AUTOMATIZAÇÃO DE TESTES DE SEGURANÇA A SOFTWARES WEB

Bruno Sulkovski Moretto[1]

RESUMO

Hoje, a sociedade faz o uso massivo de softwares, trafegando milhares de dados em segundos, usando-os para diferentes fins: fluxos de trabalho, redes sociais, *streamings* de vídeo, entre inúmeros outros. Os softwares precisam se garantir seguros, protegendo a si mesmo e aos seus usuários, não expondo vulnerabilidades para pessoas mal intencionadas. Esta deve ser uma das preocupações mais importantes ao desenvolver um novo software. Com o intuito de ajudar os desenvolvedores a testarem e protegerem suas aplicações, será desenvolvido um script para automatizar testes de segurança a softwares web. O script fará uma varredura da API (*application programming interface*), através da disponibilização da documentação gerada pela biblioteca Swagger, testando alguns dos problemas mais comuns de se encontrar nestas aplicações, gerando uma listagem das brechas encontradas para o programador. A implementação do *script* se dará através da linguagem de programação Javascript e do ambiente de execução NodeJS. No final, o objetivo é entregar maior proteção aos softwares web, evitando possíveis brechas para ataques, garantindo que os seus usuários e seus dados estejam protegidos contra pessoas mal intencionadas.

Palavras-chave: Segurança de Software; Software Web; Automatização de testes.

ABSTRACT

Today, society extensively relies on software, processing thousands of data in seconds and utilizing them for various purposes such as workflows, social networks, video streaming, among countless others. Software must ensure its own security, protecting both itself and its users, and avoiding exposure of vulnerabilities to malicious individuals. This should be one of the most important concerns when developing new software. In order to assist developers in testing and securing their applications, a script will be developed to automate security testing for web software. The script will scan the Application Programming Interface (API) by utilizing documentation generated by the Swagger library, testing some of the most common issues found in these applications, and generating a list of vulnerabilities for the programmer. The script will be implemented using the JavaScript programming language and the NodeJS execution environment. Ultimately, the goal is to provide greater protection to web software, preventing potential vulnerabilities for attacks and ensuring that users and their data are safeguarded against malicious intent.

Palavras-chave: Software Security; Web Software; Test Automation.

---

[1] Discente do Curso Ciência da Computação da Universidade La Salle - Unilasalle, matriculado na disciplina de Trabalho de Conclusão de Curso II. E-mail: bruno.201920129@unilasalle.edu.br, sob a orientação Prof. Aline Duarte Riva. E-mail: aline.riva@unilasalle.edu.br. Data de entrega: 01 dez. 2023.

1 INTRODUÇÃO

A humanidade evoluiu muito através das revoluções que a história conta. Inicialmente, se tem nos anos 1760 a primeira revolução industrial, considerada uma revolução mecânica: começa-se a utilizar água e vapor para realizar o que até antes era manufaturado. Posteriormente, se tem a segunda revolução, a revolução elétrica, onde começa-se a utilizar eletricidade nas máquinas. Por último, mas não menos importante, se tem a terceira revolução, uma revolução de automatização, onde se cria equipamentos eletrônicos para automatizar o processo humano. E ainda, mais tardar, substituir válvulas pelos, utilizados até hoje, transistores, entregando um poder de processamento que a humanidade jamais viu.

A cada revolução que a sociedade passou, preocupações com segurança precisaram ser tomadas. Em contramão da produtividade, apareciam vários perigos, a necessidade de tomar cuidado com vapores ferventes na primeira das revoluções, ou os possíveis efeitos letais da eletricidade na segunda. Hoje em dia, com todas as seguranças, pode ser dizer, físicas mapeadas, se começa a ter necessidade de assegurar em outro mundo, o mundo digital. A humanidade não vive mais sem tecnologia, sem internet, sem softwares.

Com o avanço da tecnologia e das aplicações web, a maioria dos processos e fluxos que eram feitos fisicamente, acabaram por ser digitalizados, aumentando, dessa forma, o fluxo de dados e usuários que estão na internet. Isso aconteceu devido a enorme facilidade que se tem para trabalhar num ambiente virtual, através da comodidade e da agilidade que os softwares fornecem.

Nessa temática é preciso se certificar de que o que trafega na internet está seguro, contudo, no cenário atual, isso está longe de acontecer. Um estudo realizado pelo Identity Theft Resource Center (ITRC), mostra que em 2022 mais de 422 milhões de pessoas tiveram dados comprometidos, tendo um aumento de 42% ao ano anterior, mostrando que não se está progredindo muito no quesito da segurança computacional.

