Atribuição de Métrica de Confiança para Tráfego de APIs em Arquitetura de Microsserviços

Ademario Vitor Costa[1], **Guilherme Vinicius de Oliveira Soares**[1], **Lorena Mamede Botelho**[1], **Rafael Jordão Clemente**[1], **Sylviane Silva Do Nascimento De Oliveira**[1]

[1]Instituto de Computação – Universidade Federal do Rio de Janeiro (UFRJ)
Rio de Janeiro – RJ – Brasil

{adeluigi,lorenamb,rafaeljclemente,sylvianesno}@ic.ufrj.br

Abstract. *This article proposes a methodology for monitoring microservices communication, focusing on REST APIs under the HTTP protocol. The model introduces indicators: client reputation and application health. For DDoS attacks, the methodology assesses request frequency using a Poisson distribution and Kullback-Leibler divergence. Regarding data breaches, the model examines response size variation, triggering reputation adjustments. Initial trust is determined by the ecosystem's confidence level. The proposed methodology combines statistical analysis and security measures for monitoring microservices, providing a framework for potential threat responses. Practical validation and fine-tuning are recommended for specific deployment scenarios.*

Resumo. *Este artigo propõe uma metodologia para monitorar a comunicação entre microsserviços, com foco em APIs REST sob o protocolo HTTP. O modelo introduz indicadores: reputação do cliente e saúde da aplicação. Para ataques de negação de serviço (DDoS), a metodologia avalia a frequência de requisições usando uma distribuição de Poisson e divergência de Kullback-Leibler. Em relação a violações de dados, o modelo examina a variação no tamanho da resposta, desencadeando ajustes na reputação. A confiança inicial é determinada pelo nível de confiança do ecossistema. A metodologia proposta combina análise estatística e medidas de segurança para monitorar microsserviços, oferecendo um quadro para respostas a possíveis ameaças. A validação prática e ajustes finos são recomendados para cenários de implantação específicos.*

1. Introdução

A Arquitetura de Microsserviços tem ganhado cada vez mais destaque na preferência dos desenvolvedores em alternativa à antiga Arquitetura Monolítica (Figura 1), isso porque ela atende aos principais requisitos DevOps de entrega contínua [Newman 2021]. Empresas como Netflix e Amazon já migraram da arquitetura monolítica para a de microsserviços devido à rápida entrega e à maior escalabilidade oferecida pelos microsserviços. Cada vez mais empresas e setores adotam essa abordagem, inclusive setores que oferecem recursos essenciais para a população. O governo brasileiro já tem usado esta alternativa em algumas das suas principais instituições como o Tribunal de Contas da União e o Ministério do Planejamento [Luz et al. 2018]. A grande vantagem na adoção é a heterogeneidade de tecnologias que podem ser usadas na construção de pequenas unidades autônomas de serviço, o que é um ponto relevante na escolha em relação às aplicações monolíticas

[Luz et al. 2018]. Todas as decisões de implementação são focadas no domínio da aplicação e, por isso, atendem melhor à variedade de regras de negócio de diferentes setores. Detalhes de implementação ficam escondidos pelas Application Programming Interfaces (API), que oferecem um protocolo comum para a comunicação entre diferentes serviços. Quanto mais setores optam por esta arquitetura, mais dados teremos circulando por APIs e um volume cada vez maior de serviços.

