FACULDADE DE TECNOLOGIA DE AMERICANA "MINISTRO RALPH BIASI"

Curso Superior de Tecnologia em Segurança da Informação

Wesley Martins Americo

Segurança em desenvolvimento de sistemas web

Americana,SP

2022

FACULDADE DE TECNOLOGIA DE AMERICANA "MINISTRO RALPH BIASI"

Curso Superior de Tecnologia em Segurança da Informação

Wesley Martins Americo

Segurança em desenvolvimento de sistemas web

Trabalho de Conclusão de Curso desenvolvido em cumprimento à exigência curricular do Curso Superior de Tecnologia em Segurança da Informação, sob a orientação do Prof. Dr. Henri Alves de Godoy.

Área de concentração: Segurança da Informação.

Americana, SP.

2022

Wesley Martins Americo

DESENVOLVIMENTO SEGURO DE SISTEMAS WEB

Trabalho de graduação apresentado como exigência parcial para obtenção do título de Tecnólogo em Curso Superior de Tecnologia em Segurança da Informação pelo Centro Paula Souza – FATEC Faculdade de Tecnologia de Americana – Ralph Biasi.
Área de concentração: Segurança da Informação

Americana, 21 de junho de 2022

Banca Examinadora:

Prof. Dr. Henrique Alves de Godoy (Presidente)

Prof. Renato Kraide Soffner (Membro)
Doutor
FATEC

Prof. André Ramalho dos Santos (Membro)
Mestre
FATEC

AGRADECIMENTO

A Deus por me conceder a vida e por permitir ultrapassar todos os obstáculos, ao meu saudoso pai que sempre me incentivou a buscar meus objetiovos, a minhã mãe e irmã que sempre estiveram comigo ao longo deste percurso sempre me apoiando,aos meus amigos que sempre me ajudaram e também deram contribuição valiosa para realização dos estudos, e também aos professores e em especial o professor Henri que sempre me ajudou ao longo de todo o graduação.

DEDICATÓRIA

Este trabalho é dedicado primeiramente à Deus, e a minha familia que sempre esteve comigo apoindo a realizar meus objetivos.

RESUMO

Este trabalho tem por finalidade explicar e demonstrar em forma teorica e prática, a importancia de desenvolver software de forma segura, levando em consideração todo o ciclo de desenvolvimente, que vai desde o levantamento de requisitos do sistema junto ao cliente, passando pela fase de desenvolvimento da arquitetura do sistema, resaltando a importância de não criar vulnerabilidades nesta fase,e posteriormente o trabalho falará sobre as principais fragilidades encontradas nas aplicações web e mostrará na pratica através de uma aplicação web simples que foi desenvolvida contendo algumas falhas, onde mostrará como os atacantes podem aproveita-las para obter vantagens sobre elas, também exibirá alguns métodos utilizados pelos atacantes e por fim será feita as as possiveis correções destas vulnerabilidades via codigo para manter o sistema seguro.

Palavras Chave: software, aplicações web, segurança da informação.

ABSTRACT

The purpose of this work is to demonstrate, in theoretical and practical terms, the importance of developing software in a safe way, taking into account the entire development cycle, which goes from the survey of client system requirements, through the architecture development phase. of the system, emphasizing the importance of not creating web vulnerabilities at this stage, and later it will work on the applications and will show in practice through a simple web application that was developed containing weaknesses, where it will show how some intruders can be used to obtain advantages over they, may also exhibit some methods by the attackers and may be made as possible that may be exposed to the secure system.

Keywords: software, web applications, information security.

LISTA DE FIGURAS

SUMÁRIO

1. INTRODUÇÃO

Com a crescente utilização de sistemas informatizados para realizar inumeros tipos de tarefas, que vão desde simples blocos de anotações de tarefas a complexas ferrametas de automação industrial, todas elas tem um ponto em comum, ou seja, todas são geradas através de softwares que são desenvolvido para atender cada necessidade, que são de usuarios comuns até grandes corporações.

Atualmente com maior acesso a internet os sistemas estão cada vez mais presentes em nosso cotidiano, e como também enfretamos uma grande pandemia onde muitas pessoas e comercios passram por grandes dificuldades, onde tivemos certas restrições, como por exemplo, muitas lojas fisicas não podendo abrir, funcionarios trabalhando em regime home office, entre outros muitos problemas enfrentados.

Com isso a demanda por sistemas informatizados cresceram também, muitas empresas passaram a trabalhar de forma virtual, para isso poder funcionar é necessario a implantação de softwares, entre eles sistemas web que é o foco deste trabalho, nele buscamos mostrar as muitas vulnerabilidades que eles possuem e também algumas possiveis correções para podermos ter um ambiente digital seguro.

O objetivo geral foi deste trabalho é mostrar os processos que estão envolvidos no desenvolvimento de sistemas web, mostrando algumas das principais vulnerabilidades que eles possuem e as possiveis conseguencias que essas vulnerabilidades podem acarretar para individuos ou organizações, também abordamos algumas metodologias que visam dimuir a quantidade de falhas e obtermos mais segurança em nossos sistemas.

Este trabalho tem como objetivo conscientizar desenvolvedores de sistemas a aplicar boas praticas de desenvolvimento de software para evitar possiveis danos tanto para o sistema, quanto para os usuarios e/ou clientes que possuem seus dados guardados e/ou processados pelas suas aplicalções e/ou tambem a para a

organização para poder ter maior credibilidade com relação ao mercado, por ter boas praticas de processamento e armazanamento de dados e evitar possiveis problemas com a legislação.

Este trabalho adotou metologiacamente, sistema de pesquisa bibliografica e o desenvolvimento de software web onde há algumas vulnerabilidades e mostrando como os atacantes podem aproveitar delas e também um software mostrando as possiveis correções e como as informações estão seguras.

2. Conceitos de segurança da informação e segurança de software

Para enteder as demais etapas que abordaremos neste trabalho, explicarei os conceitos basicos de segurança que, segundo Semola (2014, p. 41-42), segurança da informação é uma area do conhecimento dedicada à proteção de ativos da informação contra acessos não autrizados, alterações indevidas ou sua indisponilidade.

Assim sendo, Sêmola(2014, p. 41) diz que podemos considerar a segurança da informação como prática da gestão de risco de incedentes que provocam o comprometimento dos três pilares principais da segurança da informação, que são eles: confidencialidade, disponibilidade e integridade.

Confidencialidade: É onde somente usuarios que possui autorização podem ter acesso a informção desejada.

Disponibildade: Diz respeito a que toda informação deve estar disponivel a todo o tempo em que o usuario necessitar dela.

Integridade: Toda informação deve estar em sua forma original sem sofrer nenhuma alteração.

Além destes três pilares outro fator importante tanto para a segurança da informação, quanto para a segurança de software é a autenticidade que também é conhecida como autenticação, onde é verificado a autenticidade de alguem que deseja acessar a determinado sistema ou até mesmo local físico e vericar se ele pode ou não ter acesso a eles e quais locais ou informações podem ser visiveis e/ou acessados por eles.

Outro ponto inportante para a segurança é a privacidade, que em sistemas informatizados nada mais é do que a confidencialidade dos dados pessoais, tais como: nome, endereço, número de identificação de documentos pessoais, dados telefonicos, dados de sistemas bancarios, etc.

Outro fator que vale mencionar e segundo Sêmola(2014, p. 10) é de grande impotancia na segurança da informação são os quatros processos do ciclo de vida

da informação, independente de como a informação está armazanada, seja ela, em forma de bits, armazenadas em bancos de dados ou em algum outro tipo de sistemas de armazanamento de arquivo como planilhas, ou bloco de notas, armazanadas em papel, pode ser guardada em cofre, e/ou em átomos, por exemplo alguma formula de algum produto, elas terão estes processos ao longo de sua vida.

Manuseio: Instante em que a informação é criada ou manipulada, pode ser o momento de preenchimento de formulario com informação para serem armazenadas no banco de dados de um sistema informatizado, anotação e/ou folhear alguma agenda, senhas de acesso para autenticar em sistemas ou abrir alguma fechadura protegida por autencação.

Armazenamento: É o momento em que a informação será guardada em algum local podendo ser fisico como cofres, ou digital em algum sistema informatizado como em servidores de banco de dados.

Transporte: é o momento em que a informação é transportada de um local a outro, pode ser o momento de levar essa informação para o cofre de um banco, através de ligações telefonicas, mensagens de texto, email, trafego de mensagem via HTTP pela rede pelo browser de uma aplicação até servidor da aplicação, trafego de informações da aplicação até o servidor de aplicação, entre outros.

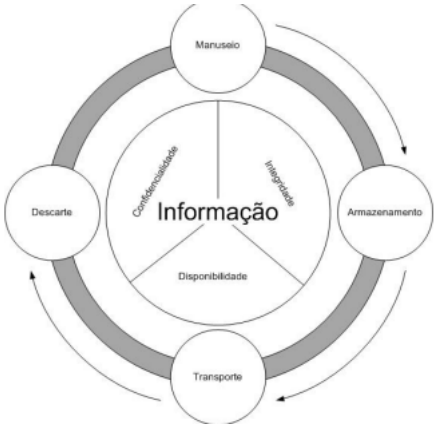
Descarte: Momento em que a informação é descartada, pode ser uma folha de agenda jogada no lixo, ou em uma trituradora de papeis, Hds ou SSDs de servidores de banco de dados, exclusão de informação em planilhas, etc.