Apesar de todos os alertas, muitas empresas ainda deixam de se comprometer com a segurança dos seus usuários, preferindo a entrega de novas funcionalidades do sistema a recursos de segurança ou correção de brechas para ataques. Há vários exemplos disso, como o da empresa de gestão de crédito Equifax, que devido ao software desatualizado há cerca de 2 meses, teve os dados de 143 milhões de americanos vazados.

Em um cenário onde falhas de segurança se encontram em grande parte dos softwares web, tem-se a problemática, como desenvolver novas técnicas de segurança de software para contribuir com o desenvolvedor na proteção de ameaças de seus softwares, tornando-os mais confiáveis, íntegros e seguros para evitar que transtornos maiores possam acontecer?

A tecnologia já está difundida na sociedade, é muito difícil a realização de qualquer tipo de tarefa sem ela. Desde processamento de dados complexos, até uma simples troca de e-mails, se tem ela como meio de fazê-lo. Em 2021, o número de domicílios com acesso à internet no Brasil chegou a 90,0%, sendo cerca de 65,6 milhões de domicílios conectados, segundo dados da Pesquisa Nacional por Amostra de Domicílio realizada pelo IBGE, em parceria com o Ministério das Comunicações (MCom).

Com isso, a tendência é o uso da tecnologia aumentar cada vez mais. Novos softwares serão lançados e absorvidos pelas pessoas, estes podendo ser, assim como já são, de qualquer área imaginável: entretenimento, governamental, saúde,

bancária, investimentos, entre outras infinitas possibilidades. Desse modo, cada indivíduo terá mais facilidade no seu dia-a-dia, gerando um aumento na qualidade de vida.

Porém, com o uso massivo da tecnologia, a vida das pessoas acaba sendo "digitalizada" e informações importantes são mantidas na rede de computadores. Dessa maneira, vários alertas relacionados à segurança são ligados. Apesar de novas leis, como a LGPD (Lei Geral de Proteção de Dados) tentarem fazer as empresas terem mais cuidado com os dados de seus clientes, muitos desses cuidados vêm sendo deixados de lado, o que coloca em xeque a segurança de importantes informações de seus usuários, abrindo possibilidades para pessoas mal intencionadas agirem. Se tem vários exemplos negativos em relação ao uso maléfico da tecnologia: redes clandestinas, onde há diversos crimes ocorrendo, como sites de pedofilia, venda de armas ilegal, documentos sigilosos vazados, entre outros vários crimes. Uso ilegal de imagem e divulgação de materiais sem permissão do detendor dos direitos, o que chamam de pirataria. A ética e a moral não se fazem muito presentes no mundo web como deveriam.

Logo, para justificativa do presente artigo, é importante manter todos os softwares seguros, fechados a todo tipo de invasão conhecida, preservando a integridade dos dados de seus usuários, bem como possíveis problemas na operação da empresa por trás do software.

Qualquer tipo de vazamento de invasão a um sistema pode ocasionar terríveis consequências na vida das pessoas. Novamente, utilizando um exemplo ligado à área bancária: se uma pessoa tem os dados de login do aplicativo de sua conta do banco vazados, este poderá perder todo o dinheiro ali guardado, bem como a contração de inúmeras dívidas devido a empréstimos feitos, por exemplo. Isso já é o suficiente para acabar com uma vida. Dados de uma pesquisa da empresa de segurança Tenable, mostram que mais de 40,4 bilhões vazamentos de dados ocorreram no mundo em 2021, sendo 815 milhões apenas no Brasil, país que ocupa a 6° posição no ranking mundial de vazamento de dados. As notícias destacam cada vez mais informações roubadas, como por exemplo, o mega vazamento de 223 milhões de brasileiros que foi revelado em janeiro de 2021.

Portanto, a segurança de software é extremamente importante e relevante para a nossa sociedade e cada vez mais estudos devem ser desenvolvidos para se alcançar o mais alto nível de segurança possível, apoiando as empresas, tanto do setor privado, quanto do setor público a melhorarem suas barreiras contra pessoas mal intencionadas.

Então, o presente estudo tem como objetivo entender possíveis vulnerabilidades que se encontram na internet, visando construir um programa para automatizar testes de segurança sob softwares web, focando em suas API’s, reportando as falhas encontradas, listando as rotas que se encontram possíveis brechas.

2 REFERENCIAL TEÓRICO

Há diversas terminologias usadas no mundo da tecnologia da informação que podem ser desconhecidas por pessoas que não estão muito familiarizadas com o meio. Para o entendimento do artigo é necessário ter conhecimento sobre algumas definições.

