida, mas continuamente avaliada ao longo do tempo [5]. Nesse âmbito, os logs desempenham um papel crucial. Com a política de "sempre verificar, nunca confiar", os pedidos de acesso devem ser avaliados em tempo real e, através das informações contidas nos logs, as organizações podem identificar atividades suspeitas e potenciais riscos à integridade do sistema.

Este trabalho propõe um sistema de controle de acesso baseado nos princípios do ZT que implementa um mecanismo de penalidade a partir da análise de logs para assegurar privilégios mínimos e acesso seguro aos recursos. O sistema é composto por um proxy, o Envoy[1], que controla a comunicação entre usuários e um serviço interno, registrando detalhadamente os logs de acesso, e uma ferramenta de busca e análise, o Opensearch[2], utilizado para processamento e monitoramento dos logs. A contribuição do sistema está na utilização de componentes comumente utilizados em arquiteturas ZT (ZTA), como *proxying*, *data analyzes*, e *alerting*, para criar uma solução simples e sofisticada.

A estrutura do trabalho é organizada da seguinte forma:Na seção 2, são apresentadas as bases teóricas do ZT e princípios das ferramentas utilizadas. Na seção 3, são discutidos os trabalhos relacionados. Na seção 4, a implementação do sistema é descrita. Finalmente, na seção 5, as conclusões são apresentadas.

[1] https://www.envoyproxy.io/
[2] https://opensearch.org/

2. FUNDAMENTAÇÃO TEÓRICA

2.1 Zero Trust

Zero Trust (ZT) é um modelo de cibersegurança que parte da premissa de que não existe confiança inata [5]. 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 apresentado um esquema abstrato de acesso em ambientes ZT.

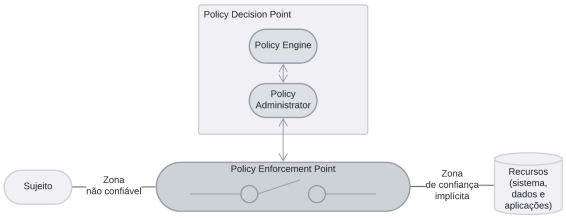

Figura 1. Esquema de acesso abstrato no ZT.

No esquema, o sujeito deseja acessar um recurso do sistema. O acesso é dado pelo *Policy Decision Point* (PDP) e pelo *Policy Enforcement Point* (PEP). 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 de fontes externas. Já o *Policy Administrator* (PA) é responsável por estabelecer a comunicação entre sujeito e o recurso por meio do envio de comandos para o 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.

Dessa forma, o acesso é concedido de modo que todos que estão após o PEP tenham o mesmo nível de confiança criando uma zona de confiança implícita [5]. Vale ressaltar que essa zona é a menor possível e que todas as entidades, sujeitos e recursos, mesmo que façam parte da rede interna, são verificados. Portanto, o PEP monitora a confiabilidade de todos os componentes do sistema.

2.1.1 Arquitetura

Em uma arquitetura de ZT (ZTA), os seguintes princípios devem ser seguidos:

P_1 Todas as fontes de dados e serviços são considerados recursos;

P_2 Toda a comunicação é protegida independente da localização;

P_3 Acesso aos recursos é baseado na sessão;

P_4 As políticas de acesso são dinâmicas e podem incluir análise comportamental e atributos do ambiente;

P_5 Os recursos são monitorados e suas integridades avaliadas;

P_6 A autenticação é dinâmica e realizada antes do acesso ser concedido;

P_7 O máximo de informações sobre os estados dos recursos, da infraestrutura da rede e da comunicação é coletado e utilizado para melhorar a segurança do sistema.

Esses princípios são universais e não dependem da tecnologia associada. Eles podem ser aplicados dentro da organização e entre organizações parceiras, mas não entre relações anônimas públicas ou processos voltados para o consumidor [5].

2.2 Gestão de Identidade e de Acesso

A Gestão de Identidade e de Acesso (IAM) é o conjunto de procedimentos e de tecnologias que auxiliam na identificação de usuário ou máquina, de modo que o acesso a recursos seja feito pelas entidades certas e no momento certo. Através de políticas e regras, a IAM assegura níveis de acesso adequados para cada recurso da organização, garantindo identidades digitais para as entidades. Isso é fundamental para a proteção dos dados contra acessos indevidos [6]. No gerenciamento de acesso, regras são definidas para determinar as permissões de um sujeito. A associação de um sujeito (pessoa, dispositivo, conta etc.) com uma ação (ler, escrever, modificar etc.), e com um recurso (arquivo, máquina, serviço etc.) corresponde a uma regra. Note que essa associação pode depender do tempo e da localização do sujeito ou recurso [7].

A implementação de IAM deve ser composta por três componentes básicos (Regra AAA): autenticação; autorização; e auditoria. A autenticação consiste em identificar corretamente uma entidade que interage com o sistema. Isso garante que qualquer ação executada sobre o sistema possua um ator. Já a autorização refere-se às permissões de um sujeito autenticado previamente. Por fim, a auditoria garante que todas as ações do sistema possam ser rastreadas [7].

2.3 Autorização e controle de acesso

Enquanto que a autenticação identifica o sujeito, a autorização determina quais são os recursos acessíveis por esse sujeito. 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 modelos: 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, esse modelo é limitado dado a dificuldade para definir os papéis e suas respectivas permissões, também conhecido como "engenharia de papel". 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 a definição de mais papéis e permissões, e isso gera um conflito [8].

O modelo de controle de acesso baseado em atributos (ABAC) é uma evolução do modelo RBAC [9]. 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, a requisição de acesso é concedida ou não. Isso permite a criação de políticas flexíveis e dinâmicas [10]. Além disso, ao

considerar papéis como atributos, o também ABAC pode implementar o RBAC.