Todos esses quatros pilares são importantes, porque o alvo é a informação, se ocorrer falhas em um desses conceitos, não adiantaria em nada garantir a segurança nos outros três, ou seja, se um desses falharem, como por exemplo, um cliente faz insere seus dados bancarios no ecommerce da loja, e para não ocorrer vazamentos da informação eles possui teclado virtual e caracteres especiais para não exibir senha e numero do cartão, no backend criptografa esses dados e envia para o banco de dados, o processo de descarte dos HDs dos servidores são corretos, mas no momento de trafego de informção entre o cliente e o servidor a informação não está criptografada e é enviada via url, de nada adiantou todo o trabalho restante, pois nesse exato momento houve vazamento de informção crucial.

A figura 1 exibe a relação entre os pilares da segurança e o ciclo de vida da

informação.

Figura 1 – PILARES DA SEGRANÇA VS CICLO DE VIDA DA INFORMAÇÃO.



Fonte: Adaptado de SÊMOLA (2014)

Estes pilares também são a base do desenvolvimento seguro de software que segundo Correia e Sousa(2017) a segurança de software diz respeito, à segurança do computador como um todo e não apenas a segurança de cada aplicação, mas sim a toda pilha de software, que vão desde o sistema operacional, bibliotecas, base de dados, isso inclui também o hardware e linguagens de programação.

Segundo o estudo que The Global State of Information Security que a PWC realiza anualmente, a edição de 2015 mostrou, que 91% das empresas que foram analisadas, terem solidas abordadegem a cybersegurança, foi notado um aumento significativo de 38% no numero de incidentes de segurança e 56% de roubo de propriedades intelectuais.

O que pode-se ter certeza é grande parte dos problemas de segurança são de vulnabilidades encontradas em softwares, sejam eles em configurações erradas, bugs de projetos, falhas em aplicar melhores praticas de codificação, das quais usuários mal intencionados se aproveitam e obtem vantagens, causando inumeros danos tanto para pessoas quanto a grandes corporações.

3. VULNERABILIDADES, ATAQUES E SEGURANÇA DE SOFTWARE

Este capitulo abordará algumas vulnerabilidades encontradas em software, explicação sobre o que é ataque sobre essas vulnerabilidades e algumas das metodologias utilizadas no mercado para desenvolver software de seguraça e prevenção de falhas.

3.1. Vulnerabilidades

A maioria dos sistemas utilizados no mundo possui alguma falha que pode ser explorada e comprometer o sistema, isso acontece porque nenhum sistema é 100% seguro, isso acontece porque sempre há atualização de sistemas, ou por falha em alguma parametro de configuração, conflitos de versões onde pode surgir falhas de compatibilidades e/ou falha humana seja ela voluntária ou involuntária.

Para Correia e Sousa(2017, p. 14) vulnerabilidade nada mais é do que um defeito encontrado no software que pode ser explorado por um atacante com o objetivo de violar a política de segurança. Dentro do conceito de vulnerabilidade existe três categorias principais, são elas:

Vulnerabilidade de projeto: esta vulnerabilidade ocorre na fase desenvolvimento do projeto, onde não são levados em consideração alguns pontos segurança, tais como: fator de autenticação fraco, separando prioridades de acesso, politicas de senhas robusta, senhas armazenas em forma de texto no banco de dados, a comunicação pode ser observada durante o trafego na rede, entre outros.

Vulnerabilidade de codificação: ocorre durante a fase de desenvolvimento do software, ou como é conhecida como codificação do sistema, ela é causado por desenvolvedores que dependendo da linguagem de programção não se atenta para alguns problemas de segurança que ele pode causar quando utiliza os recursos de forma errada e/ou utiliza os recursos de forma mais facil, tais como não verificar se há atualização no pacote da biblioteca que ele está usando, falta de verificação na escrita de buffer, se o código permite injeção, etc.

Vulnerabilidade operacional: está vulnerabilidade está associada ao ambiente

no qual o sistema está sendo executado e/ou configurado, no caso de problemas de configuração pode ser tanto do software quanto do ambiente, como por exemplo: serviço para colocar a sistema online com inumeras configurações erradas, sem senha de autenticação, portas de acesso abertas sem necessidade.

Vale ressaltar que apenas as vulnerabilidades em si não causa danos, elas precisam de um ataque que cause quebra em um ou mais pilares da segurança da informação e comprometa a informação ou acesso a ela. Quando isso ocorre chamamos esse processo de intrução, isto é, o ataque feito com sucesso e ativou uma ou mais vulnerabilidades do sistema.

O foco desde trabalho é nas vulnerabilidades de codificação, já que inumeros sistemas são comercializados com a proposta de resolver algum problema especifico, e junto com eles também vão inumeros bugs. Segundo Correia e Sousa(2017, p. 15-16) uma estimativa aceita no mercado é que qualquer pacote de software tenha entre 5 e 50 bugs por cada 1000 linhas de códigos.

No entanto as vulnerabilidades nos sistemas estão sujeitos a serem descobertas, pois na maioria das vezes quem desenvolveu nem sabe que as possui, após o descobrimentos de alguma vulnerabilidade uma boa prática utilizada é avisar a empresa responsável pelo sistema e eles irão gerar a correção para estas vulnerabilidades, e posteriormente elas serão classificadas e catalogadas.

O grande problema é que os fabricantes realizam os remendos nos sistemas, mas uma grande maioria de administradores não fazerm a atualização delas em seus sistemas, é ai que começa os grandes problemas. Em 2012 houve um estudo sobre o ciclo de vida das vulnerabilidades indicando que a percentagem de vulnerabilidades exploraveis remotamente tinha crescido para mais de 80% do total, Correia e Sousa(2017, p. 18).

3.2. Ataques

De acordo com Correia e Sousa(2017 p. 20) os ataques a softwares podem ser classificados de duas formas, os ataques feitos através de engenharia social, no qual o atacante tenta enganar suas vítimas de alguma forma, podendo ser se passando por outra pessoa no telefone com finalidade de pegar alguma informação sigilosa, ou através de phising, no qual se envia mensagem no email da vítima pedindo acesso aos dados dela, ou enviando um link de alguma pagina web parecida com a original, mas com endereço de url diferente e muitas das vezes replicando a aparencia de alguma aplicação já conhecida do público e se passando por elas, como por exemplo: um clone de alguma rede social, sites bancários, market places, entre outros. Temos tambem os ataque técnicos, para executar esse tipo de ataques os atacantes exploram as vulnerabilidades do sistama alvo, podendo ele ser manual ou utilizando alguma ferramenta ou scripts para automatizar seus processos, as formas e locais dos quais eles exploram os alvos podem ser das mais variadas possiveis, podem ser através de códigos maliciosos instalados na máquina, tambem conhecido como malware, backdoor instalados em sistemas, etc.

3.3. Segurança de software

Quando um desenvolvedor ou uma empresa procura desenvolver um sistema, o primeiro objetivo buscado é que ele tenha a funcionalidade de ser a solução para alguns objetivos ou problemas do cliente, isto é, capacidade de executar uma série de tarefas e segundo Correia e Sousa (2017, p. 30) os usuários comumente avaliam a qualidade dos produtos pela quantidade de tarefas em que ele realiza, alguns também gostam de avaliar a aplicação por ela ter uma boa aparencia e/ou boa usabilidade.

No entanto muitas das vezes a segurança passa ser visto como obstáculo no desenvolvimento, porque costumam dificultar a vida dos usuários, como por exemplo é necessário utilizar senhas complexas e não deve ser compartilhada com ninguém e também não pode ser guardada em locais onde outras pessoas possam ter acesso,

também há a necessidade da autenticação de dois fatores, que no caso o usuário pode escolher para qual aplicação deseja enviar a autenticação, podendo ser via sms, email, ligação telefônica, ou até mesmo um software para autenticação, com relação aos desenvolvedores, também deve ter mais cuidado com a forma que escreve seus codigos, pensando sempre na melhor forma de não permitir acesso a softwares terceiro ou injeção de códigos maliciosos externo que visam comprometer o sistema e pensar também em não comprometer o desempenho do software devido a grandes processamentos por causa das fortes criptografias utilizadas para não permitir que os dados sejam trafegados as claras na rede, entre outras formas de segurança, e por fim um fator não menos importante é que todo esse processo também acarreta em maior custo no desenvolvimento elevando o tempo para a chegada do produto ao mercado e o preço a ser pago pelo usuário final é maior.

Por isso sempre é ter um concesso entre todas as áreas envolvidas no processo, para isso existe alguns processos de desenvolvimento de software cuja a finalidade é reduzir os riscos para níveis aceitáveis, onde é avaliado o impacto, ou seja, o custo que estas falhas podem causar a empresa em caso de ataque, a exposição que estes sistemas sofrerão quando estiverem em utilização na internet, por exemplo não há necessidade de gastar muito com segurança em um software que resolverá um pequeno problema de uma residencia e/ou comercio onde quase ninguem sabe que ele existe, e o oposto disso também é válido, pois em um sistema de grande porte que possui muito acesso e também é de conhecimento de muitos usuários, caso há alguma falha a empresa desenvolvedora terá que pagar indenização aos clientes.

Dentre algumas metodologias importantes utilizadas por grandes empresas de softwares destaca-se algumas, não entrando no mérito de como elas são aplicadas, pois não é o foco deste trabalho discutir sobre elas. De acordo com Correia e Sousa(2017, p. 36-44) são elas: Microsoft Security Development Lifecycle, Segurança no Desenvolvimento Ágil, Verificação e Validação – Norma IEEE 1012-2012, Buiding Security in Maturity Model, Segurança Aplicacional – Norma ISO/IEC 27034.

4. SEGURANÇA E DESENVOLVIMENTO DE SOFTWARE

Software é um instrumento fundamental para automação de processos em diversas áreas da sociedade, atualmente eles são responsáveis para que inúmeras tarefas sejam realizadas, que vão desde enviar uma simples mensagem de texto até as complexas linhas de montagens em grandes corporações.