Softwares são os programas de computadores. Possuem procedimentos, regras, documentações e dados. Plenamente usados nos dias atuais em qualquer

ramo, possuem alto processamento e conseguem realizar atividades complexas de forma fácil, gerando uma alta produtividade e mais assertividade aos seus usuários.

O presente trabalho irá focar em softwares voltados para web, ou seja, aqueles feitos para serem executados no ambiente de internet nos navegadores dos usuários.

Vulnerabilidade de softwares são defeitos do software, podendo ser explorado, comprometendo o software em relação a sua segurança (NOVASKI, 2018).

Invasão a software é a exploração de vulnerabilidades por um invasor para comprometer um software, acessando de forma não autorizada o software, podendo gerar prejuízos em relação aos seus dados com vazamento e perda de informações, interrupção de serviços, entre outros.

Testes de segurança são as atividades de testes para garantir a segurança da aplicação, verificando diversos cenários para assegurar que a aplicação está livre de vulnerabilidades e brechas para um possível ataque.

Dados pessoais são informações relacionadas à pessoa natural identificada ou identificável (Art. 5.º, I, LGPD), tais como CPF, RG, local de nascimento, etc.

Dados sensíveis são dados pessoais sobre origem racial ou étnica, convicção religiosa, opinião política, filiação a sindicato ou a organização de caráter religioso, filosófico ou político, dado referente à saúde ou à vida sexual, dado genético ou biométrico, quando vinculado a uma pessoa natural" (Art. 5.º, II, da LGPD).

Automação é a ação de deixar algo de forma automática, sem necessidade de intervenção humana direta.

O Protocolo HTTP, Protocolo de transferência de Hipertexto (*Hypertext Transfer Protocol*), conforme (BEOCK, CONSONE, RODRIGUES & PETRICIA) é um protocolo por camadas das aplicações web, tendo duas partes principais, a aplicação cliente e a aplicação servidor que conversam entre si por mensagens HTTP.

Cookies são pedaços de dados guardados pelos navegadores utilizados pelas aplicações para persistências referentes a sessões HTTP.

Os tipos MIME são utilizados para identificação dos tipos de documentos que são trocados entre aplicações, por exemplo, se o tipo for 'text/html' a aplicação sabe que a resposta da requisição deve ser interpretada como uma página HTML.

As aplicações standalone são aquelas que possuem tudo que é necessário para suas execuções, não dependendo de nenhum tipo de auxílio externo.

Um *endpoint* ou rota da aplicação é um endereço mapeado pela aplicação servidor para que esta possa responder ao cliente com o bloco de código referente a rota, como buscar dados de um usuário.

Uma API, acrônimo de *application programming interface*, é uma interface que disponibiliza diversas operações para aqueles que a chamam, retornando e persistindo dados.

Frontend é a aplicação que interage com o usuário final, como um site.

Backend é a aplicação que possui as regras de negócio e processa os dados.

JSON, sigla para *JavaScript Object Notation,* é um tipo de formato de texto muito utilizado para passagem de dados pelas API's. Consiste em chaves e valores, utilizando aspas duplas para engloba-los.

XML, sigla para *Extensible Markup Language,* é um tipo de linguagem de marcação, plenamente utilizada para formatar dados, deixando facilmente legível e manipulável para humanos como máquinas.

3 DESENVOLVIMENTO

Para o desenvolvimento do script foram usadas as seguintes tecnologias:

- Javascript: linguagem de programação escolhida, já que esta trabalha muito bem com aplicações web;
- NodeJS: ambiente de execução Javascript, possibilitando a execução de programas nesta linguagem como aplicações standalone;
- Biblioteca Axios: biblioteca para execução de requisições HTTP para NodeJS;
- Biblioteca Chalk: biblioteca para estilização de saídas no console;
- Biblioteca Config: biblioteca para configuração de parâmetros da aplicação.

O teste se dará nas API's dos softwares web, utilizando de suas documentações para enumerar todas as rotas destes, além de informações importantes para os testes. Com isso, é obrigatório, nesta primeira versão do script, que a aplicação alvo do teste implemente a documentação da biblioteca Swagger. Esta é uma biblioteca que visa simplificar a documentação da API, gerando-a a partir do código fonte da aplicação. Além disso, ela segue um padrão único, facilitando a varredura das rotas e a obtenção de informações da aplicação para realizar os testes. É amplamente utilizada no mercado, abrangendo múltiplas linguagens de programação, sendo por esse motivo a escolhida para ser pioneira para o desenvolvimento dos testes de segurança.

