Desenvolvimento e Avaliação de um Fuzzer para Teste de Segurança em APIs REST

Pedro Antonio Messias Lemos[1], **Bruno Bogaz Zarpelão**[1]

[1]Departamento de Computação – Universidade Estadual de Londrina (UEL)
Caixa Postal 10.011 – CEP 86057-970 – Londrina – PR – Brasil

pedro.antonio.messias@gmail.com, brunozarpelao@uel.br

Abstract. *With the rising use of REST APIs in contemporary systems, ensuring the security of these interfaces has become a critical challenge. A fuzzer is a tool designed to test security by introducing random inputs into a program to identify vulnerabilities. However, many fuzzing tools face limitations when applied to APIs, including inadequacy with the HTTP protocol, challenges with authentication and authorization mechanisms, and difficulties in handling complex data structures. Additionally, the precise identification of errors is challenging due to the ambiguity in API responses, the use of custom error codes, and the possibility of silent or masked errors, making vulnerability detection a complex task. In light of these issues, this project focuses on the development of a specialized fuzzer for REST APIs. The aim is to identify errors and vulnerabilities through the analysis of API responses, taking into account specifics such as endpoints, arguments, and expected return codes. The fuzzer will also address challenges like ambiguous responses, authentication, and specific validations.*

Resumo. *Com o aumento na utilização de APIs REST em sistemas contemporâneos, garantir a segurança dessas interfaces tornou-se um desafio crítico. Um fuzzer é uma ferramenta projetada para testar a segurança, introduzindo entradas aleatórias em um programa para identificar vulnerabilidades. No entanto, muitas ferramentas de fuzzing enfrentam limitações quando aplicadas às APIs, incluindo inadequação ao protocolo HTTP, dificuldades com mecanismos de autenticação e autorização, e desafios na manipulação de estruturas de dados complexas. Adicionalmente, a identificação precisa de erros é desafiadora devido à ambiguidade nas respostas das APIs, uso de códigos de erro personalizados, e a possibilidade de erros silenciosos ou mascarados, tornando a detecção de vulnerabilidades uma tarefa complexa. Diante dessa problemática, este trabalho foca no desenvolvimento de um fuzzer especializado para APIs REST. O objetivo é identificar erros e vulnerabilidades através da análise de respostas da API, considerando especificidades como endpoints, argumentos e códigos de retorno esperados. O fuzzer também abordará desafios como respostas ambíguas, autenticação e validações específicas.*

1. Introdução

No mundo contemporâneo, as tecnologias da web permeiam diversos aspectos das nossas vidas cotidianas. À medida que a dependência de aplicações web intensifica, as Interfaces de Programação de Aplicativos (APIs) assumem um papel crucial, servindo como o elo que permite a comunicação eficaz entre diferentes programas [15]. Entretanto, o

incremento na complexidade das APIs também resulta no aumento de riscos de segurança associados a elas [8].

Deste modo, a segurança de APIs emerge como um tópico de grande relevância e interesse na comunidade científica e no setor industrial [6]. Há uma diversidade de métodos empregados para salvaguardar a segurança das APIs, os quais incluem a análise estática de código, a implementação de firewalls de aplicativos da web e a utilização de *scanners* de vulnerabilidades. O *fuzzing* — uma técnica de teste automatizado que envolve o fornecimento de entradas malformadas, aleatórias ou semi-aleatórias a um programa — tem se mostrado eficaz na identificação de vulnerabilidades até então desconhecidas [4].

Contudo, ferramentas de *fuzzing* originalmente não foram desenvolvidas com um foco específico em APIs [16]. APIs possuem características intrínsecas, como mecanismos de autenticação, limitação de taxa e *logs* complexos, que muitas vezes não são adequadamente abordados por *fuzzers* tradicionais [3]. Portanto, é importante reconhecer que, devido a esses desafios, tais ferramentas frequentemente não atingem um nível de adequação satisfatório para testes em APIs [3].