Podemos dizer que o software é uma ponte entre o computador e o humano que o utiliza, pois através deles hoje somos capazes de reduzir o tempo de produção de produtos, serviços, onde compra-se algo da propria casa sem precisar perder tempo em transito e também diminuir custos operacionais, entre outras diversas usabilidades.

Para que estes softwares existam e possa nos auxiliar, é necessario sua construção e para isso passa-se por um processo de desenvolvimento, onde os profissionais encarregados para este fim, são conhecidos como analistas e desenvolvedores sistemas e por sua vez, são responsáveis pelo levantamento de requisitos junto ao cliente, esse processo é de extrema importância pois é necessário levantar todas as informações importântes das tarefas que ele desempenhará, para quando chegar o momento da criação da arquitetura e codificação não desenvolver algo diferente do esperado pelo cliente causando prejuizos para ambas as partes.

Após a fase de levantamento de requisitos, os dados passam por uma série de análises e criações até chegar ao objetivo do trabalho, que é a codificação, isto é, a fase onde os softwares realmente são construidos, para que isto ocorra estes profissionais utilizam diversas linguagens de programação e variados tipos de framework, nesta fase deve-se ter bastante atenção com a forma de desenvolvimento das funções, observar atentamente se os pacotes, bibliotecas estão atualizadas, ou se não possuem algumas vulnerabilidades, como veremos a seguir.

4.1. Inicio do projeto

Quando iniciar um projeto para desenvolvimento de software é necessario atentar para não desenvolver um projeto com vulnerabilidades, por isso, é importante fazer uma boa análise de requisitos, para quando for criar a arquitetura do sistemas conseguir desenvolve-la sem vulnerabilidades do projeto, Correia e Sousa(2017 p. 44) diz que existe alguns principios de projetos que quando utilizados evitam os erros mais comuns em projetos de sistemas.

Considerar segurança desde o início: este princípio era importante ser falado há decadas atrás quando segurança ainda não era tema principal de requisitos das empresas, mas ainda hoje é importante falarmos sobre ele, e dizer que a segurança deve ser levada a sério e implementada desde o início do projeto, utilizando-a desde o começo teremos visão do que deve ou não ser realizado no processo de desenvolvimento e também dimuir custos do projeto, uma vez que não será necessário fazer grandes revisões e mudanças em termos de segurança no projeto.

Princípio de ecônomia de mecanismos: este processo visa deixar o projeto da forma mais simples possível, ele adota está prática porque considera que quanto mais complexo fica o projeto, mais vulnerável ele pode se tornar também, com isso deve-se deixar o projeto o mais simples possivel no que diz respeito a mecanismos de segurança, como por exemplo: validações de entradas de usuários, outro ponto interessante é que as aplicações tenham interfaces de fácil entendimento e usabilidade para que os usuários consiga aplicar as medidas de segurança necessarias.

Princípio por decisão segura por omissão: este princípio diz que deve estabelecer permissões e nao atribuir proibissões, isto é, deve negar o acesso e apenas conceder por decisão explícita, porque desta forma consegue ter maior segurança nos sistemas uma vez que só será permitido acessar aquilo que for permitido, o contrário disso seria aplicar regras de proibições aos usuários, o que não é interessante do ponto de vista da segurança, uma vez que existe muitos dominios, ou muitas funções do sistema a serem proibidos, e com isso acabaria

esquecendo de aplicar proibições a todos e eles.

Mediação completa: este princípio diz que devemos verificar todos as autorização de acessos que estão sendo realizado a qualquer objeto do sistema.

Princípio do projeto aberto: este princípio nos diz que o desenho do software não deve ser segredo, é não pode deixar que a segurança do software seja por meio do segredo do projeto, ou seja, ninguem saber a arquitetura do projeto, então conclui o software está seguro, o que na verdade é enorme engano, porque hoje existem ferramentas de engenharia reversa no qual pode-se decompilar o codigo do software, e através disso encontrar as vulnerabilidades.

Princípio da separação de privilégios: este princípio está relacionado a autenticação nos sistemas, ele diz que um sistema com multiplas autenticações é mais dificil de ser quebrado do que utilizando apenas uma, um exemplo prático do cotidiano são as portas que possui mais de uma fechadura e para abri-la são necessarias duas chaves, nos softwares também deve ser utilizado este mesmo conceito, como por exemplo, uma senha que o usuário conheça e também pedir um token gerado por um aplicavo confiável.

Princípio dos privilégios minimos: diz que a dependência de processos e recursos compartilhados devem ser minimizados, isto porque tudo que pode ser compartilhado ou transferido através de dados de um local para outro, causa grandes problemas de segurança.

Princípio de segurança na instalação: este princípio fala que o sistema deve ter documentação para o admistrador do sistema poder configurar ele corretamente, sistemas também devem conter ferramentas para administração e análises de segurança, uma delas que não deve faltar é um excelente sistema de logs para verificar o que está acontecendo na aplicação e quem está executando determinada função e também ter ferramentas que permite o administrador configurar o nível ideal de segurança no software.

Princípio da comunição: este princípio está ligado a comunição entre as

equipes de desenvolvimento, implantação, manutenção do sistema e ele nós diz que deve existir processos de comunição para respostas de vulnerabilidades que venha a existir nos sistemas e através deles serem estabelecidos correções, esses meios de comunição também são interessantes para sanar as possiveis dúvidas que existirão sobre a aplicação.

Princípio defesa em profundidade: este princípio fala que no sistema deve haver mais de um mecanismo de proteção, caso um falhar existam outras maneiras para executar a segurança do sistema.

Princípio proteger o elo mais fraco: este é um princípio que estamos acostumado a observar no dia a dia, que o elo atacado na maiorida das vezes é sempre o mais fraco, é nele onde os problemas geralmente acontecem e em software não seria diferente, quando um atacante vai tentar atingir a vulnerabilidade do sistema eles obviamente vão tentar pela parte mais fraca, por exemplo: eles não tentarão quebrar a criptografia do sistemas, na maioria das vezes utilizarão ataques de engenharia social para pegar crendiciais de usuários com pouco conhecimento de segurança ou realizar um ataque para tentar quebrar as credênciais de acesso do usuário.

Princípio falhar de forma segura: que não exite software 100% seguro isso já estamos cansados de saber, mas o que este princípio quer mostrar é quando um sistema falha, o sistema deve ficar de forma segura e não deixar brechas para um atacante explorar suas vulnerabilidades.

4.2. Evitando vulnerabilidades de projetos e codificação

Para evitar vulnerabilidades de projetos deve adotar a pratica de nunca assumir ou confiar, segundo Correia e Sousa (2017, p. 47) os sistemas de software composto por mais do que um único componente monolítico confiam na composição e cooperação de dois ou mais componentes ou camadas de software para cumprirem adequadamente seus objetivos. E estes sistemas serão inseguros se

alguns destes componentes ou camadas forem executados em ambientes hostis.

As aplicações web são um exemplo clássico de software de pressuposto errado de confiança, isto acontecem devido a confiança do código que rodam nas maquinas clientes, ou seja os browsers.

Para Correia e Sousa (2017, p. 153) uma regra de ouro para desenvolvimento seguro de software é nunca confiar em nas entradas, isto é, nunca confiar em dados recebidos de interfaces, sejam elas, sockets, web sockets, APIs, arquivos utilizados pela aplicação, dados recebidos do sistema operacional, dados de entradas vindo usuários, porque esses dados recebidos são considerados superfície de ataque, e todos os dados que são recebidos dessas superfícies devem ser cuidadosamente validadas antes de serem processadas.

Outro fator importantissimo para desenvolvimento seguro de software é cuidar da arquitetura do projeto, após a fase levantamento de requisitos, e ter todas informações válidadas e aprovadas, inicia a fase de arquitetura do projeto, onde é escolhida a arquitetura que atenderá melhor as necessidades do cliente, por exemplo, se a solução ideal para atender aquele requisito é um sistema desktop, ou sistema web com MVC no qual não é necessario criar uma API integrada com frontend, ou se o sistema é maior e será necessario criar API.

Essas análises são importantes porque se começar a desenvolver um sistemas sem desacoplamento de definições de responsabilidades, o software ficará todo embaraçado e não saberá onde estará determinadas funções, sem contar também que dará muito trabalho para manutenção futura do sistema, também não terá usabilidade de código e para cada nova função reescreverá o mesmo código e quando for necessario fazer alteração terá que fazer em muitos lugares diferentes, tudo isso sem contar que se não tiver um código bem estruturado não será possivel pensar na criação de estrutura para segurança do sistema.

Outro fator que também causa grandes vulnerabilidades nos softwares são causados por falha dos programadores, conhecida como falha por inoscência ou implementação inoscente, onde eles escrevem seus códigos e muitas das vezes por

não saberem ou não aplicarem as melhores práticas de codificação acabam escrevendo códigos com vulnerabilidades.

Por fim, no momento de implantar o sistema, principalmente em serviços clouds, deve estar atento se existe códigos maliciosos dentro da imagem docker, quando instalado nos servidores terá esta vulnerabilidades rodando dentro das microcapsulas, para que isso não ocorra deve sempre utilizar imagens de fontes oficiais e quando colocar o código em ambiente produtivo executar scanners que análisa código fonte para encontrar falhas já conhecidas tanto no código fonte quanto nas imagens. E também verificar os pacotes utilizados na aplicação, pois eles podem conter algum problema de segurança ou estar em versões desatualizadas, ter sccaner para todas as dependencias utilizadas, e sempre atualizar a biblioteca.