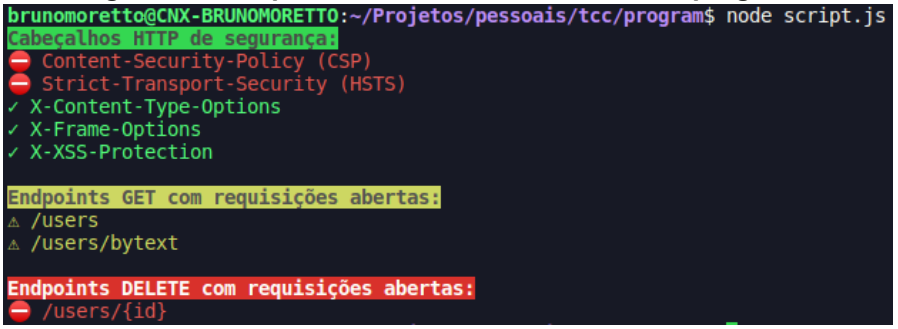
Para executar o script, é necessário ter o ambiente de desenvolvimento NodeJS com a versão 18 instalada na máquina. Para realizar a configuração dos parâmetros do teste, é preciso modificar o arquivo que fica dentro da pasta config, default.json, este é um arquivo no formato JSON que contém os parâmetros que a aplicação usará para realizar os testes. São três parâmetros:

- app_uri: parâmetro referente a URI da aplicação a ser testada, exemplo: http://localhost:8080/api;
- docs_path: parâmetro opcional referente ao caminho para acessar a documentação da API. Este possui um valor padrão de '/v3/api-docs', que é o caminho que a biblioteca Swagger utiliza;
- api_token: parâmetro opcional referente ao token de autorização para realizar requisições. Deve ser informado quando o teste deve envolver questões de autenticação e autorização.

Com a configuração realizada, basta abrir um console e utilizar o node para rodar o arquivo do script: node script.js

Após a execução do script, o resultado do teste será escrito no próprio console utilizado para iniciar o teste, como mostra a Figura 1 abaixo:

Figura 1: Exemplo de resultado de execução do programa



Fonte: (O autor, 2023).

3.1 Cabeçalhos HTTP

Os cabeçalhos, também conhecidos como *headers*, das requisições HTTP são uma ótima ferramenta para proteger aplicações de vulnerabilidades. No protocolo HTTP, ao fazer uma requisição, recebe-se uma resposta, onde os cabeçalhos desta podem incluir diversas orientações para auxiliar na prevenção de brechas para usuários mal intencionados. São diversos cabeçalhos disponíveis para ajudar na proteção das aplicações, que serão listados abaixo.

O cabeçalho X-Frame-Options é responsável por dizer se o browser deve permitir o uso de frames, como as tags: <frame>, <iframe>, <embed> e <object>. Com o uso destas tags é possível um atacante injetar uma nova camada num site para forçar um *clickjacking*, onde o usuário não sabe que está clicando numa área alterada por um atacante, podendo realizar ações indesejadas.

O cabeçalho Content Security Policy, também visto com a sigla CSP, é usado para assegurar aplicações de ataques como Cross-Site Scripting, conhecido como XSS. Nesse *header* indica-se de quais origens são permitidas carregar certos tipos de conteúdos. Então, por exemplo, a proteção para XSS ocorre quando bloquea-se a injeção de scripts por qualquer fonte externa, já que esse ataque ocorre quando um atacante faz a injeção de códigos maliciosos nas aplicações, podendo roubar informações, autenticações em cookies, redirecionar o usuário para outras páginas, entres outras possibilidades.

O cabeçalho X-XSS-Protection também protege a aplicação de ataques XSS, forçando a página parar o carregamento quando é detectado algum tipo de ataque do gênero. Porém, esta função só ocorre para os *browsers* Internet Explorer, Chrome, e Safari.

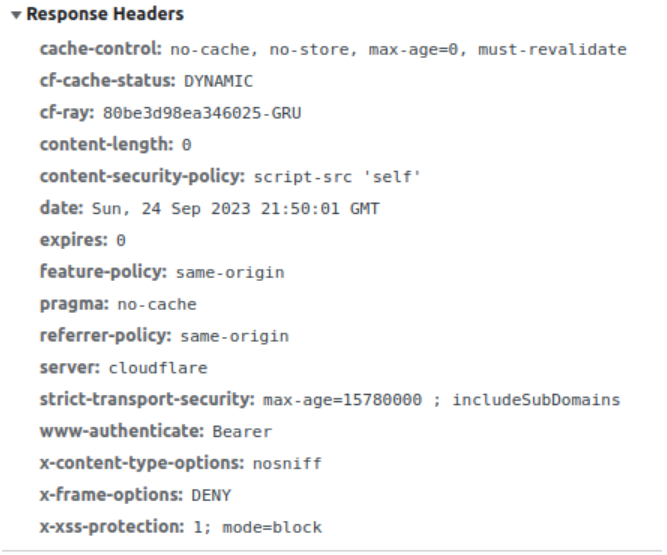
O cabeçalho X-Content-Type-Options indica para os navegadores que os tipos MIME que retornam do servidor devem estar definidos pelo recebedor da resposta, não podem ser adivinhados. Com isso, impossibilita-se um atacante de transformar respostas com tipo MIME não executável em um executável.