O presente trabalho é motivado pela necessidade de desenvolver um *fuzzer* focado na avaliação de segurança de APIs REST. O objetivo central é criar e avaliar uma ferramenta que possa simular ataques de maneira sofisticada, considerando as peculiaridades das APIs e as limitações que *fuzzers* tradicionais possuem, tais como lidar com autenticação, analisar códigos de resposta, fornecer uma forma eficiente e personalizável de identificar vulnerabilidades e erros, bem como gerenciar diferentes formatos de entrada (parâmetros, JSON, etc.). Os experimentos e avaliações propostos serão conduzidos no CIAAgro UEL (Centro de Inteligência Artificial no Agro da UEL).

A estrutura deste documento é organizada da seguinte forma: a Seção 2 oferece a fundamentação teórica para o trabalho, onde é descrito o contexto da segurança das APIs. Posteriormente, *fuzzing* e suas técnicas existentes são detalhados. A Seção 3 descreve os objetivos do trabalho. Em seguida, a metodologia empregada no desenvolvimento do *fuzzer* especializado, abordando aspectos cruciais de design e implementação. A Seção 5 mostra o cronograma de execução do projeto. Por fim, a Seção 6 apresenta as contribuições e resultados esperados.

2. Fundamentação Teórico-Metodológica e Estado da Arte

2.1. APIs

As Interfaces de Programação de Aplicativos (APIs, do inglês Application Programming Interfaces) são conjuntos de protocolos, rotinas e ferramentas que permitem a comunicação e interação entre diferentes sistemas de software. Conforme a definição fornecida pela AWS [2], uma API atua como uma porta através da qual diferentes serviços de software podem interagir, permitindo que um sistema solicite informações de outro e receba uma resposta em um formato padronizado. Redhad et al. [14] também destaca que as APIs são cruciais para a integração de sistemas, servindo como contratos que definem como os componentes de *software* devem interagir.

A importância das APIs na arquitetura de software moderna é inegável [15] [9]. Elas desempenham um papel fundamental na construção de aplicações distribuídas, permitindo que diferentes serviços e componentes se comuniquem de forma eficiente e ma-

joritariamente segura [14]. Com a ascensão de arquiteturas baseadas em microsserviços [17] e a proliferação de dispositivos conectados na Internet das Coisas (IoT, do inglês *Internet of Things*), as APIs tornaram-se ainda mais críticas para o funcionamento eficaz dos sistemas [1].

2.1.1. APIs REST

O paradigma REST (do inglês, *REpresentational State Transfer*) estabelece um conjunto de princípios arquiteturais que têm se mostrado fundamentais para o desenvolvimento de sistemas distribuídos na web [15]. As APIs que seguem esses princípios são frequentemente denominadas APIs *RESTful*. Essas interfaces são organizadas em uma coleção de serviços Web *RESTful*, cada um encarregado de gerenciar operações *CRUD* (Criar, Ler, Atualizar, Deletar do inglês, *Create*, *Read*, *Update*, *Delete*) sobre determinados recursos [15].

Neste cenário, um "recurso" é uma entidade que pode ser disponibilizada na web, podendo ser tão variada quanto um vídeo, uma imagem ou um pedido de compra. Estes recursos são comumente acessíveis através de Identificadores de Recursos Uniformes (URIs, do inglês *Uniform Resource Identifier*), o que as torna localizáveis e manipuláveis através de protocolos de aplicação, como o *HTTP* (do inglês, *HyperText Transfer Protocol*). Cada URI que aponta para um ou mais desses recursos é conhecido como um *endpoint* de API [15].