4.3. Autenticação

Segundo Correia e Sousa(2017, p. 49) autentição é ato de validar a identidade de uma entidade. E seu principal objetivo é previnir que uma entidade ganhe aceso ao sistema ou serviço sem antes de autenticar, e após autenticado o sistema deve garantir que essa entidade não possa trocar de identidade sem se autenticar.

Quando for criar uma identidade para uma entidade no sistema é de extrema importância que tenha criptografia de forma irreversiva, isto garante que os dados não possam ser lido por atacante durante o transporte da informação na rede e também nenhum usuário que tenha acesso ao sistema de banco de dados possa olhar as informações de login das entidades.

Para um sistema possuir uma autenticação segura é necessario ter uma autentição de multiplos fatores, caso uma delas sejam violadas o atacante necessita ter acesso os demais fatores de autenticação para conseguir entrar no sistema, por exemplo o sistema permitir que o usuario crie uma senha complexa que siga boas

politicas de senhas e também ter outro fator de autenticação, por exemplo, enviar uma senha por sms, ou o mais interessante é utilizar softwares já consolidados no mercado para autenticação, estes softwares possuem mecanismos confiaveis e também tem equipes trabalhando em melhorias constantes para prevenir ataques de roubos de credenciais e resolver as vulnerabilidades existentes.

Outro fator que deve ser seguido, não apenas pessoas devem fazer a autenticação no sistemas, mas tambem outras interfaces que se conectam a aplicação e fazem comunicação entre si, esse procedimento evita sistemas de acessarem o software e fazer coleta e/ou causar violção na base de dados.

Também deve estar atento que os sistemas devem autorizar explicitamente as operações pelos utilizadores autenticado, um falha muito comum que ocorre nos sistemas é um usuário com acesso básico conseguir acessar funções ou arquivos de outros usuários, por exemplo, o usuário x trabalha como asistente financeiro em uma determinada companhia e quando ele se autentica no sistema ele tem acesso a determinas funçoes no sistema, e tambem pode ver seus rendimentos da empresa, mas se ele trocar alguns parâmetro da url do navegador ele passa ter acesso a dados de outras pessoas, causando violção no pilar de integridade do sistema.

Como vemos no caso acima, segurança não é apenas proteger dados de acessos ilegais, mas também é necessario se preocupar com quem tem privilégios para acessar determinas funções, porque os maiores problemas de segurança vem de dentro das empresas e não de fora e uma falha grave que poderia ocorrer neste erro de estrutura é alguem com conhecimentos básicos e de forma maliciosa criar um robô que consiga ir trocando os parâmetros dos usuários e fazer dump na base de dados dando a ele todos os dados da empresa, essa é uma das formas que atacantes conseguem roubar informações das empresas e posteriormente fazer o vazamento de dados.

4.4. Criptografia

Para Correia e Sousa (2017, p. 50) criptografia é uma das ferramentas mais importantes para construir softwares seguros, é através dela que podemos ter o pilar da confidencialidade e proteger dados de modificações não autorizadas, entratanto ela é uma das ferramentas mais complexas de ser utilizada, e muita das vezes deve se recorrer a um especialista para utiliza-la.

Uma coisa que deve ter em mente é: não invente a roda, cada framework de desenvolvimento tem suas classes especializadas em criptografia e os desenvolvedores devem apenas implementar em seus sistemas, o que causa vulnerabilidade nos sistemas são que muita das vezes os desenvolvedores com algum conhecimento na área criam suas próprias criptografias e colocam nos sistemas, causando grandes problemas futuros.

5. DESENVOLVIMENTO DE APLICAÇÃO WEB

Neste capítulo é apresentado o desenvolvimento de uma aplicação web com algumas das principais vulnerabilidades de acordo com o projeto OWASP, e também é apresentado algumas soluções para corrigir as respectivas vulnerabilidades.

5.1. Ambiente de desenvolvimento

O sistema foi desenvolvido com a linguagem de programação Javascript através do framework Node com pradrão de arquitetura MVC, ou seja um sistema monolitico com renderização server side e integrado ao banco de dados MYSQL.

5.2. Definição de sistema web

Com a expanção da conexão com a internet, atualmente a maioria dos sistemas desenvolvidos utilizam esse mecanismo, pois ele permite que vários clientes, também conhecido como navegadores ou browser consiga acessar os servidores de aplicação web e executar as funções necessárias.

As aplicações web melhoram a vida das corporações porque não necessita ter um software instalado em cada máquina e os usuários conseguem acessar as aplicações remotamente.

Com isso pode dizer que Aplicações web são sistemas armazenados e/ou executados em um mais servidores, que tem como função processar os dados recebidos através de requisições recebidas de clientes (browser) por meio da rede de comunicação (internet) utilizando de protocolos, em especial o HTTP, ela funciona através de requisições que são enviadas do cliente para o servidor,

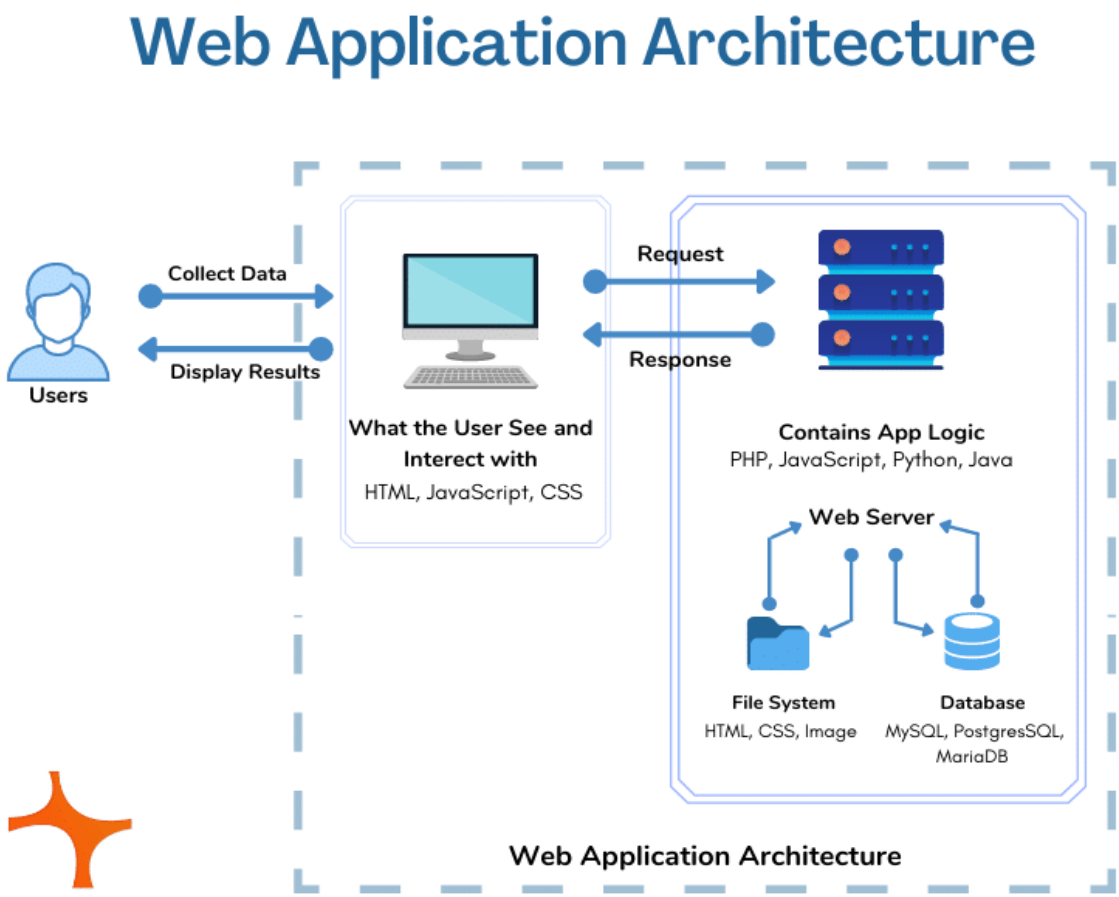
conhecida como request, e por meio de respostas que o servidor envia para o cliente, também chamada de response.

Aplicações web muitas vezes são confundidas com páginas web ou sistemas web tradicionais, mas possui muitas diferenças, entre elas podemos destacar, que os sistemas tradicionais muitas vezes podem ter conteúdos estáticos, e geralmente mostram os mesmos conteúdos para todos usuários, já as aplicações web são desenvolvidas com objetivo de fornecer conteúdos personalizados de acordo com perfil de cada visitante, como por exemplo, o Facebook, as permissões de usuários são algo necessário, a usuabilidade também pode ser diferente de acordo com o perfil de cada usuário, as pessoas podem ter maior iteração com o sistema, como por exemplo, manipular informações na base de dados, realizar compras, ter descontos em produtos conforme a quantidade de compra de cada cliente, entre outras muitas funções que a aplicação web pode fornecer a seus clientes.

Entretando como as aplicações web estão a disposição para acesso de inúmeros clientes, ela também é alvo de maior quantidade de ataques, e também é necessário a combinação de várias tecnologias para faze-las funcionar corretamente, tornando-as complexas, pois existem diversos navegadores para acessar os servidores, que por sua vezes dependendo do tamanho do negócio da empresa, pode ter dezenas ou até centenas de máquinas para rodar suas aplicações, isso sem contar a quantidade de plugins e scripts que buscam manipular informações no frontend da aplicação (CSS,HTML, JavaScript), de acordo com Correia e Sousa(2017, p. 171) todas essas junções de componetes faz as aplicações web se tornarem extensiva, formando uma trindade perigosa e aumentando o risco, isto porque complexidade, conectividade e extensibilidade aumentam a existência de vulnerabilidade e por sua vez a conectividade aumenta o nível de ameça.