O cabeçalho Strict-Transport-Security, também conhecido como HSTS, diz para os navegadores que a aplicação deve ser carregada com o protocolo HTTPS, onde qualquer tentativa de uso HTTP deve ser redirecionada como HTTPS. A segurança do protocolo HTTPS é superior ao HTTP, já que este não possui criptografia, transmitindo os dados de forma vulnerável, enquanto no HTTPS, utiliza-se um certificado SSL/TLS para a criptografia dos dados transmitidos.

O cabeçalho Set-Cookie é usado para a troca de *cookies* entre servidor e *browser*, onde o *browser* usará futuramente o *cookie* para conversar com o servidor. O mais importante deste cabeçalho são seus atributos que dizem respeito se o *cookie* será encriptado, se estes poderão ser acessados, ou, quem poderá enviar os *cookies*.

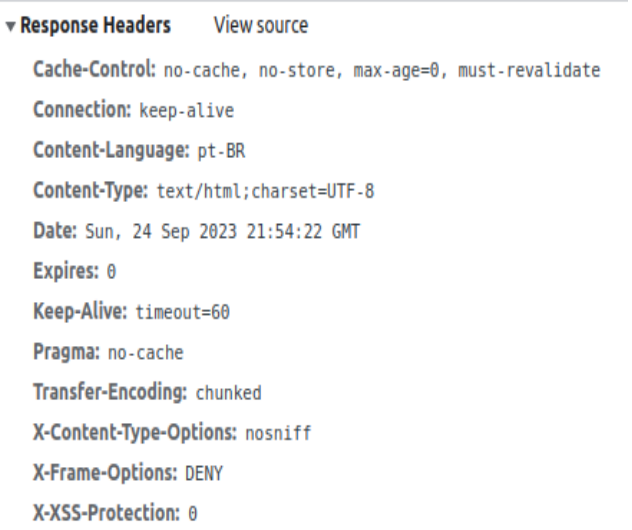
É possível facilmente visualizar se um site, por exemplo, implementa os *headers* acima. Basta abrir o console de desenvolvedor do navegador e observar as respostas das requisições na aba Rede. Nas Figuras 2 e 3 pode-se ver os cabeçalhos de resposta de duas aplicações web: a Figura 2 mostra o não uso dos cabeçalhos, enquanto a Figura 3 mostra o uso dos cabeçalhos:

Figura 2: Resposta HTTP sem cabeçalhos de segurança



Fonte: (O autor, 2023).

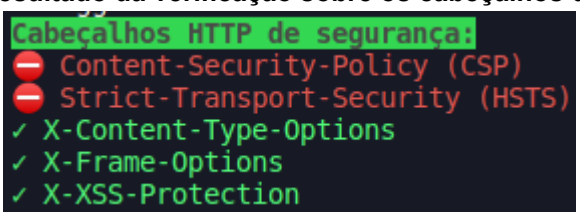
Figura 3: Resposta HTTP com cabeçalhos de segurança



Fonte: (O autor, 2023).

Portanto, para a plena proteção de uma aplicação web, é necessário que todos estes cabeçalhos de resposta sejam levados em consideração e implementados. Com isso, o script faz a listagem destes principais cabeçalhos de segurança, mostrando quais estão sendo ou não utilizados, como mostra a Figura 4.

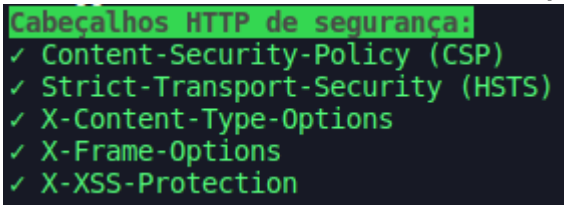
Figura 4: Resultado da verificação sobre os cabeçalhos de resposta



Fonte: (O autor, 2023).

É possível ver que a aplicação testada na Figura 4 está utilizando parcialmente os cabeçalhos de segurança. Agora na Figura 5, visualiza-se uma aplicação cobrindo totalmente os cabeçalhos de resposta HTTP.