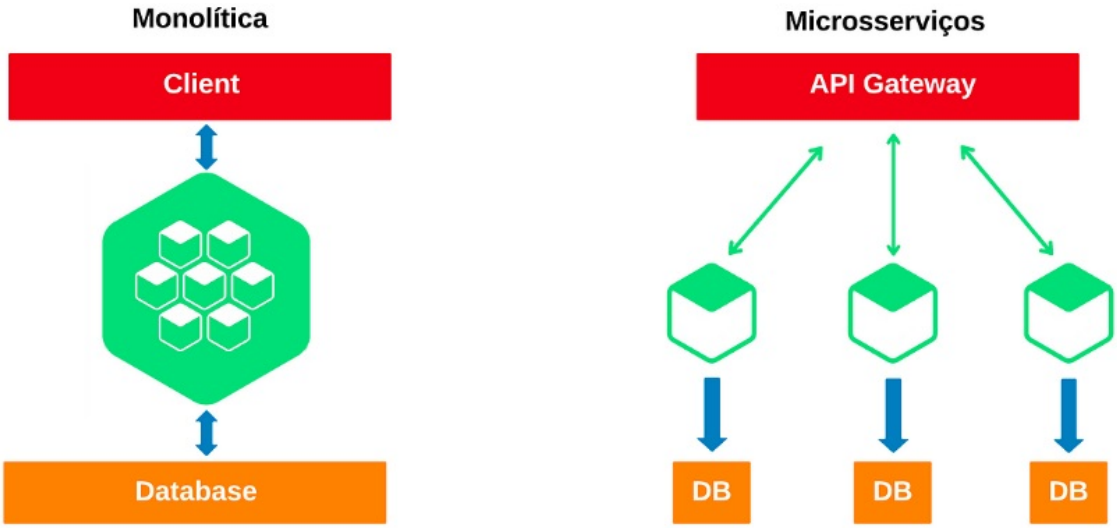

Figure 1. Comparação entre Arquitetura Monolítica e Arquitetura de Microsserviços

A comunicação por APIs permite que os serviços sejam desacoplados. Como resultado, mudanças na implementação causam menos impacto nos demais, o que facilita o processo de entrega, não sendo incomum que uma aplicação seja dividida em múltiplos microsserviços [Mateus-Coelho et al. 2021]. Por isso, um número grande de endpoints é presente nesta arquitetura, oferecendo uma superfície maior de ataque [Newman 2021], o que gera dificuldade em monitoramento e detecção. Por exemplo, não é difícil que uma API esquecida seja vulnerável e apresente brechas na exposição de dados (Data Breach), permitindo que um atacante consiga informações não autorizadas [Sowmya et al. 2023]. Outra possibilidade, são ataques de Denial of Service (DoS), que buscam derrubar a aplicação ao realizar uma quantidade de requisições que supere a capacidade de atendimento do servidor, deixando os serviços fora do ar. Esse ataque também possui uma variação distribuída o Distribuited Denial of Service (DDoS), que possui o mesmo objetivo, a única diferença é que o ataque é realizado a partir de mais de um cliente, sendo distribuído em servers ou netbots, que podem hosts que foram comprometidos em um ataque prévio que garantiu ganho de acesso indevido ao atacante. Não é surpresa que ganho de acesso indevido, exposição de dados e negação de serviço estejam entre os TOP 10 riscos da OWASP [OWASP Top 10 API Security Risks 2023]. Em um cenário em que a Arquitetura de Microsserviços se faz cada vez mais presente, e trafegam cada vez mais dados entre os microsserviços, essas preocupações se tornam necessárias para garantir a proteção dos usuários que a utilizam. É fundamental que os desenvolvedores e a equipe de segurança tenham formas de monitorar em tempo real os endpoints para detectar e se recuperar mais rapidamente dessas falhas. [Chandramouli 2019]

Este trabalho busca desenvolver métricas que auxiliem no monitoramento de aplicações que fazem uso de APIs, a ideia é aproveitar métricas que foram desenvolvidas em outros contextos e trazê-las para o cenário de Microsserviços. Com base na coleta dos dados das requisições, propomos uma métrica para avaliar a confiança no cliente e outra para a saúde de aplicação, de modo a auxiliar nas tomadas de decisão. Usaremos análise estatísticas com o Teorema Central do Limite, distribuição de Poisson e divergência de Kullback-Leibler [Kullback 1959] para definir o perfil da aplicação durante ataques DoS, DDoS e violação de dados, identificando endpoints que estão sob ataque. O objetivo é trazer um ferramental para a implementação de mais camadas de segurança na arquitetura de microsserviços, incrementando o processo de tracing ao oferecer monitoramento em tempo real para APIs. Este trabalho se diferencia por monitorar o tráfego a nível de aplicação, mais especificamente as requisições HTTP, enquanto outros trabalhos encontrados na área optaram por uma abordagem a nível de rede. Além disso, a maior preocupação identificada nestes trabalhos, na área de segurança de microsserviços, foi em relação a métodos de controle de acesso, e não no monitoramento de possíveis explorações de vulnerabilidades. Enriqueceremos as propostas anteriores ao incluir mais parâmetros nas métricas, o que permitirá mais precisão na identificação de ataques e anomalias.