As APIs *RESTful* adotam um conjunto padronizado de diretrizes para a implementação de métodos *HTTP*, conforme descrito a seguir [15]:

- **GET/Ler**: utilizado para acessar e recuperar informações sobre um recurso ou conjunto de recursos;
- **POST/Criar**: empregado para a criação de novos recursos;
- **PUT/Alterar**: serve para modificar um recurso previamente existente;
- **DELETE/Deletar**: aplicado para a remoção de um recurso específico;

2.2. Segurança de APIs Web

Com a evolução da arquitetura de software moderna, as APIs têm sido incorporadas em diversas soluções tecnológicas. Devido à sua ampla utilização e ao fato de estarem entre os principais vetores de ataque [9], evidencia-se a relevância de se adotar padrões de segurança rigorosos no desenvolvimento e manutenção dessas interfaces [6] [9].

A segurança das APIs é comprometida por uma série de vulnerabilidades potenciais, muitas das quais são destacadas no OWASP API Security Top 10 [7]. A OWASP (do inglês, *Open Web Application Security Project*) é uma organização sem fins lucrativos que se dedica a melhorar a segurança dos softwares e, devido à sua abordagem aberta e colaborativa, as informações e ferramentas que ela fornece são amplamente reconhecidas pela comunidade global de segurança da informação. O documento da OWASP descreve as dez principais vulnerabilidades de segurança de APIs. Essas vulnerabilidades abrangem desde falhas em mecanismos de autenticação e autorização até a exposição indevida de dados sensíveis. Tais fragilidades não apenas ameaçam a integridade dos sistemas que

dependem desses serviços, mas também podem resultar em violações de dados e impactos negativos nas operações comerciais [6]. A lista a seguir apresenta as dez principais vulnerabilidades de segurança de APIs, conforme definido pela OWASP [7] em 2023.

- **API1:2023 Broken Object Level Authorization**: trata-se de situações em que os *endpoints* de API são suscetíveis à manipulação do ID de um objeto enviado na solicitação, permitindo que atacantes acessem ou modifiquem objetos aos quais não deveriam ter acesso;

- **API2:2023 Broken Authentication**: vulnerabilidades surgem quando os mecanismos de autenticação de uma API são insuficientes ou mal implementados, tornando-os alvos fáceis para ataques;

- **API3:2023 Broken Object Property Level Authorization**: destaca-se pela exposição inadequada de propriedades de um objeto através de *endpoints* de API, particularmente em APIs REST e GraphQL;

- **API4: 2023 Unrestricted Resource Consumption**: surge quando uma API não impõe limites adequados no consumo de recursos, como largura de banda, CPU, memória e armazenamento;

- **API5:2023 Broken Function Level Authorization**: representa situações em que uma API permite que usuários não autorizados acessem funções ou recursos específicos;

- **API6:2023 Unrestricted Access to Sensitive Business Flows**: relaciona-se à exposição de fluxos de negócios críticos sem restrições de acesso adequadas;

- **API7:2023 Server Side Request Forgery**: evidencia-se quando um *endpoint* da API busca um recurso remoto com base em URLs fornecidas pelo usuário, sem validação adequada;

- **API8:2023 Security Misconfiguration**: manifesta-se quando uma API opera com configurações de segurança inadequadas;

- **API9:2023 Improper Inventory Management**: resulta da falta de controle e documentação sobre as versões e *endpoints* de uma API;

- **API10:2023 Unsafe Consumption of APIs**: surge devido à ausência de controle e documentação adequados sobre as versões e *endpoints* de uma API;

Além da lista específica para segurança de APIs, a OWASP também elaborou o renomado "Top 10" que destaca as principais vulnerabilidades de segurança em aplicações web em geral. Embora não seja focada em APIs, esta lista é relevante para o contexto deste trabalho, pois muitas das vulnerabilidades listadas também podem ser exploradas através de APIs. A lista a seguir apresenta as dez principais vulnerabilidades de segurança em aplicações *web*, conforme definido pela *OWASP* [7] em 2021.