2.4 Envoy

Envoy é uma ferramenta de código aberto de *proxying* certificada pelo *Cloud Native Computing Foundation* (CNCF), o que indica que é estável e pode ser usado em produção [11]. Envoy busca criar transparência para as aplicações ao lidar com o problema comunicação de modo que as aplicações não precisem entender a topologia da rede. Isso permite que os desenvolvedores das aplicações foquem nas lógicas de negócio, já que todo o problema de compatibilidade entre diferentes tecnologias é direcionado ao Envoy. Além disso, dissociar as preocupações da rede da aplicação, evitam inconsistência nos mecanismos de *tracing* e *logging*, tornando o processo de depuração menos custoso [12].

O funcionamento do Envoy é baseado em *building blocks*. Os quatro principais são: *listener*, *route*, *cluster*, e *endpoint*. Envoy expõe *listeners* que recebem requisições e as encaminha para *filter chains*. Uma *filter chain* é uma lista encadeada com filtros que processam uma requisição. Na Figura 2, é representada a relação dos listeners com as *filters chains*.

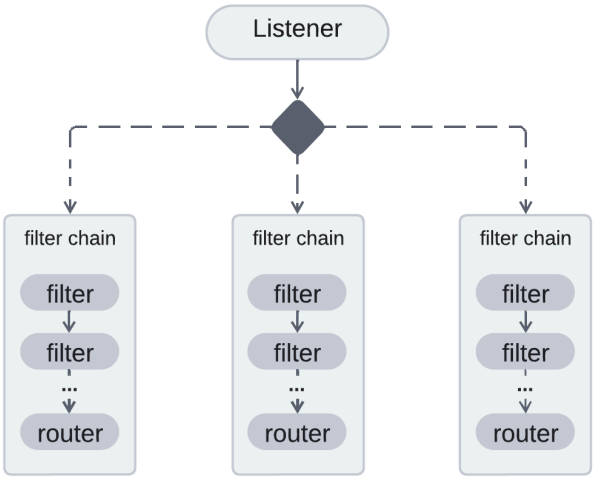

Figura 2. Relação entre Listener e Filter Chains.

Ao final de toda *filter chain*, existe o *router* que encaminha a requisição para um *cluster* de acordo com as configurações de roteamento. Esse cluster define um conjunto de *endpoints* de um serviço e, a partir das informações da requisição, a requisição é encaminhada para um dos endpoints [13].

Outro diferencial oferecido pelo Envoy é a capacidade de monitoramento. Envoy possui suporte para estatísticas compatíveis com diversas ferramentas de monitoramento; para formatação de logs; e para *tracing* distribuído com diferentes provedores [12].

Assim, sua configuração extensível e seu sistema robusto de observabilidade tornam o Envoy ideal para cenários ZT. As configurações do Envoy exploradas neste trabalho são o filtro Lua, que permite executar *scripts* em *Lua*, e a customização dos logs de acesso.

2.5 OpenSearch

OpenSearch é um motor de busca e análise de dados de código aberto criado a partir do ElasticSearch[3]. Oferecendo flexibilidade e alta escalabilidade, o OpenSearch 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[4]; já os índices agrupam esses documentos otimizando a performance das consultas [14].

Além das capacidades básicas, o OpenSearch possui uma vasta gama de plugins que adicionam funcionalidades à ferramenta. Dentre os plugins, cabe destacar o OpenSearch Alerting[5]. Esse plugin envia notificações, os alertas, para anormalidades encontradas através de monitoramento. Para gerar alertas, primeiramente é necessário definir monitores. Um monitor é um componente que realiza consultas periódicas em janelas de tempo sob índices e, quando certas condições, conhecidas como *thresholds*, são atendidas, *triggers* são acionados, gerando alertas. Um único monitor pode ter mais de um *trigger*. Cada *trigger* 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, esse trabalho explora a capacidade do OpenSearch de processar grandes volumes de dados e identificar irregulares por meio do suporte a auditoria e monitoramento, atividade indispensável para uma ZTA.

2.6 State Cache

O cache é uma técnica utilizada para armazenar dados temporariamente em um local de acesso rápido com o objetivo de otimizar o desempenho de sistemas. Devido a sua eficiência, o cache é aplicado em diversos contextos como armazenamento de consultas recorrentes em banco de dados; armazenamento de DNS (*Domain Name System*); gerenciamento de sessões HTTP; e armazenamento de estados [15]. Dentre as tecnologias de cache, será descrita a mais utilizada: Redis [16].

Redis (Remote Dictionary Server) é um banco de dados de código aberto de armazenamento chave-valor de alta performance. No Redis, os dados são armazenados em memória e isso permite realizar operações extremamente rápidas em comparação com outros bancos de dados que fazem operação em disco como MongoDB [17]. Isso o torna ideal para cenários que exigem baixa latência, como cache, gerenciamento de sessões e classificações. O tipo de dados básico do Redis é a string, porém Redis suporte outros tipos mais complexos como listas, conjuntos não ordenados e conjuntos ordenados [18].

Ciente disso, este trabalho utiliza o Redis para armazenar os estados das penalidades dos usuários. A velocidade, a confiabilidade e o desempenho do Redis são ideias para o sistema proposto, pois permite que as decisões de acesso sejam realizadas mais rapidamente, diminuindo o impacto na usabilidade do serviço. O tipo de dados explorado é o conjunto ordenado[6], dada a sua eficiência em operações em intervalos.

[3] https://www.elastic.co
[4] https://www.json.org/json-en.html
[5] https://github.com/opensearch-project/alerting
[6] https://redis.io/docs/latest/develop/data-types/sorted-sets/

Vale ressaltar que a versão Redis Community Edition (CE) permite apenas comunicação via TCP socket, enquanto que o filtro Lua do Envoy possui suporte nativo apenas para chamadas HTTP [19][20]. Sendo assim, foi necessário adicionar uma camada que encaminha as chamadas HTTP do Envoy para o Redis. A solução utilizada para isso foi o Webdis[7], uma implementação de código aberto de uma interface HTTP para Redis. A partir deste ponto, todas as menções à State Cache referem-se à abstração proporcionada pelo Webdis com o Redis.