Esse artigo está dividido em: a seção 2 traz uma análise resumida de trabalhos relacionados. A seção 3 descreve os parâmetros necessários para realização da modelagem matemática e detalha a infraestrutura projetada. Na seção 4, o modelo é descrito através da elaboração de cada métrica envolvida e os cálculos necessários, a seção é encerrada com a formalização de todos os parâmetros para geração dos indicadores de cliente e aplicação. A seção 5 detalha os experimentos e resultados obtidos com o dataset escolhido e testes de carga para simulação de ataques. A seção 6 conclui o trabalho e discute possíveis futuros trabalhos.

2. Trabalhos Relacionados

A dificuldade em monitorar e proteger APIs na Arquitetura de Microsserviços está principalmente no grande número de microsserviços que é colocado em produção, havendo uma quantidade significativa de endpoints. O volume de clientes interagindo com cada endpoint é ainda maior, podendo ser tanto usuários quanto serviços. Montar um perfil de ataque para esses clientes é complexo, pois envolve encontrar parâmetros adequados para identificá-los, IPs podem ser falsificados ou mascarados e as conexões podem mudar constantemente. Há dificuldade em dicernir um comportamento atípico de uma mudança legítima no tráfego da aplicação.

Mateus-Coelho et al. [Mateus-Coelho et al. 2021] descrevem os principais requisitos de segurança na implementação de microsserviços, citando dentre muitos aspectos a proteção de APIs com uso de SSL/TLS, SAML, OpenID, API keys e HMAC dentre outros mecanismos de controle de acesso. Porém, a discussão é limitada apenas às alternativas de Autenticação e Autorização, não descrevendo métodos que permitam identificar comportamentos nocivos dos clientes no uso dos endpoints.

Chatterjee e Prinz [Chatterjee and Prinz 2022] buscaram, através de um estudo de caso, criar uma implementação de controle de acesso para APIs, na arquitetura de microsserviços. O foco principal foi trazer o uso de plataformas de controle de acesso

como o Keycloak, integrado ao Spring Security Framework e SAML para criar uma forma de autenticação e autorização mais segura. Uma infraestrutura digital foi apresentada para dificultar acesso não-autorizado ao backend da aplicação. O combate a DoS/DDoS e outros tipos de ataque foi delegado totalmente à implementação do Spring Security Framework, não sendo oferecido uma forma de monitoramento.

Sowmya et al. [Sowmya et al. 2023] propõem o uso de RandomForest para monitoramento frequente de API a nível de aplicação, com uso de compressor recursivo para extrair dados dos logs. Pruned Exact Linear Time é usado para detecção de ataques DoS. O problema em relação a usar os métodos propostos está principalmente no desafio de desempenho, além de ser condicionado a um balanceamento muito específico dos dados, não se adequando a qualquer ambiente de produção.

Macedo et al. [Macedo et al. 2022] tratam da elaboração de uma métrica de confiança para avaliar a taxa de dados entre dispositivos IoT. O modelo combina a perspectiva de rede e de aplicação usando blockchain e técnicas de Teoria da Informação. Usamos as técnicas estatísticas e de teoria da informação descrita neste trabalho para trazer para o contexto de Microsserviços.