- **A1:2021 Broken Access Control**: mecanismos de autenticação de uma API insuficientes ou mal implementados facilitam ataques;

- **A2:2021 Cryptographic Failures**: uso de algoritmos criptográficos fracos ou implementação incorreta de algoritmos robustos;

- **A3:2021 Injection**: possibilidade de injeção de código malicioso, como SQL, *Cross-Site Scripting (XSS)* e injeção de comandos;

- **A4:2021 Insecure Design**: decisões de design que levam à falta de validação de entrada, ausência de mecanismos de autenticação, autorização e uso de componentes vulneráveis;

- **A5:2021 Security Misconfiguration**: configurações inadequadas, como padrões inseguros, armazenamento em nuvem mal configurado e definições incorretas de segurança do servidor web;

- **A6:2021 Vulnerable and Outdated Components**: uso de componentes com vulnerabilidades reconhecidas;

- **A7:2021 Identification and Authentication Failures**: falhas na identificação e autenticação, como senhas fracas, métodos inseguros e falta de proteção contra-ataques de força bruta;

- **A8:2021 Software and Data Integrity Failures**: falhas na integridade de dados e software, como ausência de validações e proteção contra-ataques de repetição;

- **A9:2021 Security Logging and Monitoring Failures**: ausência de mecanismos adequados de monitoramento e *logging*;

- **A10:2021 Server-Side Request Forgery**: *endpoints* da API que buscam recursos remotos baseados em URLs do usuário sem validação apropriada;

2.3. Fuzzing

Fuzzing, abreviação de *fuzz testing* é uma técnica automatizada de teste de software que emprega uma grande diversidade de entradas, tanto normais como anormais, para a aplicação alvo [10][11]. Introduzido inicialmente por Miller et al. em 1988 [13], o processo envolve não apenas o fornecimento dessas entradas à aplicação, mas também o monitoramento contínuo dos estados de execução do software para identificar comportamentos inesperados, falhas ou travamentos. Esse monitoramento permite que os desenvolvedores e testadores detectem e corrijam vulnerabilidades que só se manifestam quando o software é submetido a condições extremas ou atípicas [11]. Em contraste com outras técnicas, o *fuzzing* destaca-se pela facilidade de implantação, extensibilidade e aplicabilidade, podendo ser executado com ou sem acesso ao código-fonte [10]. Adicionalmente, devido à sua natureza baseada na execução real do *software*, o *fuzzing* apresenta alta precisão [10].

A ferramenta que viabiliza essa técnica é denominada *fuzzer*. Atuando como um gerador de entradas, ele cria os dados de teste e injeta no programa alvo. Existem di-

ferentes abordagens para a geração de testes, como os métodos baseados em mutação e gramática [3] — métodos de mutação modificam entradas de teste já existentes, enquanto técnicas baseadas em gramática geram novas entradas a partir de uma descrição estrutural da entrada. Além disso, os *fuzzers* podem operar em abordagens que vão desde a caixa-preta [16], sem conhecimento do código, até a caixa-branca, com total acesso ao código-fonte [11].

O panorama atual do desenvolvimento de *fuzzing* reflete um crescente interesse na abordagem de segurança de software. Nos últimos anos, a adoção de técnicas de *fuzzing* mostra um aumento constante [11] [10] [5] e tem sido reconhecida por sua eficácia para descobrir tanto falhas de implementação, quanto vulnerabilidades de segurança. Além disso, o *fuzzing* tem se integrado com outras técnicas avançadas, como algoritmo genético, análise de contaminação e execução simbólica [11]. A comunidade acadêmica demonstra um interesse crescente no assunto, como evidenciado pelo aumento nas publicações sobre *fuzzing* [6]. Este foco contínuo sugere que o *fuzzing* continuará a ser uma técnica valiosa no campo da segurança de software nos próximos anos [5].