3. TRABALHOS RELACIONADOS

A aplicação de 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 dos recursos por usuários maliciosos, enquanto garantem o acesso dos usuários legítimos. Outros, por sua vez, se limitaram em definir e categorizar controles que implementam princípios do ZT. Nessa seção, alguns desses trabalhos serão apresentados e sua relação com o presente trabalho.

Em [21], um modelo controle de acesso ABAC dinâmico com ZT foi apresentado para ambientes de e-health. 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 nível de confiança pré-determinado ou se reautenticar, caso esse nível 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.

Um esquema de controles para integração de ZT em *cyber supply chains* foi proposto por [22]. No estudo, práticas de mitigação de risco de cinco domínios (infraestrutura e redes, identidade, dispositivo, governança e dados, e aplicação) foram categorizadas em três estágios (básico, intermediário, e avançado), de acordo com o seu grau de conformidade com o modelo de ZT. Dentre as práticas, o estudo identificou políticas de audibilidade e formatação dos logs como controles básicos e essenciais para os demais estágios; análise de logs como uma boa prática; e o monitoramento constante com decisões de acesso em tempo real e alertas de segurança como atividades com alto grau de automação e concordância com o ZT.

Apesar do trabalho [21] apresentar uma solução eficaz que implementa os princípios do ZT em ambientes de e-health, aplicar essa solução em outros contextos, onde os sistemas são compostos por diversos componentes e estão em constante evolução, não é simples, haja vista a carga necessária para definir a sensibilidade dos recursos e a penalidade associada às perspectivas. Em relação ao trabalho [22], este apenas enumera controles para integração do ZT a fim de ajudar os administradores de sistemas a identificarem o grau de concordância dos seus sistemas com o ZT e não propõe de fato soluções para o problema de segurança. Diante disso, o presente trabalho visa implementar um modelo de controle de acesso dinâmico ABAC em concordância com os princípios do ZT, utilizando um mecanismo de penalidades baseado em comportamentos suspeitos identificados através do monitoramento dos logs de acesso em tempo real.

[7] https://webd.is/

4. SISTEMA PROPOSTO

O sistema proposto baseou-se no esquema de acesso ZT descrito na seção 3.1. No sistema, indivíduos autenticados requisitam acesso sobre determinado serviço e, através de um mecanismo de penalidades, sua requisição pode ser aprovada ou negada. Note que, mesmo autenticados, não deve-se assumir que são confiáveis. O objetivo é garantir o acesso dos usuários legítimos e proteger a saúde do serviço, enquanto se defende contra usuários maliciosos.

Para simular um ambiente real, foi criado um serviço que atende a requisições HTTP, consulta um banco de dados e retorna os itens presentes neste banco. Isso representa um cenário comum onde usuários acessam uma API (*Application Programming Interface*) através de requisições HTTP para obter informações. Para identificar os usuários, são utilizados certificados X.509 convencionais com campo *Uniform Resource Identifier* (URI) indicando a sua identidade.

O modelo de controle de acesso implementado é baseado em sistema de penalidades. O sistema avalia o comportamento do sujeito a partir de políticas de acesso e atribui penalidades quando identifica comportamentos que violem alguma política. Da mesma forma, o sistema monitora o serviço e atribui penalidades ao identificar anomalias. Assim, quando a requisição de acesso é avaliada, são verificados os níveis de penalidade do sujeito e do serviço e então a decisão de acesso é tomada pelo Envoy. Isso garante que todos os componentes estão sendo verificados antes da autorização ser concedida, como é definido no modelo ZT.

Os diagramas de sequência das Figuras 3, 4 e 5 ilustram a interação dos componentes nos dois fluxos principais: requisição de acesso de um sujeito ao serviço (bem sucedida e rejeitada); e processamento de alertas gerados pelo OpenSearch sobre anomalias encontradas na análise dos logs de acesso. Esses fluxos serão detalhados nas seções 4.3 e 4.4.

4.1 Políticas de acesso

As políticas de acesso definem quais comportamentos não seguem boas práticas e estão fora do padrão. Definir políticas robustas é essencial para controlar o acesso aos recursos, pois o nível de penalidade do sujeito será medido por meio delas. Ao usar o OpenSearch 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.

As políticas de penalidade são divididas em duas categorias: sujeito e global. Políticas do sujeito indicam comportamentos esperados de um usuário consciente e responsável. Já políticas globais são aplicadas ao serviço e monitoram o uso de recursos computacionais. Para o sistema implementado, foram definidas quatro políticas do sujeito:

- **Quantidade de requisições**. Políticas de Rate Limiting são fundamentais para garantir a equidade dos recursos, evitando que um único usuário impacte negativamente o acesso dos demais.
- **Duração das requisições**. A duração de quanto tempo uma requisição leva para ser processada por um serviço é um bom indicativo de quanto tempo de CPU foi necessário. Sendo assim, controlar a duração das

requisições é um bom parâmetro para medir quanto de recursos um sujeito está consumindo do serviço.

- **Quantidade de requisições negadas**. Ter uma requisição negada indica que uma ou mais políticas foram violadas. Portanto, o sujeito deve aguardar para fazer uma nova requisição. Se continuar a realizar requisições mesmo tendo o acesso negado, esse sujeito deve ser penalizado.
- **Quantidade de alertas**. Um alerta indica que foi identificado uma anormalidade no comportamento de um sujeito. Quando um sujeito aciona diversos alertas, isso indica que ele não está respeitando as políticas de uso do sistema. Sendo assim, o sujeito deve ser penalizado com severidade.

Além dessas, foram definidas duas políticas globais:

- **Consumo de CPU**. Rejeitar acessos quando o uso de CPU está muito elevado protege serviços críticos que necessitam de alta disponibilidade. Isso dá tempo para o serviço se estabilizar, evitando degradação de desempenho.
- **Consumo de memória**. Definir limites baseado no uso de memória protege o serviço de erros como *out of memory* (OOM) e *memory leaks*.

4.2 Sistema de Penalidades

As penalidades indicam que uma ou mais políticas de acesso foram violadas. Todas as penalidades são armazenadas no State Cache. 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 {*time*, $penalty_A + penalty_A$}.