A maioria dos trabalhos que abordam o tema focam em configurações rule base ou signature-based (IDS e IPS), além de tratarem de soluções de monitoramento de tráfego a nível de rede, o que não é tão adequado quando se trata de monitorar a interação dos clientes com as APIs. Não é oferecido uma estrategia de monitoramento em tempo real, a elaboração de métricas que forneçam uma visão geral do estado da aplicação são aplicadas a outros cenários, que não o de Microsserviços. Portanto, este trabalho busca preencher esta lacuna e criar métricas que ajudem a identificar ataques durante seu acontecimento.

3. Coleta de informações

A interface mais comumente implementada na comunicação de microsserviços são REAST APIs, que são regidas pelo protocolo HTTP. Esse protocolo oferece uma série de métodos e parâmetros para a troca de informações na camada de aplicação. Nossa análise consiste em usar a informação dos headers dessas requisições para mapear as características dos envolvidos na comunicação. Coletamos os seguintes dados da requisição da aplicação:

- IP do cliente que está tentando se comunicar com a API

- Endpoint da API

- Tamanho da Resposta em bytes

- TimeStamp da requisição

Durante a comunicação do cliente com um endpoint, o cliente precisa passar por um gateway que será responsável por localizar o serviço requerido. Na arquitetura de Microsserviço, a aplicação é fragmentada em múltiplas instâncias usualmente com uso de containers, no qual cada container contém uma cópia do microsserviço e um microsserviço poderá expor múltiplos endpoints. Por isso, toda implementação da arquitetura conta com esse tipo de proxy reverso para ajudar a realizar a localização do serviço desejado e também direcionar a resposta para o cliente.

Aproveitando a característica dessa infraestrutura, a nossa coleta é projetada para

ser realizada nesse ponto de contato, no qual um módulo responsável por extrair os dados da requisição é colocado, a esse módulo chamamos de Coletor de Requesições. A coleta é feita para cada requisição que chega e sai do gateway. Um esquema ilustrando essa infraestrutura pode ser visualizado na Figura 2.

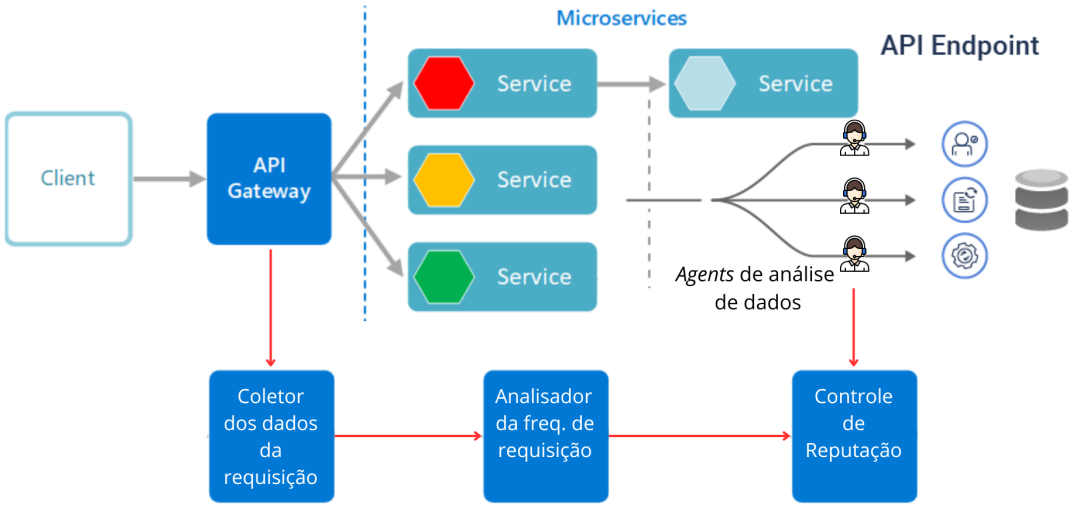

Figure 2. Esquema da coleta de informações

4. Modelo

No modelo desenvolvido, escolhemos criar dois tipos de indicadores: a reputação do cliente (T_c) e a saúde da aplicação (T_a), com base no relatório de interação com os endpoints requisitados, ambos estão entre 0 e 1. Mapeamos o perfil da aplicação durantes os principais ataques e para cada um, calculamos as métricas que atualizam cada indicador.