2.3.1. Fuzzing de APIs

Dada a importância estabelecida das APIs REST como um canal crítico de comunicação entre serviços, a garantia de sua confiabilidade e segurança tornou-se uma preocupação relevante [8] [9]. Nesse cenário, o *fuzzing* de APIs surgiu como uma técnica para identificar erros e vulnerabilidades em serviços baseados em nuvem. Diversos projetos têm se destacado neste domínio. O *REST-ler*, por exemplo, é uma ferramenta, desenvolvida pela Microsoft, que analisa especificações *OpenAPI* (antigo *Swagger*) para gerar testes que interagem com um serviço de nuvem através de sua API *REST* [4]. Tsai et al [16] introduziram um método de *fuzzing* para APIs *REST* que utiliza feedback de Nível de Cobertura de Teste — TCL (do inglês, *Test leve Coverage*) é um conjunto de oito critérios usados para avaliar a cobertura de código e a eficácia dos testes na detecção de falhas. Ele é utilizado como feedback no método de fuzzing proposto por Tsai et al. para APIs REST.

Paralelamente, o DocTer foi concebido para extrair restrições a partir de descrições de API, usadas para gerar entradas válidas e inválidas para testar as funções de aprendizado profundo [18]. Abordagens como o MINER adotam soluções híbridas orientadas por dados para aprimorar o *fuzzing* [12], enquanto o Pythia combina técnicas baseadas em gramática com feedback de cobertura e mutações orientadas por aprendizado [3]. Estas inovações refletem a especialização e evolução contínua na área de *fuzzing* de APIs, sublinhando a necessidade de abordagens de teste robustas no ambiente de desenvolvimento contemporâneo.

3. Objetivos

Esse trabalho tem como principal objetivo desenvolver e avaliar um *fuzzer* especializado para testes de segurança em APIs REST. Para atingir este objetivo, os seguintes objetivos específicos foram estabelecidos.

1. Aprimorar o *fuzzer* para identificar erros e vulnerabilidades nas respostas da API, considerando códigos de retorno e respostas ambíguas.

2. Permitir a configuração precisa de endpoints e seus argumentos, seja em formatos como JSON, parâmetros ou outros.
3. Desenvolver uma funcionalidade que forneça um relatório detalhado das entradas fornecidas em comparação com as respostas esperadas da API.
4. Implementar funcionalidades que gerenciem desafios associados à autenticação e autorização em APIs REST, bem como considerar validações específicas da API durante o processo de *fuzzing*.
5. Explorar a geração de entradas baseadas em gramáticas pré-configuradas e geração aleatória, oferecendo ambas as opções ao usuário e propondo uma sintaxe clara para um arquivo de configuração do *fuzzer*.

4. Procedimentos metodológicos/Métodos e técnicas

Inicialmente, será realizada uma revisão bibliográfica para compreender o escopo e os desafios associados à segurança dessas APIs. Este estudo envolverá a investigação das principais vulnerabilidades e desafios relacionados à segurança das APIs REST, bem como a avaliação do impacto dessas vulnerabilidades nos sistemas e a consequente importância de implementar medidas de mitigação. Além disso, o estudo investigará técnicas e ferramentas de *fuzzing*.

Será feita uma análise crítica das ferramentas de *fuzzing* disponíveis no mercado, considerando suas capacidades e limitações. Além disso, as principais técnicas de *fuzzing* serão estudadas, com ênfase especial em suas aplicações para APIs REST. Esta etapa também envolverá a identificação das características essenciais que um *fuzzer* deve possuir para ser considerado especializado em APIs REST.

Com base no conhecimento adquirido, a próxima fase do projeto será dedicada ao desenvolvimento de um *fuzzer* especializado para APIs REST. Este *fuzzer* será projetado para realizar testes automatizados em APIs, com funcionalidades que permitam identificar vulnerabilidades comuns, como injeção de SQL, *Cross-Site Scripting (XSS)* e falhas de autenticação.

Durante o desenvolvimento, o *fuzzer* será configurado para abordar uma variedade de endpoints, permitindo a configuração precisa de seus respectivos argumentos, sejam eles por parâmetros ou JSON. A ferramenta será projetada para explorar entradas com base em uma gramática pré-configurada — sendo possível especificar tipos de dados, por exemplo, arquivos, inteiros ou *strings* — bem como a geração aleatória de entradas (tanto válidas, como inválidas). Além disso, o *fuzzer* contará com mecanismos para analisar e interpretar respostas ambíguas, focando em mensagens de retorno e códigos de status HTTP, de forma que a resposta da API indique se o comportamento foi como esperado ou não. Adicionalmente, a autenticação e autorização também serão foco dessa ferramenta, com o *fuzzer* conseguindo gerenciar desafios relacionados a API keys, tokens JWT entre outros mecanismos de acesso.