A figura abaixo demonstra a arquitetura de uma aplicação web.

Figura 2 - ARQUITETURA DE APLICAÇOES WEB.



Fonte: https://cynoteck.com (2021)

5.3. HTTP

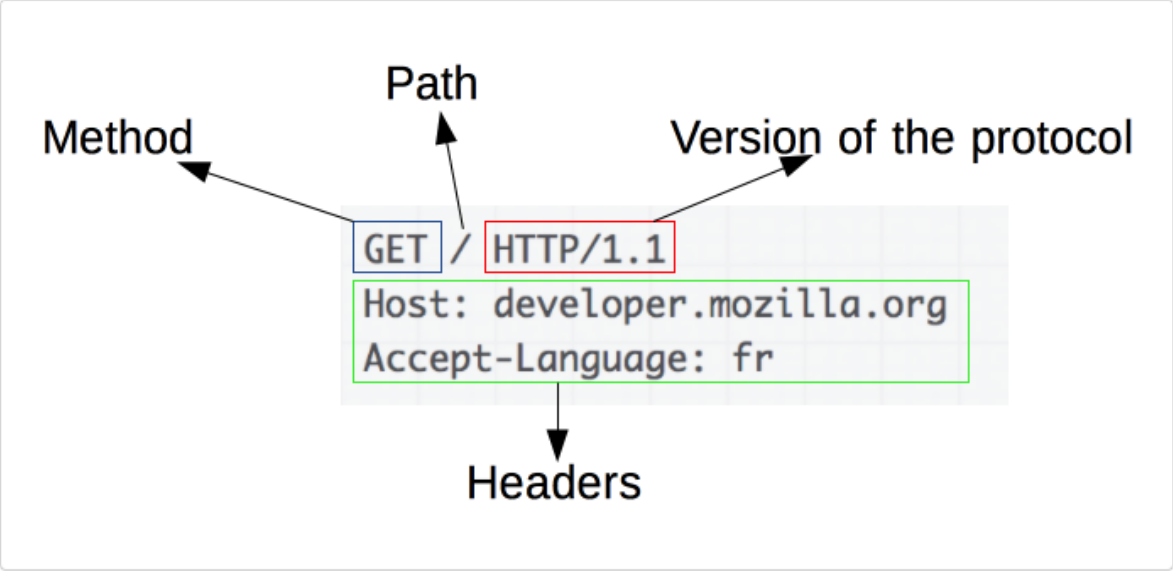
HTTP é o protocolo de comunicação mais utilizado pelas aplicações web, é através dele que os sistemas obtem os recursos para funcionamento, o principal recurso utilizado é o HTML, ou seja, o HTTP é base de comunicação e troca de dados na web, ele também é um protocolo cliente-servidor, as requisições sempre são inciadas pelo cliente, ou seja, na maioria da vezes um navegador web (isso porque existe outras formas de clientes, mas pouco utilizado).

Cliente: para que uma pagina web seja exibida, é necessário o browser enviar uma requisição (“resquest”) para buscar o documento HTML, se o servidor estiver utilizando uma aplicação monolítica, ou buscar um arquivo JSON se estiver

buscando dados de API, após o processamento o servidor envia uma resposta para o browser de acordo com a solicitação e o browser mostra a informação em HTML para o usuário.

A figura abaixa mostra o protocolo HTTP enviando uma request para o servidor

Figura 3 - EXEMPLO DE UMA REQUISIÇÃO HTTP.

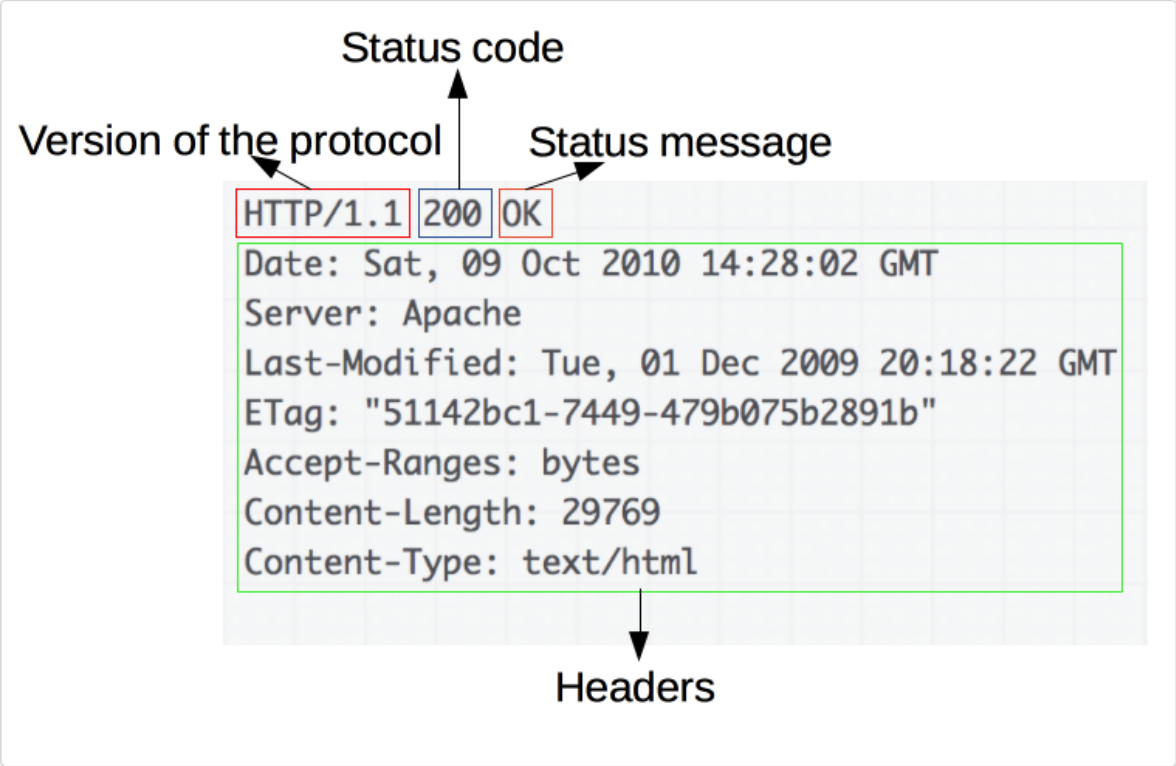


Fonte: https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Overview (2022)

Servidor: quando o browser envia uma solicitação para um determinado documento, quem recebe essa solicitação é o servidor, que por sua vez processa o pedido do cliente, através de suas funções desenvolvidas de acordo com a necessidade do negocio, após o processamento o servidor envia uma resposta (response) de acordo com o pedido do cliente.

A figura 4 mostra o protocolo HTTP enviando uma response para o cliente.

Figura 4 - EXEMPLO DE UMA RESPOSTA HTTP.



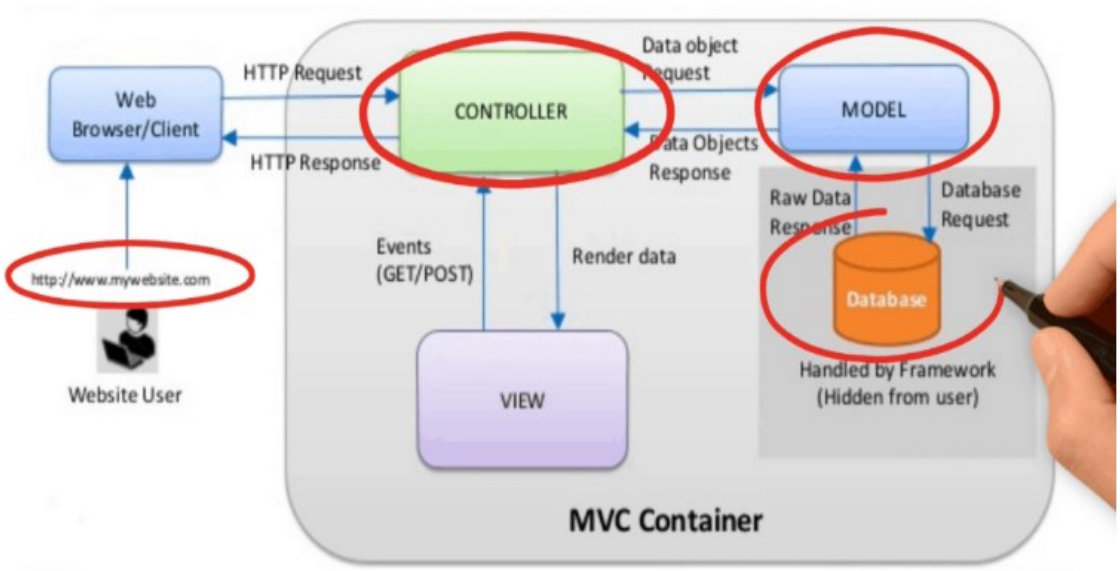
Fonte: https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Overview (2022)

5.4. MVC

MVC é uma arquitetura de desenvolvimento de sistemas web desenvolvido em três camadas, que trabalha através de requisições baseadas no protocolo HTTP, e busca melhorar a forma de desenvolvimento de sistema, separando as responsabilidades, tornando a aplicação escalável, tendo maior organização dos arquivos, facilitando o reuso do código e futura manutenção da aplicação quando for necessário.

A figura abaixo mostra o funcionamento e a divisão das tarefas na arquitetura MVC.

Figura 5 - ARQUITETURA MVC.