4.1. Confiança Inicial

Para um endpoint, o estado inicial é configurado pelo próprio operador, como regra geral, esse valor sempre vai ser máximo (1.0), representando que o sistema está saudável.

O valor de reputação inicial (C_1) de um cliente expressa o valor inicial de confiança na primeira conexão com o serviço. É importante relembrar que na comunicação a nível de aplicação não temos como guardar um histórico do cliente a longo prazo, pois seus dados como IP e User Agent podem ser facilmente modificados, por isso, assumimos que a atualização da reputação é temporária e limitada ao tempo de sessão daquele IP associado. Definimos como primeira comunicação o início de uma sessão, e toda a análise individual do cliente é a partir de sua sessão.

Para o cálculo de C_1, estabelecemos o conceito de grau de confiança do ecossistema, que é basicamente o quão provável é um cliente, que interage com tal aplicação (ecossistema), agir maliciosamente. Esse grau de confiança basicamente pode ser ilustrado pela quantidade de vulnerabilidades e exploração destas na aplicação. Serviços que têm um histórico de vulnerabilidade e passaram, ou estão passando, por ataques vão

possuir maior probabilidade de que um novo cliente também seja malicioso. Esta métrica é usada para medir a saúde da aplicação como um todo e evitar que durante a exposição de uma vulnerabilidade mais atacantes se aproveitem da brecha, o que pode ajudar na contenção de ataques DDoS.

Para estabelecer esse grau de confiança, buscamos o histórico de reputação dos clientes que já interagiram com o sistema em um intervalo de tempo configurado pelo operador. Deste intervalo, buscamos grupos de n reputações de modo que seja possível extrair um mínimo de 30 amostras. Para cada amostra, extraímos a média amostral e, pelo Teorema Central do Limite, temos uma distribuição normal das médias. Buscamos então pela probabilidade da reputação média dos usuários ser acima de 0.8 (limiar de confiança alta). Usamos essa probabilidade para gerar um valor aleatório no intervalo de 0.5 (limiar de confiança) e 1.0. Isso quer dizer que se a tendência de uma reputação alta for grande, o usuário terá maior probabilidade de iniciar com uma reputação alta.

O cálculo dessa métrica fica dedicado ao módulo de Analisador de Frequência de Requisição que fica responsável por analisar os dados que chegam ao Coletor de Requisições e identificar a chegada de novos clientes.

4.2. Mapeamento do perfil DoS/DDoS

Determinamos que no caso de ataques DoS e DDoS, o principal aspecto a ser considerado é a frequência de comunicação do cliente com o endpoint. O padrão dessa frequência é ditado principalmente pelo intervalo entre as requisições.

Por isso, analisamos a frequência de todas as requisições feitas a um determinado endpoint. A não diferenciação entre qual requisição é de qual cliente, nessa primeira análise, é importante pois ataques distribuídos de negação de serviço podem ter vários clientes se revezando no disparo de requisições. Assim, calculamos a quantidade geral de requisições que um endpoint está recebendo em um intervalo de tempo.

C_2 é a métrica que avalia a frequência de requisições em um endpoint, ela depende de um treinamento prévio para identificar a frequência usual de requisições. Assumindo as requisições como independentes, o treinamento é feito calculando a quantidade de requisições recebidas por endpoint, em cada intervalo de tempo t_i (em minutos) contido em um intervalo maior T, $t_i \in T, i \in [30, \infty)$. Para obter a taxa média de requisições por segundo, dividimos o total encontrado em cada t_i pelo seu total de segundos. Sendo L a variável aleatória das frequências encontradas, sabemos que pelo Teorema Central do Limite, a distribuição d_L é normal e, portanto, a frequência de maior probabilidade é a sua moda. Assumimos então a moda λ como referência de frequência para analisar os futuros tráfegos observados.