Após o desenvolvimento, o próximo passo será avaliar o desempenho e a eficácia do *fuzzer* especializado. Esta avaliação será conduzida através de experimentos práticos, onde o *fuzzer* será aplicado em diferentes cenários e APIs. Em particular, os testes iniciais serão realizados na API do ConsistencIA, uma plataforma desenvolvida pelo CIAAgro (Centro de Inteligência Artificial no Agro da UEL). Esta escolha não só proporciona um

ambiente controlado, mas também representa um cenário real e complexo, ideal para avaliar a eficácia do *fuzzer*. Por fim, os resultados obtidos serão analisados detalhadamente, permitindo uma discussão sobre as vantagens, desvantagens e boas práticas associadas ao uso do *fuzzer* especializado desenvolvido. Esta análise final proporcionará percepções valiosas e recomendações para futuras pesquisas e aplicações na área.

5. Cronograma de Execução

1. Levantamento bibliográfico acerca de segurança de APIs e *fuzzing*.
2. Análise de segurança em APIs REST e estudo de vulnerabilidades.
3. Investigação de técnicas e ferramentas de *fuzzing*.
4. Desenvolvimento de um *fuzzer* especializado para APIs REST.
5. Avaliação prática do *fuzzer* desenvolvido.
6. Análise e discussão dos resultados obtidos.
7. Escrita do TCC.

Tabela 1. Cronograma de Execução

	set	out	nov	dez	jan	fev	mar	abr
Atividade 1	X	X						
Atividade 2		X	X					
Atividade 3		X	X					
Atividade 4			X	X	X	X		
Atividade 5					X	X		
Atividade 6						X	X	
Atividade 7					X	X	X	X

6. Contribuições e/ou Resultados esperados

Com a crescente adoção de arquiteturas baseadas em APIs, garantir a segurança dessas interfaces tornou-se essencial [8]. Ao desenvolver um *fuzzer* focado em APIs REST, este projeto poderá oferecer uma ferramenta mais eficaz e direcionada para profissionais de segurança e desenvolvedores.

Além disso, os resultados obtidos a partir da avaliação *fuzzer* desenvolvido poderão fornecer informações valiosas sobre a eficácia dos diferentes métodos de *fuzzing* no contexto específico de APIs REST. Isso pode levar ao aprimoramento das técnicas de *fuzzing* e, eventualmente, à criação de novas abordagens para a segurança de APIs.

Também é importante destacar que este projeto pode contribuir para a conscientização sobre a importância da segurança de APIs. O desenvolvimento e divulgação de uma ferramenta especializada pode encorajar a adoção de práticas de segurança mais robustas e mostrar possíveis vulnerabilidades, levando a um ecossistema de aplicações mais seguro.

Por fim, o projeto pode servir como uma base para pesquisas futuras. Outros pesquisadores podem se basear nos resultados e no *fuzzer* desenvolvido para explorar aspectos adicionais da segurança de APIs ou para adaptar e melhorar a ferramenta.

Em suma, espera-se que este projeto não apenas forneça uma ferramenta valiosa para testar APIs, mas também fomente o desenvolvimento de práticas de segurança

melhores, conscientize sobre a importância da analisar vulnerabilidades de APIs e sirva como um ponto de partida para futuras investigações na área.

7. Espaço para assinaturas

Londrina, *06 de julho de 2023*.

_____ _____

Aluno Orientador

Referências

[1] Eyhab Al-Masri. Enhancing the Microservices Architecture for the Internet of Things. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 5119–5125, December 2018.

[2] Inc. Amazon Web Services. What is an API? - Application Programming Interfaces Explained, 2023. Acessado em: 2023-08-30.