Fonte: https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Overview (2022)

Controller: o controlador é o resposável por tratar as requisições do protocolo HTTP, sua função é como se fosse de um porteiro, pois ele permite ou rejeita as requisições, validando se o usuário tem a permissão para executar determinada função, se o usuário e a entrada for válida, é por ele que os models são chamados para processarem as informações desejadas e posteriormente se necessário também faz as chamadas para que as respectivas views renderizem seus templates e devolve as informçãoes para os clientes.

Model: após a validação do controller é chamado um model, que nada mais é do que as regras de negócios definas no sistema, ele é responsável por executar as funções do sistema, caso tenha necissidade ele também e responsável por manipular e acessar o banco de dados, podendo realizar transações de cadastro de informações no banco e/ou trazer informações de alguma tabela, após fazer essas tarefas ele pode ou não enviar dados para o controller, que por sua vez esses dados será enviados para as views.

View: após o model realizar sua tarefa e devolver as informações para controller, ele chama as respectivas views e elas tem por responsabilidade renderizar os templates, ou seja, criar layout que o usuario visualizará em seu navegador, após criar os dados ela devolve as informções para o controller que por

sua vez envia através de response do protocolo HTTP para o cliente responsável que exibirá as informações para o usuário final.

5.5. As top 10 vulnarabilidades da OWASP

O objetivo do projeto OWASP tem por finalidade a publicação periódica das 10 vulnerabilidades que presentam o maior risco para as organizações em suas aplicações web.

O projeto teve inicio em 2003, a publicação que está sendo utilizada no projeto é do ano de 2021, a lista das top 10 vulnerabilidades destaca as 10 falhas mais perigosas das aplicações e mostra os métodos mais eficazes para mitigar os respectivos riscos, e são analisadas através de colobaração de especialistas os riscos críticos no desenvolvimento de aplicações.

A tabela a seguir mostra a mudança das vulnerabilidades dos anos de 2017 e 2021 e as principais mudanças entre as duas avaliações.

Figura 6 - ARQUITETURA MVC.

Fonte: https://www.gcsec.com.br/owasp-top-ten-2021-seguranca-de-aplicacoes-web/index.htmlOverview (2021)

Antes de mostrar aplicação web implementanda com as falhas e as possíveis correções, será mostrado um resumo sobre as top 10 vulnerabilidades da OWASP.

AO1: 2021-Quebra do controle de acesso: As falhas de controle de acesso

subiu da quinta para a primeira posição, esta falha faz com que os usuários consiga operar fora de suas permissões prevista, é através desta falha que ocorre as divulgações de dados não autorizados, também leva a destruição ou modificação dos dados e 94% aplicações foram testadas para alguma forma de comprometimento de controle de acesso.

A02: 2021-Falhas Criptográficas: está falha subiu uma posição no ranking, anteriormente era conhecida como Exposição de Dados Sensíveis, esta vulnerabilidade é mais uma consequencia do que causa raiz, ela retrada a exposição dos dados sensíveis, é considerado uma consequência devido os dados serem expostos mais por falta da criptografia durante o transporte e/ou armazanamentos dos dados.

A03: 2021-Injeção: esta vulnerabilidade caiu da da primeira para a terceira posição, e da mesma forma que a primeira falha da OWASP cerca de 94% da aplicações foram testadas e teve 19% de incidência, a revisão de código é a melhor forma de previnir esse tipo de falha e os tipos mais comuns são: relacionados a injeção de códigos de banco de dados, como SQL.

A04: 2021-Design Inseguro: esta foi uma nova categoria adicionada na versão 2021, o foco dela está no design e modelagem dos padrões de arquitetura dos projetos de software, e está associada a falta de determinar qual o nível de segurança o sistema vai precisar no momento do desenvolvimento.

A05: 2021-Configuração Incorreta de Segurança: esta vulnerabilidade ganhou uma posição desde a versão anterior e devido a falta de processos de configurações de segurança nas aplicações consistentes e que podem ser replicados faz com que as aplicações fiquem expostas a riscos.

A06: 2021-Componentes Vulneraveis e Desatualizados: era anteriormente entitulado como: usando componentes com vulnerabilidades e está relacionada a componentes desatualizados nos sistemas, ela fala qua as organizações devem adotar um método com plano de monitoramento e aplicação continua das atualizações durante toda a vida útil dos aplicativos.

A07: 2021-Identificação e Falhas de Autenticação: nas versões anteriores era descrita como quebra de autenticação, confirmar a identidade do usuário, gerenciamento de sessão são essenciais para proteção contra este tipo de ataque.

A08: 2021-Falhas de Integridade de Software de Dados: está é uma nova categoria incluída nesta nova versão e está associada a fazer atualizações de softwares e dados críticos e pipelines de CI e CD.

A09: 2021-Falhas de Registro e Monitoramento de Segurança: Impacta diretamente a visibilidade, o alerta de incedente e a perícia, está vulnerabilidade está associada a falta de logs, para os responsáveis saberem o que está ocorrendo nas aplicações e também quem teve acesso e/ou causou alguma mudança nas informações.

A10: 2021-Server Side Request Forgery: essa vulnerabilidade está relacionada a toda vez a aplicação web busca recursos remota sem validar a url de destino, ela consegue conduzir a vítima a executar ações não desejada em uma página web vulnerável.

5.6. Aplicação WEB

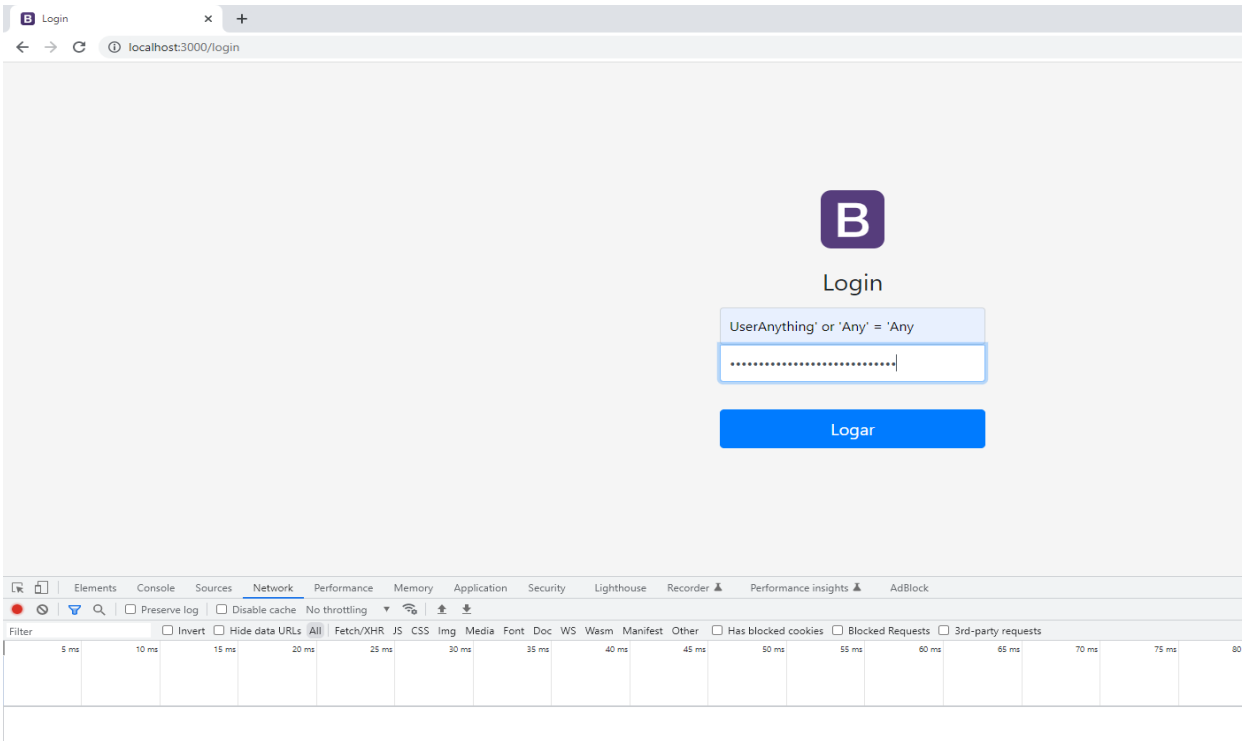
5.6.1. Aplicação com vulnerabilidade de sql injection

Os problemas de injeção em sistemas podem ser de vários tipos, neste trabalho o foco é em injeções de comando SQL, estes dados não confiáveis são recebidos através do browser, são injetados pelos usuários que tentam enganar o interpretador para executar comando indesejáveis ou possibilitar a entrada em sistemas sem possuir credenciais de acessos, executar leitura, criação, manipulação ou exclusões de informações na base de dados.

A figura 7 e 8 mostrará um usuário injetando código SQL para poder acessar

o sistema sem possuir credenciais de acesso.

Figura 7- ATAQUE DE SQL INJECTION.