Durante o monitoramento, no mesmo intervalo de tamanho t_i usado na construção do modelo, é registrado o número x_j de requisições por segundo j, para $j \in \{0, t_i * 60\}$. Para cada x_j observado no intervalo t_i, calculamos sua probabilidade E_j esperada para $Poisson(\lambda)$.

Precisamos avaliar se há divergência significativa entre o observado e o estimado. A hipótese nula H_0 formulada é de que o tráfego observado segue a distribuição Poisson com a média λ estimada, e H_1 é a hipótese contrária. Usando qui-quadrado (χ^2), definimos a significância $\alpha = 0.05$, e os graus de liberdade como $df = j - 1$, sendo O_j a

probabilidade de x_j na distribuição d_X observada:

$$\chi^2 = \sum_j \frac{(O_j - E_j)^2}{E_j}$$

Se o $\chi^2 > \chi^2{}_{critico}$, então há anomalia no tráfego. Para atualizar o estado da aplicação, medimos a divergência em relação ao tráfego estimado e decaímos o valor de saúde. Usamos Kullback-Leibler para obter a entropia relativa entre as distribuições.

$$D(d_E||d_X) = \sum_j d_{E_j}(x_j) log \frac{d_{E_j}(x_j)}{d_X(x_j)}$$

O D encontrado vai estar entre $[0, \infty]$. Para mantê-lo limitado ao intervalo $[0, 1]$, para qualquer valor de $D > 1$, consideramos $D = 1$.

Em caso de ser dectado a anomalia, verificamos a quantidade de requisições por cliente no endpoint prejudicado. Os clientes com quantidade de requisições para fazer D variar em 50% são penalizados juntamente com o endpoint.

Por fim, usamos a função de ativação tangente hiperbólica, como forma de mapear o D encontrado entre -1 e 1. Isso é importante para a atualização dos indicadores, penalizaremos valores muito divergentes e atribuimos recuperação ao indicador em caso de diminuição da divergência.

$$C_2 = tanh(0.5 - D)$$

Esse cálculo fica sob a responsabilidade do módulo Analisador de Frequência de Requisição, que verifica os dados que chegam no Coletor de Requisições em busca de frequências incomuns. Ao terminar o cáclculo, a componente é enviada para o módulo de Controle de Reputação.

4.3. Mapeamento do Perfil de Violação de Dados

Nesse tipo de ataque, a principal característica que identificamos é a variação do tamanho da resposta da requisição em relação àquela prevista inicialmente, o atacante recebe como resposta um valor maior do que o esperado. Logo, o parâmetro mais importante na construção da métrica é o tamanho de resposta da requisição. Para isso, precisamos calcular previamente o tamanho ideal previsto para comparar com o que está sendo monitorado.

Para este cálculo, dedicamos um agent para cada endpoint, que monitora as requisições de acordo com os parâmetros de controle aprendidos no treinamento.

A métrica C_3 é calculada a partir deste treino prévio para determinar o tamanho de dados esperado como resposta das requisições do endpoint. No treinamento, estudamos o tamanho que o endpoint e costuma transmitir para os clientes. Sendo S_e a variável aleatória que representa o tamanho de dados estimado, extraímos a sua distribuição d_{S_e}. De posse do resultado deste treinamento, o agent vinculado ao endpoint analisa o tamanho dos dados enviados em cada uma das k respostas de requisições observadas e, através de uma comparação com o resultado do treino, tenta identificar anomalias.