Figura 5: Resultado da verificação sobre os cabeçalhos de resposta 100% cobertos



Fonte: (O autor, 2023).

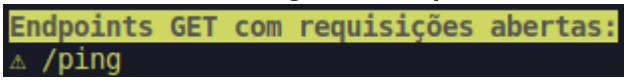
Sendo assim, a partir do relatório gerado pelo script, o desenvolvedor tem capacidade de atentar a este detalhe e proteger, de forma simples, sua aplicação.

3.2 Endpoints abertos

Uma das maiores vulnerabilidades possíveis para uma aplicação é a não utilização de mecanismos de autenticação e autorização. Através da falta disto, o atacante pode realizar tentativas de invasão ao sistemas sem nenhuma barreira, já que a aplicação está totalmente aberta. Uma aplicação que utiliza autenticação e autorização irá restringir o acesso a ela, podendo a partir da autenticação indicar que apenas usuários com credenciais e identificados possam realizar requisições, e, a partir da autorização, se certificar que os usuários tenham acesso a apenas aos privilégios que os dizem respeito.;

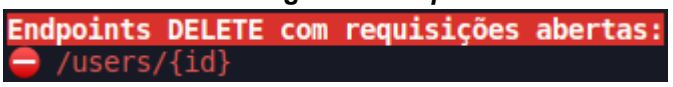
É difícil, nos dias atuais, achar aplicações que estejam totalmente desprotegidas, porém, não é raro ver algumas rotas da API se perderem da configuração de segurança, o que é muito perigoso, pois basta uma simples brecha para que um atacante consiga realizar muitos danos. Sendo assim, o script realiza uma varredura em todas as rotas GET e DELETE, julgadas sendo as mais prejudiciais, para verificar quais podem ser chamadas sem qualquer tipo de autenticação ou autorização. O relatório é gerado a nível de aviso com as requisições do tipo GET, já que pode haver algumas exceções onde a rota deve realmente ficar aberta, como uma rota para verificar se a aplicação está saudável e estável, porém as requisições do tipo DELETE listadas como problema, já que se trata de exclusão de recursos. Conforme as Figuras 6 e 7, é possível visualizar como o relatório é gerado pelo script.

Figura 6: resultado da listagem de *endpoints* GET abertos



Fonte: (O autor, 2023).

Figura 7: resultado da listagem de *endpoints* DELETE abertos



Fonte: (O autor, 2023).

A partir da listagem dos *endpoints* o desenvolvedor pode verificar se cada um destes deveriam, de fato, estar abertos ou se é algum problema com a segurança.

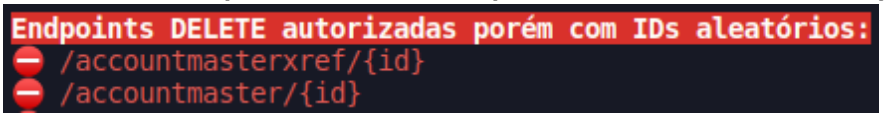
3.3 Visualização de dados de outros donos de recurso

Um dos padrões mais utilizados no desenvolvimento de software é o padrão cliente servidor, esse padrão consiste em uma aplicação cliente, geralmente sendo uma aplicação frontend, enquanto a aplicação servidor é a aplicação que realiza a lógica de negócio, fornecendo e persistindo dados para a aplicação cliente. Numa arquitetura cliente servidor segura, para o cliente poder realizar requisições ao servidor, precisará enviar junto a requisição um token de acesso reconhecido pela aplicação servidor, quase em 100% dos casos é enviado um cabeçalho HTTP da requisição chamado de *Authorization* com o valor do token.

O cenário descrito acima garante que a aplicação servidor receba apenas as requisições autorizadas de acontecer, porém, em muitas API's uma grande falha de segurança é deixada de lado: a obtenção de dados de outras pessoas que não pertencem ao token de acesso enviado, exemplificando: um usuário realizou login no software e obteve seu token de acesso, porém, com este token de acesso ele conseguiu chamar a API e obter dados de um usuário desconhecido. Com isso, se um usuário malicioso obter de alguma forma um token de acesso, este poderia roubar muitas informações de qualquer usuário possível. Portanto, é muito importante garantir que cada usuário dono de recursos possa visualizar aquilo que lhe pertence, bloqueando quaisquer requisições por dados de outros usuários.