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*. Note que usar essa fórmula pode deixar o sistema em um estado de *deadlock*, pois as penalidades são cumulativas. Por isso, foi implementada uma abordagem de janela de tempo. Antes de calcular a penalidade, o script *access_control.lua*[8] remove 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 realizado. Em seguida, as penalidades são calculadas a partir da fórmula (1) normalmente. Através dessa estratégia simples, uma janela de tempo é implementada. Essa abordagem além de prevenir *deadlocks*, também diminui a quantidade de entradas no State Cache, reduzindo o custo e melhorando a performance. Assim, a fórmula para o cálculo de penalidades pode ser reescrita como:

$$Pen_{key} = \sum_{\{time_{key},\ penalty_{key}\} \in {}^{*}Penalties_{key}} penalty_{key} \quad (2)$$

Onde

$$^{*}\ Penalties_{key} = \{\ (time,\ penalty) \in Penalties_{key} \mid time + WINDOW\_SIZE \geq start\_time\} \quad (3)$$

Quando um alerta é processado, a penalidade é adicionada ao conjunto de penalidades da chave *key* passada na mensagem do alerta, ou uma chave global caso o escopo do alerta seja global. No Código 2, é apresentado um exemplo de uma mensagem de alerta.

Vale ressaltar que a eficiência do sistema de penalidades depende completamente das políticas de acesso e do monitoramento constante. No sistema proposto, o Envoy centraliza os acessos e registra eles no OpenSearch. Isso é um bom cenário para o sistema de penalidades.

4.3 Requisição de acesso

Para acessar o serviço, o sujeito deve enviar uma requisição de acesso ao Envoy, que, por sua vez, irá avaliar a requisição com informações externas e, em seguida, realizará a decisão de acesso: autorizar ou rejeitar. Nas Figuras 3 e 4, são ilustrados as interações das entidades em uma requisição de acesso bem sucedida e em uma requisição de acesso rejeitada. O fluxo principal para os dois diagramas é o mesmo, a diferença está na decisão tomada pelo filtro Lua.

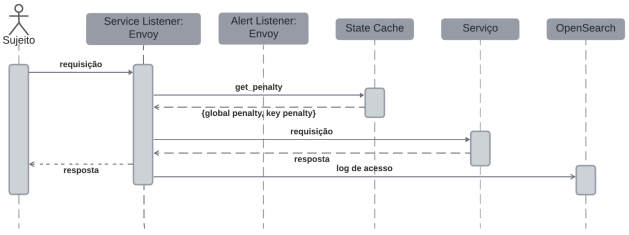

Figura 3. Diagrama de sequência para requisição de acesso bem sucedida.

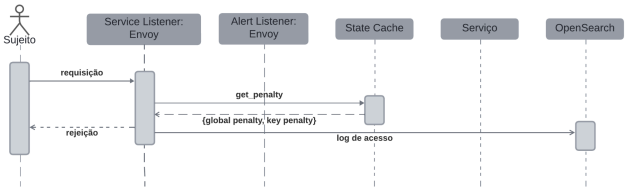

Figura 4. Diagrama de sequência para requisição de acesso rejeitada.

A descrição do fluxo é a seguinte. Inicialmente, o sujeito autenticado realiza uma requisição para o Service Listener do Envoy. Esse listener encaminha a requisição para o filtro Lua que executa o script *access_control.lua*. Nesta etapa, o script calcula as penalidades do sujeito (Pen_{sub}) e serviço (Pen_{global}) através da estratégia discutida na Seção 4.2, onde *sub* é o URI que identifica o sujeito passado no certificado X.509, e *global* é a chave do escopo global. O *WINDOW_SIZE* utilizado foi de 300 segundos. Deste modo, todas as entidades envolvidas na comunicação estão sendo verificadas antes da autorização ser concedida, como é definido no modelo ZT.

[8] https://github.com/davihsg/tcc/blob/main/envoy/lua/access_control.lua

Caso algumas das penalidades ultrapasse um limite pré-definido (Pen_{lim}), o acesso é negado (Figura 4) e o Envoy retorna uma mensagem com o código 429 (*Too Many Requests*). Caso contrário, o acesso é autorizado e a requisição é encaminhada para o serviço normalmente (Figura 3).

Após processar a requisição, o Service Listener gera um log de acesso e envia de forma assíncrona para o OpenSearch. O formato padrão dos logs do Envoy não possibilita a extração de dados para consulta, e, por isso, foi adicionado uma configuração do formato dos logs. Essa configuração está descrita no Código 1.

```
access_log:
- name: envoy.access_loggers.file
  typed_config:
    "@type":
type.googleapis.com/envoy.extensions.access_loggers.file.v3.FileAcc
essLog
    path: "/dev/stdout"
    log_format:
      json_format:
        start_time: "%START_TIME%"
        method: "%REQ(:METHOD)%"
        path: "%REQ(X-ENVOY-ORIGINAL-PATH?:PATH)%"
        protocol: "%PROTOCOL%"
        response_code: "%RESPONSE_CODE%"
        response_flags: "%RESPONSE_FLAGS%"
        bytes_received: "%BYTES_RECEIVED%"
        bytes_sent: "%BYTES_SENT%"
        duration: "%DURATION%"
        upstream_service_time:
"%RESP(X-ENVOY-UPSTREAM-SERVICE-TIME)%"
        x_forwarded_for: "%REQ(X-FORWARDED-FOR)%"
        user_agent: "%REQ(USER-AGENT)%"
        request_id: "%REQ(X-REQUEST-ID)%"
        authority: "%REQ(:AUTHORITY)%"
        upstream_host: "%UPSTREAM_HOST%"
        uri: "%DOWNSTREAM_PEER_URI_SAN%"
        listener: "service"
        downstream_remote_address: "%DOWNSTREAM_REMOTE_ADDRESS%"
```

Código 1. Configuração dos logs no Envoy.