Para montar a distribuição que será usada no controle, cada tamanho de transmissão s_i observado em determinado conjunto de requisições tem o seu grau de entropia calculado:

$$I(s_i) = -logd_{S_e}(s_i)$$

Com a entropia de cada observação, é calculada a entropia esperada do grupo, ou seja, é calculada uma entropia média:

$$H(S_e) = \sum_{i \in x} d_{S_e}(s_i) log d_{S_e}(s_i)$$

Esta entropia é usada como controle para definir se um tamanho de transmissão de dados y feito por uma resposta é uma anomalia. Para cada quantidade k de respostas retornadas, calculamos a distribuição d_Y. Para determinar a divergência entre $H(S_e)$ e d_Y, usamos as amostras s_i angariadas anteriormente e aplicamos em d_Y para ver a probabilidade de s_i na distribuição d_Y observada. Com isso, podemos calcular a entropia relativa entre os dois grupos e concluir se as distribuições são aproximadas ou distantes:

$$D_2(dS_e||dY) = \sum_{i \in x} d_{S_e}(s_i) log \frac{d_{S_e}(s_i)}{d_Y(s_i)}$$

E, com isso:

$$C_3 = tanh(0.5 - D_2)$$

A divergência é significativa para ser considerada uma anomalia apenas se a entropia relativa calculada for maior que 1. O agent envia o valor encontrado e os clientes associados para o módulo de Controle de Reputação, que reduz os indicadores em ambos os envolvidos (endpoint e cliente). Novamente se $D > 1$ consideramos $D = 1$.

4.4. Atualização dos Indicadores

Atualização é realizada no módulo de Controle de Reputação, que recebe os relatórios enviados pelos agents e pelo Analisador de Frequência de Requisição. É importante ressaltar que no caso de clientes, a atualização é feita apenas dentro de um intervalo de tempo em que se considera que o cliente analisado ainda pode ser identificado por determinado IP.

T_c é iniciado com C_1 e T_a inicia em 1 (ou outro valor arbitrário). À medida que as métricas são calculadas, $T_{c_{k+1}}$ e $T_{a_{k+1}}$ (próximos estados) são atualizados usando os valores calculados para C_2 e C_3. Lembrando que C_2 apenas é computado para T_c em casos que o cliente é identificado em um ataque de negação e C_3 é adicionado quando o cliente atua em um ataque de violação de dados.

$$T_{c_{k+1}} = \sigma(T_c + C_2 + C_3)$$
$$T_{a_{k+1}} = \sigma(T_a + C_2 + C_3)$$

σ se refere à função sigmoid usada para que o valor de T continue limitado entre 0 e 1 após as atualizações.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Valores de T menores que 0.5 são alertas sobre atividades maliciosas, e podem em trabalhos futuros implementarem ações temporárias, como adição em blacklists ou desativação de um endpoint (em casos de vazamento de dados).

5. Resultados e Discussão

A fim de verificar o desempenho da solução proposta, é preciso encontrar formas viáveis de implementação e realizar os testes necessários. Nesse sentido, é importante decidir se serão utilizados dados reais ou dados sintéticos. No caso de dados reais, existem diversos datasets disponíveis que podem ser utilizados, principalmente em relação à ataques DoS. Já para os dados sintéticos, é apropriado construir um ambiente controlado e simular os ataques enquanto coleta os dados necessários.

Durante o prazo para elaboração deste trabalho, não foi possível incluir esta etapa, pois exigia o desenvolvimento de uma aplicação de teste que atendesse aos requisitos da arquitetura de microsserviços, além de implementar os módulos e os agents descritos, e garantir a sua instalação em um ambiente livre de enviesamento, preferivelmente remoto para que dados como latência e sobrecarga pudessem ser mensurados. Essas implementações também exigiriam a escolha de tecnologias adequadas para modelagem das fórmulas e otimização dos cálculos.

Para os cenários de simulação dos ataques DoS/DDoS, seria necessário o uso de uma ferramenta de testes de carga para replicar usuários enviando requisições em massa nos endpoints da aplicação. A ferramenta mais adequada seria a Apache JMeter [Apache jmeter 2023] que oferece as configurações necessárias para o ambiente proposto.

Para a simulação de ataques de violação dos dados, seria necessário implementar na aplicação proposta uma vulnerabilidade que permitisse simular a exploração desta por clientes configurados pela ferramenta de testes de carga.