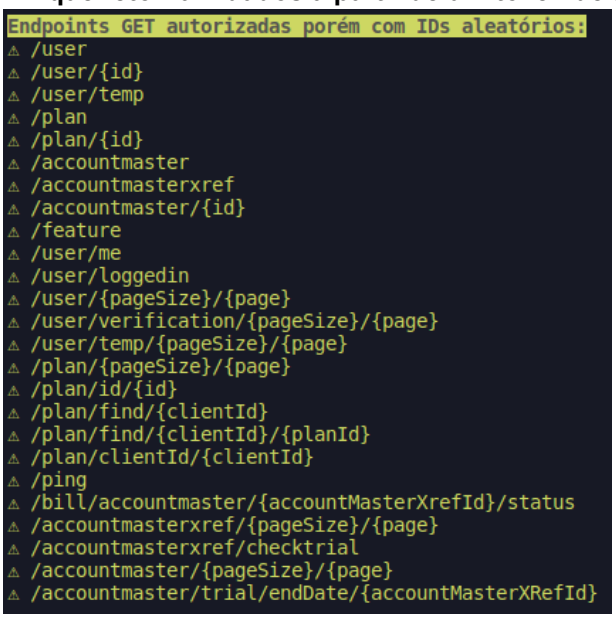
Para este tipo de vazamento de dados o script realiza uma varredura parecida com a listagem de *endpoints* abertos anterior, porém utilizando a forma de autorização disponibilizada pelo desenvolvedor na execução do script, enviando-o juntamente com a requisição e verificando se algum identificador aleatório retorna dados. O resultado é mostrado nas Figuras 8 e 9:

Figura 8: Rotas DELETE que excluem dados a partir de um token de acesso qualquer



Fonte: (O autor, 2023).

Figura 9: rotas GET que retornam dados a partir de um token de acesso qualquer



Fonte: (O autor, 2023).

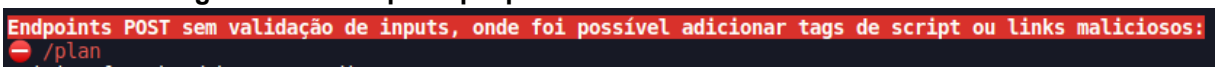
A partir do relatório, os desenvolvedores podem adicionar filtros aos seus códigos validando se os dados que retornam realmente pertencem ao usuário que os requereu.

3.4 Validação de entradas

A função de uma API se baseia em receber dados e processá-los. Esses dados podem ser recebidos de algumas formas, mas as formas mais utilizadas são através do corpo da requisição com o formato de arquivo JSON ou XML. O padrão comumente utilizado pelas aplicações é utilizar rotas do tipo HTTP *post* para receber os dados, sendo na maioria das vezes a aplicação cliente *frontend* mandando para o servidor os dados que o usuário informou. Apesar de ser um fluxo natural das API's é preciso que estas estejam protegidas contra possíveis entradas maliciosas enviadas por atacantes. Através dos dados de entrada é possível que uma pessoa mal intencionada envie códigos de injeção, ou até mesmo, arquivos executáveis que, caso executados, podem gerar perdas inimagináveis. Por exemplo, num cenário onde um atacante enviou para o servidor num campo de escrita, dados que eram na verdade um script que redireciona o usuário para uma página maliciosa, ao salvar este dado e renderizando-o na tela, o sistema se auto sabota criando um ataque de *cross-site scripting*, conhecido como XSS.

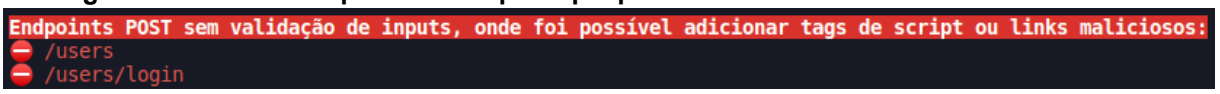
Com isso, é necessário que uma API não aceite todo e qualquer dado enviado por um cliente, realizando uma validação da entrada informada para evitar esses tipos de *inputs* maliciosos. Para ajudar o desenvolvedor neste sentido, desenvolveu-se uma funcionalidade no programa de teste que varre os *endpoints* do tipo HTTP *post* procurando aqueles que aceitam requisições com dados parecidos com scripts ou links maliciosos, por exemplo, injeção do código <script>console.log('Você foi hackeado!')</script> num campo *string*. Então o resultado da varredura irá mostrar as rotas que estão suscetíveis a esse tipo de ataque, como mostra a Figura 10 e a Figura 11 a seguir:

Figura 10: Rotas post que permitem dados maliciosos como entrada



Fonte: (O autor, 2023).

Figura 11: Outro exemplo de rotas post que permitem dados maliciosos como entrada



Fonte: (O autor, 2023).