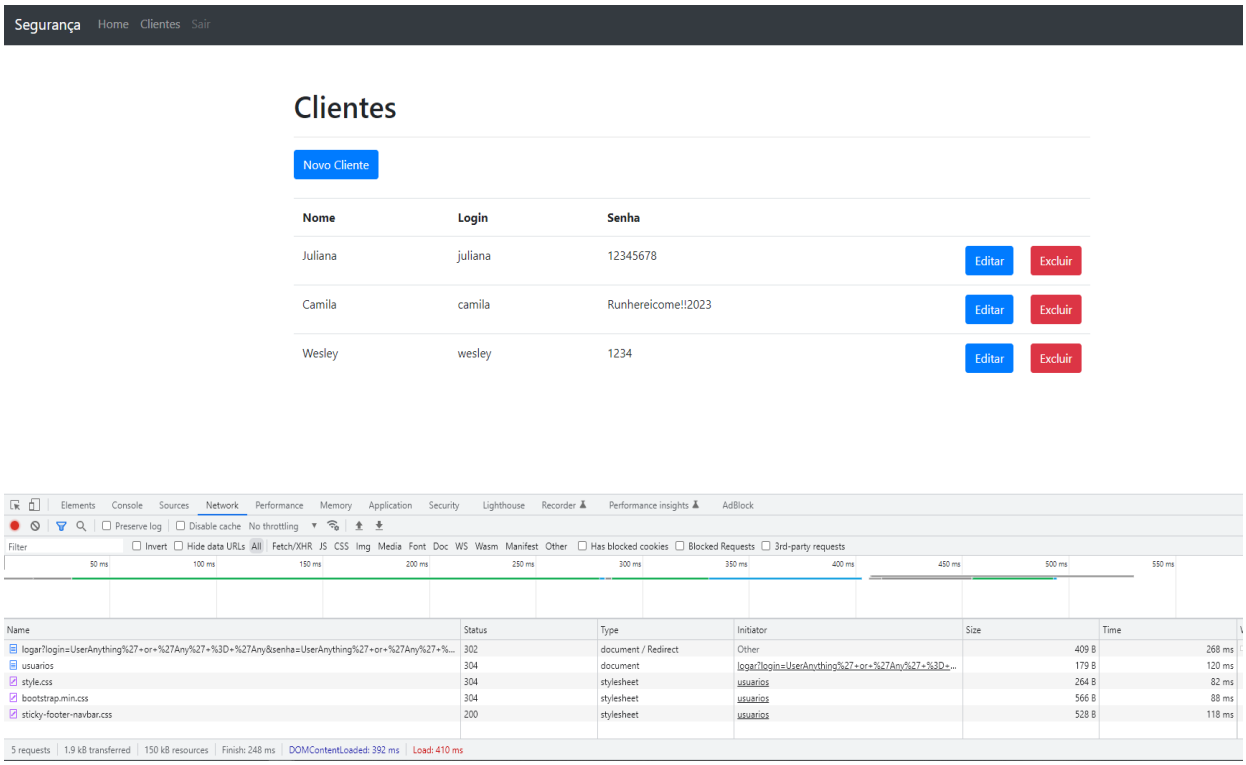


Fonte: Autoria Propria (2022)

Como pode ver, o usuário informa um login qualquer nos campos e insere alguns parâmetros de sql, onde os valores sempre serão verdadeiros, dessa forma se o sistema não possuir uma codificação que não permita inserção de injeção de códigos, esse código irá para query sql e será executado no interpretador do banco de dados.

A figura 8 mostrará que o ataque foi bem-sucedido e o usuário conseguiu acessar a área logada do sistema.

Figura 8 - ATAQUE BEM-SUCEDIDO.



Fonte: Autoria Propria (2022)

Este tipo de ataque aconteceu porque o sistema tem uma vulnerabilidade no momento da criação da variável que verificará se as informações passadas pelo usuário realmente são validas e se essas informações realmente estão cadastradas na base dados, como o usuário está passando uma expressão válida, no caso or 'Any' = 'Any, o sistema recebe os dados do browser e concatena (junta) essas informações e envia para a base de dados, que por sua vez vai retornar como verdadeiro e executar o que foi pedido.

A figura 9 mostra como está a codificação que permite esse tipo de vulnerabilidade ser executada.

Figura 9 – VULNERABILIDADE NA CODIFICAÇÃO PERMITINDO SQL INJECTION.

```
const db = require("../config/db")

module.exports = class Cliente{
  constructor(){
    this.id = 0
    this.nome = ""
    this.login = ""
    this.senha = ""
  }

  static async login(login, senha){
    let usuarios = await db.exec("select id, nome, login, senha from clientes where login='" + login +  "' and senha='" + senha + "' ")
    return usuarios[0];
  }

  static async todos(){
    return await db.exec("select id, nome, login, senha from clientes order by id desc limit 100")
  }

  static async busca(id){
    return await db.exec("select id, nome, login, senha from clientes where id = " + id)
  }

  static async apagar(id){
    return await db.exec("delete from clientes where id = " + id)
  }

  async salvar(){
    try {
      await db.exec(`insert into clientes(nome, login, senha) values('${this.nome}', '${this.login}', '${this.senha}')`)
    }
    catch(e){
      if(e.message.indexOf("Duplicate entry") !== -1){
        throw { message: "Login já existe, crie com um login diferente de: " + this.login }
      }
      throw e
    }
  }
```

Fonte: Autoria Propria (2022)

Esse tipo de codificação que foi mostrado na figura acima é um dos erros mais comuns em um sistema que não está preparado para verificação de sql injection, porque permite a concatenação dos parâmetros do nome da tabela junto com os valores passados pelo usuário.

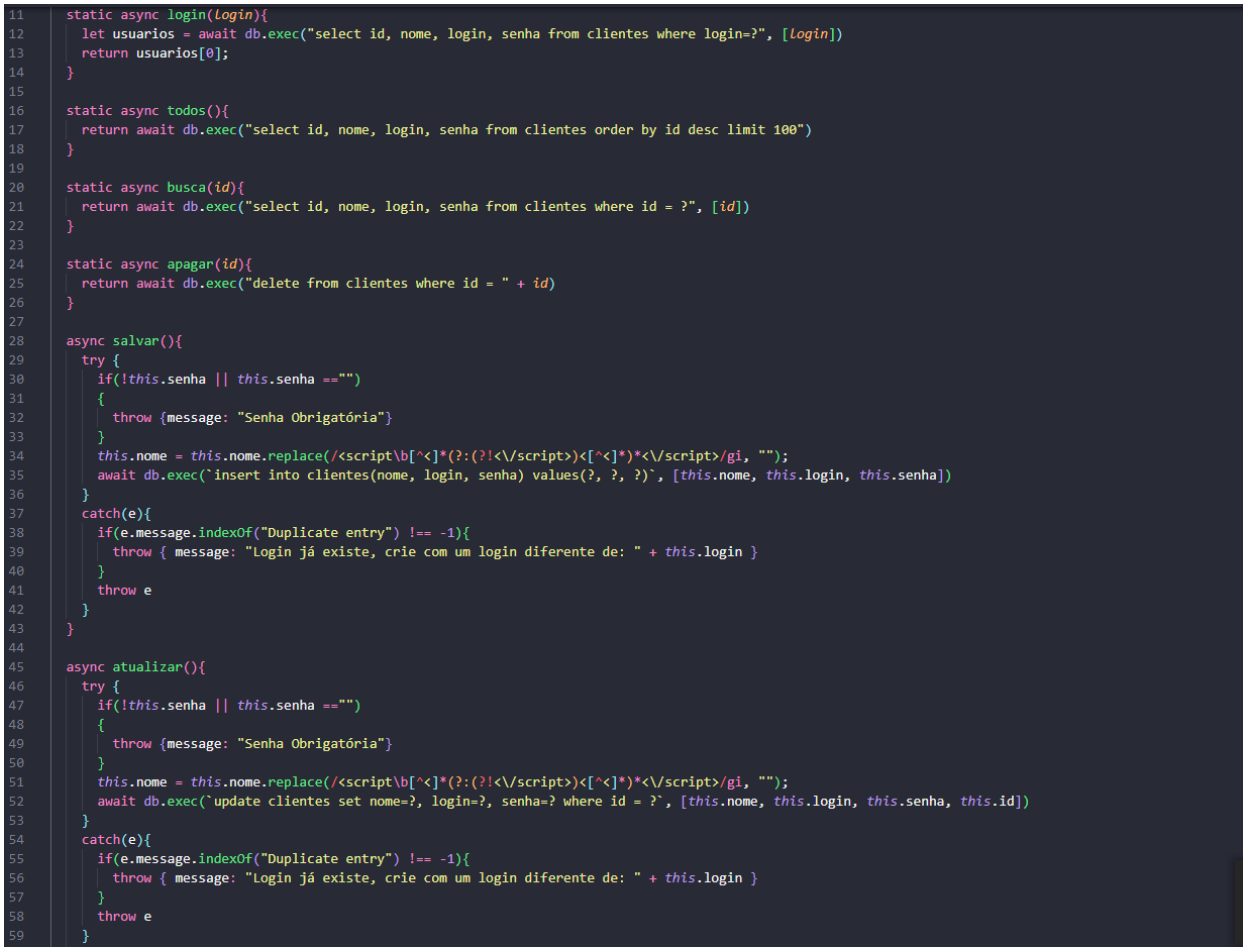
5.6.2. Possivel correção da vulnerabilidade sql injection na aplicação

Uma das formas possíveis para tratar o problema de sql injection na aplicação é alterar a forma que tratamos os parâmetros que serão passados para a query, como será mostrado na figura 10. Dessa forma o driver do banco de dados será responsável por tratar a vulnerabilidade, uma vez que não será possível fazer a concatenação das informações através da query.

Como será possível ver, será acrescentado “?” na frente de cada parâmetro que representa o nome das colunas e ao invés de concatenar os dados recebidos do browser, será passado um objeto array para sistema gerenciador de banco de

dados.

Figura 10 – CORREÇÃO DA VULNERABILIDADE SQL INJECTION.