Os campos monitorados pelo sistema são: *start_time*, *duration*, *response_code*, *uri*. Porém, os demais campos são fundamentais para dar contexto ao log e isso ajuda em casos de auditoria. Coletar o máximo de informações também é um princípio do ZT.

Esse esquema de acesso segue o modelo de acesso definido em [5] e discutido na Seção 2.1, onde o PEP, neste caso o Service Listener, encaminha a solicitação para o PDP, o filtro Lua que avalia a requisição e determina o acesso (ou não) - o filtro Lua implementa o PE e o PA.

4.4 Processamento de alerta

No contexto de ZTA, monitoramento constante é um dos pilares fundamentais para garantir segurança e reduzir o tempo de resposta a anomalias. Com vigilância contínua dos componentes do sistema, irregularidades e ameaças podem ser identificadas em tempo real, o que reduz o tempo de resposta. Nesse cenário, os alertas são o primeiro ponto de identificação de anomalias. Eles são gerados automaticamente e notificam que algum comportamento anômalo foi identificado em um dos recursos monitorados. Ciente disso, o sistema proposto utiliza o plugin de alertas do OpenSearch, o OpenSearch Alerting, para monitorar os logs de acesso e gerar alertas a partir das políticas de acesso. Para cada uma das políticas descritas na Seção 4.2, foram definidos monitores e *triggers* conforme a Tabela 1. O valor limite de cada *trigger* foi definido com base na experiência e são apenas exemplos, por isso, em uma implantação em produção, estes valores devem ser ajustados considerando os fatores específicos do sistema.

Tabela 1. Descrição dos triggers para cada política de acesso.

Política	Tipo de política	Monitor	Threshold	Intervalo de tempo	Severidade
Quantidade de requisições	Sujeito	Envoy	>200	10 minutos	4
Duração das requisições	Sujeito	Envoy	>2000	10 minutos	3
Quantidade de requisições negadas	Sujeito	Envoy - 429	>40	60 minutos	2
Quantidade alertas	Sujeito	Alerts	>10	240 minutos	1
Consumo de CPU	Global	CPU - Service	>80%	10 minutos	3
Consumo de memória	Global	Memory - Service	>70%	10 minutos	2

A sequência definida na Figura 5 é a seguinte. Os alertas gerados pelos *triggers* do OpenSearch notificam o Envoy, que possui um listener dedicado para processá-los, o Alert Listener. A mensagem enviada com o alerta contém informações relevantes como quem causou o alerta, qual o nível de gravidade, quando começou, quando terminou e se é escopo global. O Código 2 mostra um exemplo de mensagem.

```
{
  "alerts": [
    {
      "global_scope": false,
      "key": "spiffe://dhsg.com/anomalous",
      "monitor_name": "envoy-monitor",
      "period_end": "2024-09-22T02:24:21.562Z",
      "period_start": "2024-09-22T02:23:21.562Z",
      "trigger_name": "high-request-count",
      "trigger_severity": 4,
      "type": "bucket"
    },
    null
  ]
}
```

Código 2. Exemplo de mensagem de alerta.

O valor *null* é sempre passado no fim da lista de alertas devido a limitação do OpenSearch Alerting [23].

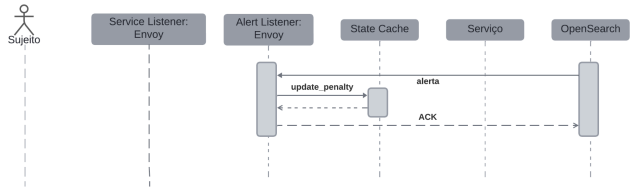

Figura 5. Diagrama de sequência para processamento de alerta.

No Alerta Listener, o script *alert.lua*[9] é executado para processar o alerta. Nessa etapa, a penalidade é definida a partir do grau de gravidade do alerta e vai de 1 a 5. Quanto maior a gravidade (próximo a 1), maior a penalidade (próximo a 5). Em seguida, o script atualiza as informações no State Cache incrementando o valor {*period_end*, *penalty*} no conjunto da chave *key*, onde *period_end* é o tempo do alerta, *penalty* é o valor da penalidade e *key* é a chave que identifica o sujeito que causou o alerta, ou a chave global, para alertas de escopo global. Ao terminar de processar o alerta, o Envoy responde com uma mensagem qualquer.

5. AVALIAÇÃO E RESULTADOS

Para os testes, foram criados três cenários: sem controle de acesso, com controle de acesso e com controle, mas sem penalidades. Em todos os cenários, um teste de carga foi realizado sob o sistema, simulando um roubo de credenciais de um sujeito legítimo, onde o invasor realizar um ataque de força bruta, um tipo de violação comum usada para sobrecarregar o sistema ou, por exemplo, para quebrar a segurança de um serviço testando várias combinações de usuário e senha. A carga foi simulada através do Vegeta[10], uma ferramenta de código aberto para teste de carga HTTP com taxa constante de requisições. Durante os testes, o sujeito que interage com o sistema já está autenticado e, portanto, apresenta o certificado X.509, cujo campo URI contém suas informações de identificação.

Diante disso, os cenários visam avaliar o desempenho da solução proposta a partir de métricas como requisições legítimas negadas, uso de CPU e memória do serviço durante o ataque e acessos indevidos. Além disso, o impacto do tempo da tomada de decisão na latência de cada requisição é uma métrica importante para determinar a viabilidade da solução e também foi considerada.

5.1 Cenário 1: Sem controle de acesso

Nesse cenário, não existe nenhum mecanismo de controle de acesso sendo aplicado ao sistema, permitindo acesso irrestrito. Nesse caso, as requisições são enviadas diretamente para o serviço, sem nenhum elemento intermediário. O principal objetivo é definir uma base para avaliar o modelo de controle de acesso proposto. Isso ajuda a entender as vulnerabilidades de um ambiente sem restrições e como o presente trabalho consegue superá-las.