6 CONSIDERAÇÕES FINAIS

Ao longo deste artigo, desenvolveu-se um script para realizar a automatização de testes de segurança a aplicações web, dando ênfase às API's que as servem. Focou-se em vulnerabilidades e problemas mais comuns de se encontrar nos softwares de hoje. Com o desenvolvimento do programa, implementou-se aquilo proposto, gerando valor a segurança de software e apresentando resultados importantes para avançar nesta área.

A segurança de software sempre será tema de preocupação em todos os meios da tecnologia, já que cada vez mais a sociedade mostra que utilizará softwares e fornecerá mais e mais dos seus dados na internet. Por isso, é

importante o surgimento de novas ferramentas que possam auxiliar o desenvolvedor a proteger suas aplicações, não dando brechas para pessoas mal intencionadas, garantindo proteção aos seus softwares e aos dados de seus usuários.

O programa desenvolvido garantirá que seja possível a equipe de desenvolvimento estar ciente de algumas das vulnerabilidades que atacantes utilizam para burlar a segurança de aplicações, podendo assim amarrar as pontas soltas dos seus softwares, buscando deixá-los o mais protegidos possíveis. Para isso, ao final do desenvolvimento do script, as seguintes verificações foram feitas:

- Validação dos cabeçalhos HTTP;
- Verificação de rotas abertas sem implementações de autenticação ou autorização para serem chamadas;
- Verificação de permissões para visualização de recursos;
- Validação de entradas maliciosas.

Através do relatório gerado pelo programa, já é possível adicionar um bom nível de segurança à aplicação. Portanto, o intuito do desenvolvimento da ferramenta foi cumprido, agregando segurança aos softwares web.

Como próximos passos têm-se a criação de novas verificações de vulnerabilidades, assim como a implementação para absorver outros tipos de documentação das API's. Além disso, adicionar recursos gráficos para melhorar os relatórios ao final da execução do programa.

REFERÊNCIAS

IBGE – INSTITUTO BRASILEIRO DE GEOGRAFIA E ESTATÍSTICA. **Pesquisa Nacional por Amostra de Domicílios**. Rio de Janeiro: IBGE, 2021.

Tenable. **TENABLE'S 2021 THREAT LANDSCAPE RETROSPECTIVE**, 2022. Disponível em: https://www.tenable.com/cyber-exposure/2021-threat-landscape-retrospective. Acesso em: 1 mai. 2023.

Megavazamento de dados de 223 milhões de brasileiros: o que se sabe e o que falta saber. **G1** [online], 28 jan. 2021. Economia. Disponível em: https://g1.globo.com/economia/tecnologia/noticia/2021/01/28/vazamento-de-dados-de-223-milhoes-de-brasileiros-o-que-se-sabe-e-o-que-falta-saber.ghtml. Acesso em: 1 mai. 2023.

BEZERRA, Juliana. Revolução Industrial: o que foi (resumo). **Toda Matéria**, *[s.d.]*. Disponível em: https://www.todamateria.com.br/revolucao-industrial/. Acesso em: 4 mai. 2023.

ITRC - Identity Theft Resource Center. **Identity Theft Resource Center's 2022 Annual Data Breach Report Reveals Near-Record Number of Compromises**. 25 jan. 2023. Disponível em: https://www.idtheftcenter.org/post/2022-annual-data-breach-report-reveals-near-record-number-compromises/. Acesso em: 4 mai. 2023.

Equifax, empresa de crédito dos EUA, sofre ataque hacker e dados de 143 milhões de pessoas são expostos. **Por Agência EFE**, 7 set. 2017. Economia. Disponível em: https://g1.globo.com/tecnologia/noticia/equifax-empresa-de-credito-dos-eua-sofre-ata

que-hacker-e-dados-de-143-milhoes-de-pessoas-sao-expostos.ghtml. Acesso em: 4 mai. 2023.

FILHO, Nemesio Freitas Duarte; SERIQUE, Kleberson Junio do Amaral; PONTIN, Renata. Pesquisa Experimental. Apresentação do Power Point. Disponível em: https://escritacientifica.sc.usp.br/wp-content/uploads/MPCC_5_DataAnalysis06-PesquisaExperimental.pdf. Acesso em 8 jun. 2023.

REZENDE, Denis Alcides. **Engenharia de Software e Sistemas de Informação**. 3.ed. Rio de Janeiro: Brasport, 2005.

RODRIGO PREIS BEOCK, Leandro; DE FREITAS CONSONE, Cibele; RODRIGUES LIMA, Leandro; PETRICA, Eder. Protocolo HTTP. **UNIVERSIDADE DO ESTADO DE MATO GROSSO/UNEMAT**. Disponível em: https://www.vivaolinux.com.br/imagens/artigos/comunidade/Protocolo%20HTTP.pdf