[3] Vaggelis Atlidakis, Roxana Geambasu, Patrice Godefroid, Marina Polishchuk, and Baishakhi Ray. Pythia: Grammar-Based Fuzzing of REST APIs with Coverage-guided Feedback and Learning-based Mutations, May 2020. arXiv:2005.11498 [cs].

[4] Vaggelis Atlidakis, Patrice Godefroid, and Marina Polishchuk. REST-ler: Automatic Intelligent REST API Fuzzing, June 2018. arXiv:1806.09739 [cs].

[5] Marcel Boehme, Cristian Cadar, and Abhik ROYCHOUDHURY. Fuzzing: Challenges and Reflections. *IEEE Software*, 38(3):79–86, May 2021. Conference Name: IEEE Software.

[6] Josué Alejandro Díaz-Rojas, Jorge Octavio Ocharán-Hernández, Juan Carlos Pérez-Arriaga, and Xavier Limón. Web API Security Vulnerabilities and Mitigation Mechanisms: A Systematic Mapping Study. In *2021 9th International Conference in Software Engineering Research and Innovation (CONISOFT)*, pages 207–218, October 2021.

[7] OWASP Foundation. Owasp top ten. `https://owasp.org/Top10/`, 2023. Acessado em: 2023-09-02.

[8] Muhammad Idris, Iwan Syarif, and Idris Winarno. Development of Vulnerable Web Application Based on OWASP API Security Risks. In *2021 International Electronics Symposium (IES)*, pages 190–194, September 2021.

[9] Muhammad Idris, Iwan Syarif, and Idris Winarno. Web Application Security Education Platform Based on OWASP API Security Project. *EMITTER International Journal of Engineering Technology*, pages 246–261, December 2022.

[10] Jun Li, Bodong Zhao, and Chao Zhang. Fuzzing: a survey. *Cybersecurity*, 1(1):6, June 2018.

[11] Hongliang Liang, Xiaoxiao Pei, Xiaodong Jia, Wuwei Shen, and Jian Zhang. Fuzzing: State of the Art. *IEEE Transactions on Reliability*, 67(3):1199–1218, September 2018. Conference Name: IEEE Transactions on Reliability.

[12] Chenyang Lyu, Jiacheng Xu, Shouling Ji, Xuhong Zhang, Qinying Wang, Binbin Zhao, Gaoning Pan, Wei Cao, and Raheem Beyah. MINER: A Hybrid Data-Driven Approach for REST API Fuzzing, March 2023. arXiv:2303.02545 [cs].

[13] Barton P. Miller, Lars Fredriksen, and Bryan So. An empirical study of the reliability of UNIX utilities. *Communications of the ACM*, 33(12):32–44, December 1990.

[14] Inc. Red Hat. What is an API? `https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces`, 2023. Acessado em: 2023-08-30.

[15] Sergio Segura, José A. Parejo, Javier Troya, and Antonio Ruiz-Cortés. Metamorphic Testing of RESTful Web APIs. *IEEE Transactions on Software Engineering*, 44(11):1083–1099, November 2018. Conference Name: IEEE Transactions on Software Engineering.

[16] Chung-Hsuan Tsai, Shi-Chun Tsai, and Shih-Kun Huang. REST API Fuzzing by Coverage Level Guided Blackbox Testing, December 2021. arXiv:2112.15485 [cs].

[17] Xinkai Wang, Chao Li, Lu Zhang, Xiaofeng Hou, Quan Chen, and Minyi Guo. Exploring Efficient Microservice Level Parallelism. In *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 223–233, May 2022. ISSN: 1530-2075.

[18] Danning Xie, Yitong Li, Mijung Kim, Hung Viet Pham, Lin Tan, Xiangyu Zhang, and Michael W. Godfrey. DocTer: documentation-guided fuzzing for testing deep learning API functions. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2022, pages 176–188, New York, NY, USA, July 2022. Association for Computing Machinery.