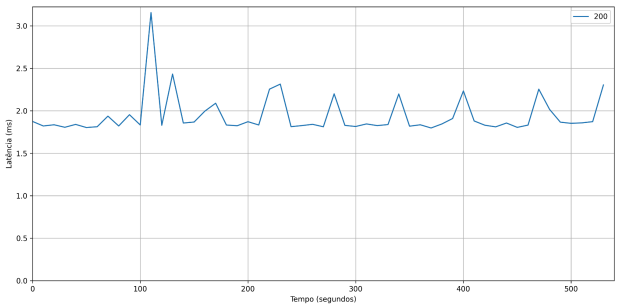

Figura 6. Latência média no Cenário 1.

[9] https://github.com/davihsg/tcc/blob/main/envoy/lua/alert.lua
[10] https://github.com/tsenart/vegeta

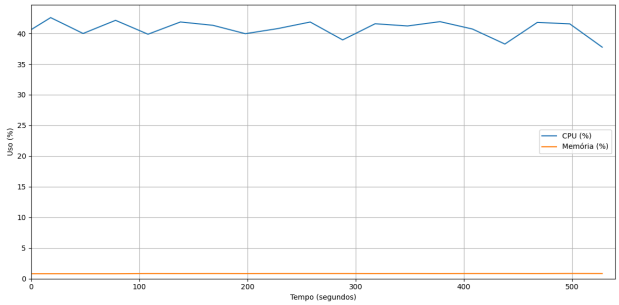

Figura 7. Uso de CPU e memória do serviço no cenário 1.

Ao longo do teste, foram realizadas 162 mil requisições ao longo de 9 minutos (540 segundos) a uma taxa de 300 requisições por segundo. Observe o comportamento da latência na Figura 6. Ela se manteve estável com alguns picos. A latência média, mediana, *p95* (95º percentil) e máxima foram de 1,941 ms, 1,826 ms, 2,234 ms, 99,746 ms, respectivamente. Na Figura 7 é apresentado o percentual de uso dos recursos do serviço em função do tempo. O consumo de CPU se manteve estável na faixa de 39% a 42%. Já o uso de memória não foi afetado e se manteve a mesma durante os testes, o que indica que o serviço é limitado pela CPU. Os próximos cenários são uma extensão deste cenário, porém com a implementação do sistema proposto.

5.2 Cenário 2: Com controle de acesso

Nesse segundo cenário, os testes foram conduzidos com as mesmas configurações do cenário anterior com a adição do sistema de controle de acesso proposto. Dessa vez, ao invés de enviar as requisições diretamente ao serviço, as requisições foram enviadas ao Envoy que autorizou ou negou a depender das penalidades do sujeito e do serviço, conforme o processo de decisão descrito na Seção 4.3. A penalidade limite Pen_{lim} definida foi 3. O objetivo do teste é avaliar o desempenho do sistema em proteger contra acessos ilegítimos de invasores.

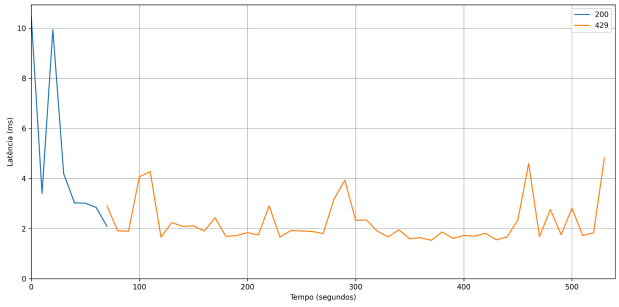

Figura 8. Latência média no Cenário 2.

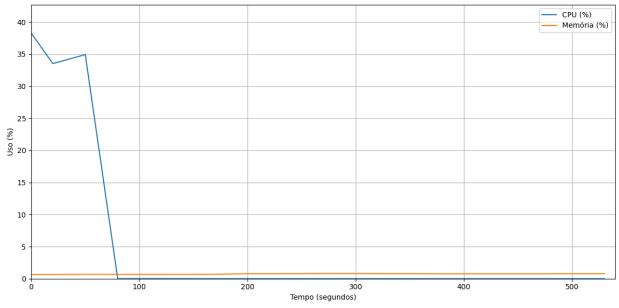

Figura 9. Uso de CPU e memória do serviço no cenário 1.

Da mesma forma do cenário anterior, os testes duraram 9 minutos (540 segundos) com 300 requisições por segundo, totalizando 162 mil requisições. Na Figura 8, é possível observar o controle de acesso sendo aplicado após 1 minuto do início dos testes. Nesse momento, dois dos alertas definidos na Tabela 1 foram acionados: quantidade de requisições, com penalidade 2, e duração das requisições, com penalidade 3. A soma das penalidades dos dois alertas ultrapassou a Pen_{lim}, e, a partir desse momento, as requisições de acesso foram negadas pelo Envoy. Após mais 1 minuto, o alerta para quantidade de requisições negadas, com penalidade 4, foi acionado, e o acesso continuou sendo negado, uma vez que a penalidade do sujeito ficou 9 (2 + 3 + 4). Em relação aos recursos do serviço, é possível observar na Figura 9 que, antes dos alertas serem acionados, o consumo de CPU estava em torno de 35%. Após 1 minuto, quando os alertas foram acionados, o uso de CPU caiu para 0%. Para a memória, não houve mudanças e permaneceu constante. Além disso, como a carga sobre o sistema foi rapidamente mitigada, os alertas para o consumo de recursos não foram acionados.

Esses resultados mostram a eficácia do modelo de controle de acesso em proteger o sistema contra acessos ilícitos. Com 1 minuto, a anormalidade foi identificada e as requisições começaram imediatamente a serem rejeitadas. Os efeitos no serviço também foram imediatos. Diferentemente do primeiro cenário, o mecanismo de controle de acesso protegeu o serviço contra uma possível sobrecarga causada por um invasor.