O teste proposto deve ser executado para diferentes cenários, coletando e classificando os resultados obtidos. O intervalo e quantidade de requisições devem ser alterados para diferentes baterias de teste. Os estados de latência, disponibilidade do serviço, armazenamento dos dados, tempo de processamento, tempo de resposta aos ataques e valor dos indicadores devem ser monitorados ao final de acada execução.

6. Conclusão e Trabalhos Futuros

Apesar da solução proposta elevar o nível de segurança da aplicação como um todo, é preciso atentar-se aos desafios que ainda persistem. Dentre eles, a análise de requisições requer uma adpatação constante diante de um tráfego dinâmico. A possibilidade do IP poder ser falsificado adiciona uma camada a mais de complexidade. Por último, mas tão importante quanto os anteriores, é possível que uma checagem constante de reputação cause uma sobrecarga além do desejado, o que exigirá uma avaliação frequente para encontrar um equilíbrio entre segurança e desempenho.

No futuro, seria desejável fazer uma validação da modelagem por meio do uso de dados reais e comparar seu desempenho com quando utilizado dados sintéticos. Essa abordagem permitirá uma avaliação mais precisa da modelagem tanto em relação a situações que já ocorreram quanto em condições hipotéticas futuras, contribuindo assim para a robustez e confiabilidade da solução proposta.

Para trabalhos futuros, acreditamos que para aprimorar o entendimento do comportamento padrão do sistema, é necessário o desenvolvimento de um modelo de aprendizado de máquina. Este modelo deverá ser treinado utilizando dados oriundos do próprio sistema, garantindo uma representação mais fidedigna da realidade operacional. Criando uma ferramenta de análise preditiva altamente adaptada às características específicas do sistema em questão.

A melhoria proposta, que emprega aprendizado de máquina, é crucial para entender o comportamento padrão do sistema. Contudo, neste contexto, torna-se essencial manter um equilíbrio entre o aumento da segurança e o desempenho do sistema, assegurando que as medidas de segurança implementadas não prejudiquem o desempenho global.

References

Apache jmeter (2023). The Apache Software Foundation. [online] available: https://jmeter.apache.org/.

Chandramouli, R. (2019). Microservices-based application systems. *NIST Special Publication*, 800(204):800–204.

Chatterjee, A. and Prinz, A. (2022). Applying spring security framework with keycloak-based oauth2 to protect microservice architecture apis: A case study. *Sensors*, 22(5).

Kullback, S. (1959). *Information Theory and Statistics*. Wiley, New York.

Luz, W., Agilar, E., de Oliveira, M. C., de Melo, C. E. R., Pinto, G., and Bonifácio, R. (2018). An experience report on the adoption of microservices in three brazilian government institutions. In *Proceedings of the XXXII Brazilian Symposium on Software Engineering*, SBES '18, page 32–41, New York, NY, USA. Association for Computing Machinery.

Macedo, E. L. C., Delicato, F. C., Moraes, L. F., and Fortino, G. (2022). A two-level integrated approach for assigning trust metrics to internet of things devices. In *International Conference on Internet of Things, Big Data and Security*.

Mateus-Coelho, N., Cruz-Cunha, M., and Ferreira, L. G. (2021). Security in microservices architectures. *Procedia Computer Science*, 181:1225–1236. CENTERIS 2020 - International Conference on ENTERprise Information Systems / ProjMAN 2020 - International Conference on Project MANagement / HCist 2020 - International Conference on Health and Social Care Information Systems and Technologies 2020, CENTERIS/ProjMAN/HCist 2020.

Newman, S. (2021). *Building Microservices*. O'Reilly Media, Inc, 2nd edition.

OWASP Top 10 API Security Risks (2023). https://owasp.org/API-Security/editions/2023/en/0x11-t10/.

Sowmya, M., Rai, A. J., Spoorthi, V., Irfan, M., Honnavalli, P. B., and Nagasundari, S. (2023). Api traffic anomaly detection in microservice architecture. In *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing Workshops (CCGridW)*, pages 206–213.