5.3 Cenário 3: Com controle de acesso sem penalidades

Nesse terceiro cenário, os testes foram realizados no mesmo formato do segundo cenário, onde as requisições de acesso foram feitas ao Envoy, porém, agora, o script *access_control.lua*, após consultar o State Cache e calcular as penalidades do sujeito e do serviço, ignora a penalidade limite definida e sempre permite o acesso. O objetivo desse cenário foi determinar o tempo médio da decisão de acesso do modelo proposto a partir dos dados do primeiro cenário. Isso foi fundamental para determinar a viabilidade da solução proposta.

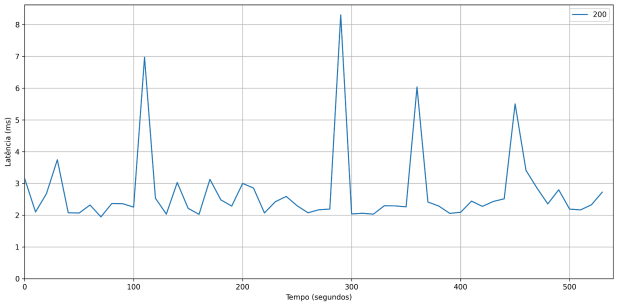

Figura 10. Latência média no Cenário 1.

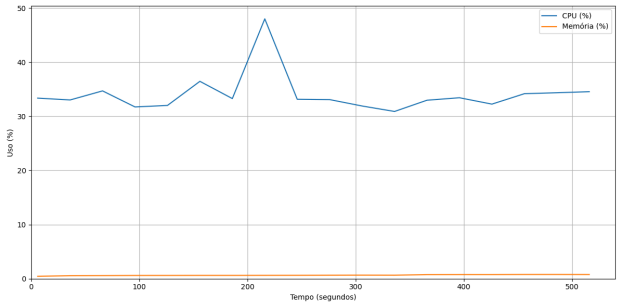

Figura 11. Uso de CPU e memória do serviço no cenário 1.

Mais uma vez, os testes duraram 9 minutos com uma taxa de 300 requisições por segundo. Como o Envoy sempre permitiu o acesso, independente da penalidade, todas as requisições foram encaminhadas para o serviço e seguiram a sequência de um acesso bem sucedido definido na Figura 3. Nesse cenário, a latência média, mediana, *p95* e máxima foram de 2,731 ms, 1,93 ms, 2,41 ms, 1,016 s, respectivamente. Em relação ao primeiro cenário, o impacto do Envoy com o mecanismo de controle de acesso não foi tão significativo no tempo de resposta médio, o qual aumentou 790 ms, enquanto que a mediana e p95 aumentaram apenas 104 ms e 176 ms, nessa ordem. Entretanto, através da Figura 10, é possível notar instabilidades na latência mais frequentes, quando comparadas às instabilidades identificadas no primeiro cenário. Finalmente, o uso de CPU e memória descritos na Figura 11 seguem a tendência do primeiro cenário, o que era esperado.

Diante disso, apesar das instabilidades, o tempo de resposta permaneceu dentro de limites aceitáveis, sem comprometer a performance geral do sistema. Esses resultados reforçam a viabilidade do sistema demonstrando sua capacidade mesmo em altas demandas.

6. CONCLUSÃO

Este trabalho apresentou um sistema de controle de acesso dinâmico que utiliza um mecanismo de penalidades para tomar a decisão de acesso em tempo real baseada em logs. O sistema explorou as capacidades do Envoy, para controlar o acesso e implementar a tomada de decisão, e do OpenSearch, para processar os logs de acesso ao sistema e identificar anomalias. O modelo segue os princípios da ZTA e se mostrou eficaz em preservar a segurança do sistema sob ataque de um usuário malicioso. Os resultados indicam a importância de utilizar políticas de controle de acesso e também do monitoramento das atividades dentro do sistema através dos logs, muitas vezes deixado de lado nesse aspecto. É importante destacar que o verdadeiro poder do modelo reside na integração eficaz de ferramentas amplamente empregadas em ambientes de ZTA, o OpenSearch e o Envoy. O sistema está disponível publicamente em um repositório do Github[11] para referência e colaboração.

Como trabalhos futuros, é fundamental realizar testes em ambientes mais realistas com usuários reais interagindo com o sistema. Outro ponto de melhoria é utilizar a funcionalidade de Anomaly Detection do OpenSearch para aumentar ainda mais a robustez das políticas de acesso. Além disso, ter uma instância

[11] https://github.com/davihsg/tcc

externa ao Envoy, como o Open Policy Agent[12], realizando a tomada de decisão de acesso diminui a sobrecarga de responsabilidade do Envoy e separa o PEP e o PDP em dois componentes diferentes, o que é desejável. Ainda, forçar a reautenticação através do cálculo de penalidade tornaria o sistema ainda mais seguro.

Este trabalho é uma evolução do trabalho desenvolvido em [24], onde foi modelado um sistema de controle de acesso com o Envoy para uma API do SmartCampus[13] da Universidade Federal de Campina Grande (UFCG).

7. AGRADECIMENTOS

Primeiramente, gostaria de agradecer a meus pais e a minha irmã, por sempre me apoiarem durante a graduação. Não teria chegado até aqui sem o suporte deles. Agradeço também aos meus avós por terem lutado pela educação dos meus pais. Aos amigos que fiz durante a graduação, agradeço por estarem ao meu lado durante esses longos anos. Em especial, gostaria de agradecer ao meu amigo de longa data, Abraão, por ter contribuído nas discussões deste trabalho.

Finalmente, agradeço ao professor Andrey Brito pela orientação, paciência e confiança ao longo do desenvolvimento deste trabalho e também aos projetos "Pesquisa, desenvolvimento & inovação em inteligência de negócios aplicada à gestão acadêmica" (CEEI/UFCG) e "Ecossistema baseado em IoT para gerência de água e energia" (FAPESQ 09/2021, Demanda Universal) que apoiaram a infraestrutura de computação e monitoramento do campus.

8. REFERÊNCIAS

[1] XIAO, Y.; JIA, Y.; LIU, C.; CHENG, X.; YU, J.; LV, W. Edge computing security: state of the art and challenges. Proceedings of the IEEE, v. 107, n. 8, p. 1608–1631, 2019.

[2] ATHENA, J.; SUMATHY, V. Survey on Public Key Cryptography Scheme for Securing Data in Cloud Computing. Circuits and Systems, v. 8, n. 3, 2017.

[3] FELDMAN, D.; FOX, E.; GILMAN, E. et al. Solving the Bottom Turtle — a SPIFFE Way to Establish Trust in Your Infrastructure via Universal Identity. 1. ed. [S.l.]: SPIFFE, 2020. Disponível em: <https://thebottomturtle.io>. Acesso em: 22 set. 2024.

[4] IBM. Cost of a Data Breach Report 2024. 2024. Disponível em: <https://www.ibm.com/security/data-breach>. Acesso em: 25 set. 2024.

[5] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (NIST). Zero Trust Architecture. NIST Special Publication 800-207. Gaithersburg, MD, EUA: NIST, 2020. Disponível em: <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207.pdf>. Acesso em: 23 set. 2024.

[6] SINGH, C.; THAKKAR, R.; WARRAICH, J. IAM Identity Access Management—Importance in Maintaining Security Systems within Organizations. European Journal of Engineering and Technology Research, v. 8, n. 4, p. 30–38, ago. 2023. DOI: https://doi.org/10.24018/ejeng.2023.8.4.3074.

[7] PACE, A. Identity management. Journal of Physics: Conference Series, [S.l.], v. 119, p. 012002, 2008.

[8] DE MELLO, Emerson Ribeiro; DE CHAVES, Shirlei Aparecida; DA SILVA, Carlos; WANGHAM, Michelle Silva; BRITO, Andrey; HENRIQUES, Marco Aurélio Amaral Henriques. Autenticação e autorização: antigas demandas, novos desafios e tecnologias emergentes. In: DOS SANTOS, Carlos Raniery Paula; PRIESNITZ FILHO, Walter; DA SILVA GONÇALVES, Paulo André; HENKE, Marcia (eds.). Minicursos do XXII Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais. Sociedade Brasileira de Computação, 2022. p. 1-50.

[9] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (NIST). Guide to Attribute Based Access Control (ABAC) Definition and Considerations. NIST Special Publication 800-162. Gaithersburg, MD, EUA: NIST, 2012. Disponível em: <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-162.pdf>. Acesso em: 23 set. 2024.

[10] MHETRE, N. A.; DESHPANDE, A. V.; MAHALLE, P. N. Experience-Based Access Control in UbiComp: A New Paradigm. Journal of Computer and Communications, v. 10, n. 1, 2022.

[11] CNCF. Graduated and incubating projects. 2024. Disponível em: <https://www.cncf.io/projects/>. Acesso em: 26 set. 2024.

[12] ENVOY. What is Envoy. Disponível em: <https://www.envoyproxy.io/docs/envoy/latest/intro/what_is_envoy>. Acesso em: 25 set. 2024.

[13] ENVOY. Life of a request. Disponível em: https://www.envoyproxy.io/docs/envoy/latest/intro/life_of_a_request. Acesso em: 25 out. 2024.

[14] OPENSEARCH. Introduction to OpenSearch. Disponível em: <https://opensearch.org/docs/latest/getting-started/intro/>. Acesso em: 26 set. 2024.

[15] AMAZON WEB SERVICES. Visão geral do armazenamento em cache. Disponível em: https://aws.amazon.com/pt/caching/. Acesso em: 01 out. 2024.

[16] DB-ENGINES. DB-Engines Ranking of Key-Value Stores. Disponível em: https://db-engines.com/en/ranking/key-value+store. Acesso em: 01 out. 2024.

[17] IBM. O que é Redis? Disponível em: <https://www.ibm.com/br-pt/topics/redis>. Acesso em: 26 set. 2024.

[12] https://www.openpolicyagent.org/

[13] https://github.com/ufcg-lsd/smartufcg-energia

[18] REDIS. Get started with Redis. Disponível em: <https://redis.io/docs/latest/get-started/>. Acesso em: 26 set. 2024.

[19] REDIS. What is Redis. [repositório] GitHub. Disponível em: <https://github.com/redis/redis?tab=readme-ov-file#what-is-redis>. Acesso em: 26 set. 2024.

[20] ENVOY. Lua HTTP Filter. Disponível em: <https://www.envoyproxy.io/docs/envoy/latest/configuration/http/http_filters/lua_filter#config-http-filters-lua>. Acesso em: 26 set. 2024.

[21] FREITAS, Lucas Lino do C.; COELHO, Kristtopher K.; NOGUEIRA, Michele; VIEIRA, Alex Borges; NACIF, José Augusto M.; SILVA, Edelberto Franco. Controle de Acesso Sensível ao Contexto e Zero Trust para a Segurança em E-Health. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES E SISTEMAS DISTRIBUÍDOS (SBRC), 42. , 2024, Niterói/RJ. Anais [...]. Porto Alegre: Sociedade Brasileira de Computação, 2024 . p. 770-783. ISSN 2177-9384. DOI: https://doi.org/10.5753/sbrc.2024.1471.

[22] DO AMARAL, T. M. S.; GONDIM, J. J. C. Integrating Zero Trust in the cyber supply chain security. In: WORKSHOP ON COMMUNICATION NETWORKS AND POWER SYSTEMS (WCNPS), 2021, Brasília, Brasil. Proceedings… Brasília: IEEE, 2021. p. 1-6. Disponível em: <https://doi.org/10.1109/WCNPS53648.2021.9626299>. Acesso em: 25 set. 2024.

[23] OPENSEARCH PROJECT. Alerting issue #59. [repositório] GitHub. Disponível em: <https://github.com/opensearch-project/alerting/issues/59>. Acesso em: 26 set. 2024.

[24] ALVES, Brenda Louisy Morais. Controle de acesso em API do projeto SmartCampus: uma abordagem usando SPIRE em conjunto com serviços de proxy. Disponível em: http://lattes.cnpq.br/2960030172402811. Acesso em: 21 set. 2024.