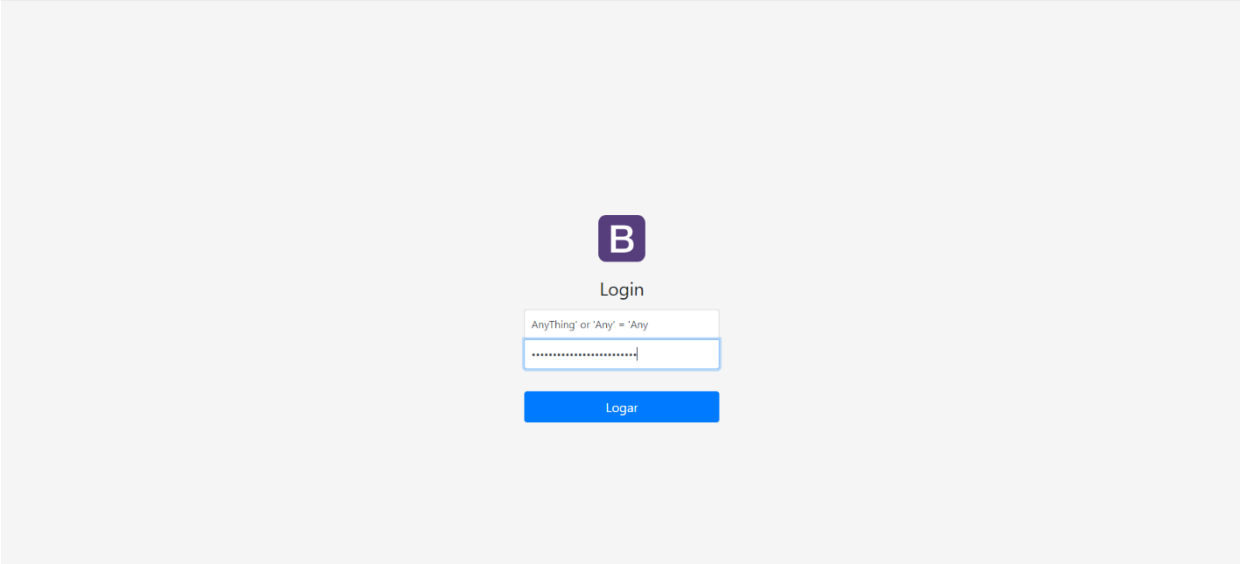


```
static async login(Login){
  let usuarios = await db.exec("select id, nome, login, senha from clientes where login=?", [Login])
  return usuarios[0];
}

static async todos(){
  return await db.exec("select id, nome, login, senha from clientes order by id desc limit 100")
}

static async busca(id){
  return await db.exec("select id, nome, login, senha from clientes where id = ?", [id])
}

static async apagar(id){
  return await db.exec("delete from clientes where id = " + id)
}

async salvar(){
  try {
    if(!this.senha || this.senha =="")
    {
      throw {message: "Senha Obrigatória"}
    }
    this.nome = this.nome.replace(/<script\b[^<]*(?:(?!<\/script>)<[^<]*)*<\/script>/gi, "");
    await db.exec(`insert into clientes(nome, login, senha) values(?, ?, ?)`, [this.nome, this.login, this.senha])
  }
  catch(e){
    if(e.message.indexOf("Duplicate entry") !== -1){
      throw { message: "Login já existe, crie com um login diferente de: " + this.login }
    }
    throw e
  }
}

async atualizar(){
  try {
    if(!this.senha || this.senha =="")
    {
      throw {message: "Senha Obrigatória"}
    }
    this.nome = this.nome.replace(/<script\b[^<]*(?:(?!<\/script>)<[^<]*)*<\/script>/gi, "");
    await db.exec(`update clientes set nome=?, login=?, senha=? where id = ?`, [this.nome, this.login, this.senha, this.id])
  }
  catch(e){
    if(e.message.indexOf("Duplicate entry") !== -1){
      throw { message: "Login já existe, crie com um login diferente de: " + this.login }
    }
    throw e
  }
```

Fonte: Autoria Propria (2022)

Após fazer as alterações mostrada acima, pode-se observar que no sistema não é possível mais realizar login com a injeção de comandos SQL, como mostrado na figura 11.

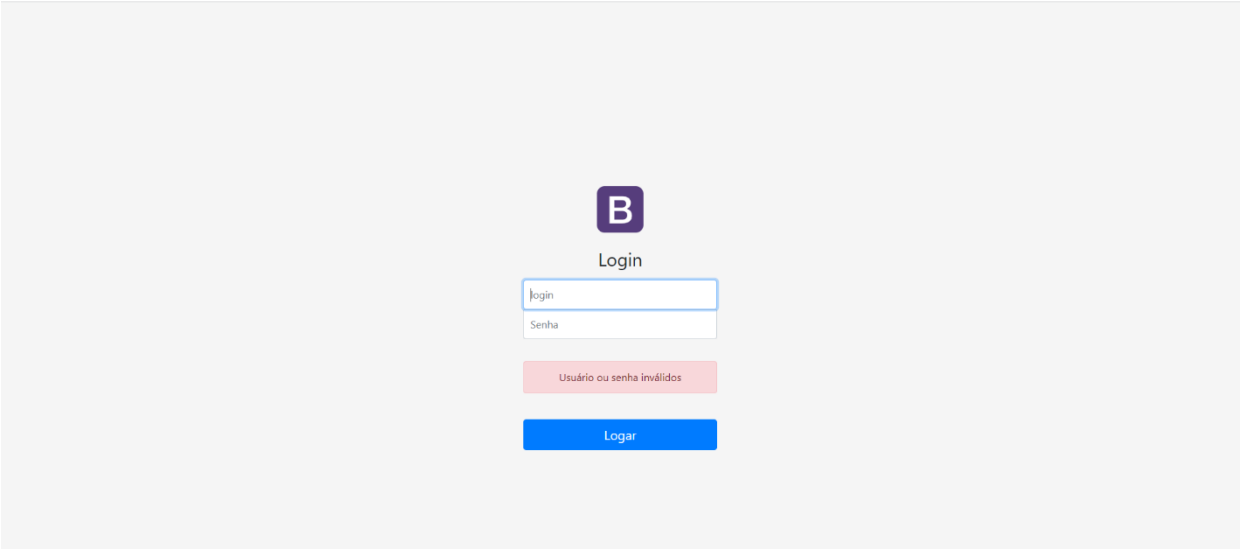
Figura 11 - TENTATIVA DE LOGIN COM SQL INJECTION APÓS CORREÇÃO.



Fonte: Autoria Propria (2022)

Neste processo foram inseridos os mesmos parâmetros dos quais o atacante passou na tela de login, onde conseguiu logar na aplicação mesmo sem possuir as credenciais para ter acesso ela, no entanto, como será mostrado o resultado na figura 12, a aplicação não aceitará mais esse tipo de injeção e retornará um erro de autenticação na tela para o usuário.

Figura 12 – FALHA NA TENTATIVA DE INSERIR SQL INJECTION.



Fonte: Autoria Propria (2022)

Como pode se observar o problema de injeção de SQL foi resolvido com uma simples alteração na estrutura de codificação, no entanto podemos visualizar que o

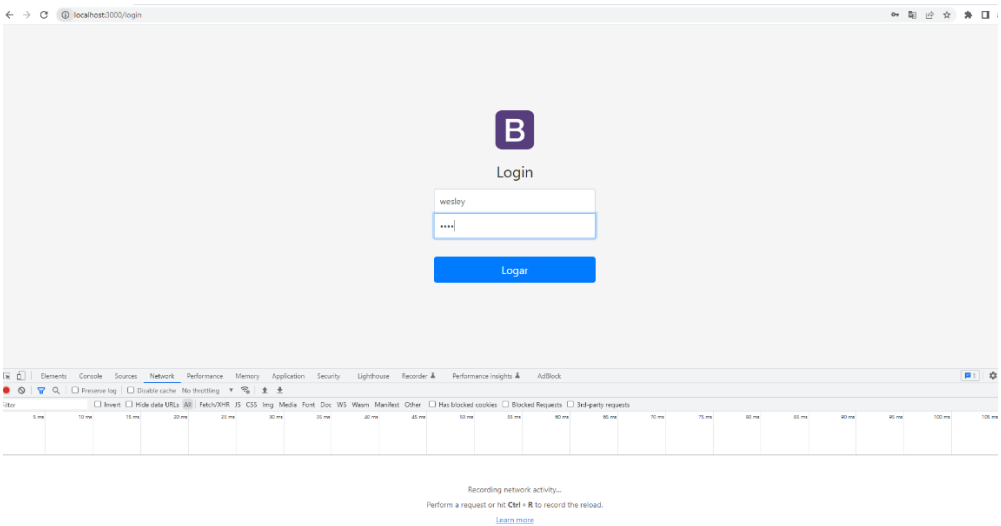
sistema possui outra vulnerabilidade, nós podemos verificar dados trafegando exposto na rede.

5.6.3.Aplicação com vulnerabilidade de exposição de dados

Como foi observado na figura 12 a aplicação tem a vulnerabilidade de exposição de dados, isto acontece porque ela está enviando os parâmetros de login e senha para o servidor através do método GET do protocolo HTTP, ou seja, o método get envia os parâmetros através da url do browser, tornando os dados visíveis e de fácil captura por parte do atacante, que pode simplesmente visualizar a url do usuário, e/ou acessar a opção inspecionar elemento do navegador e encontrar as informações desejada, os dados também pode ser capturado se algum hacker estiver com um sniffer monitorando o tráfego na rede, ele poderá capturar o pacote com os dados e fazer a leitura dele, tudo isso porque a informação está trafegando no cabeçalho do HTTP e também está em texto claro, ou seja, sem criptografia.

A figura 13 mostrará o usuário inserindo suas credenciais de acesso para efetuar login no sistema.

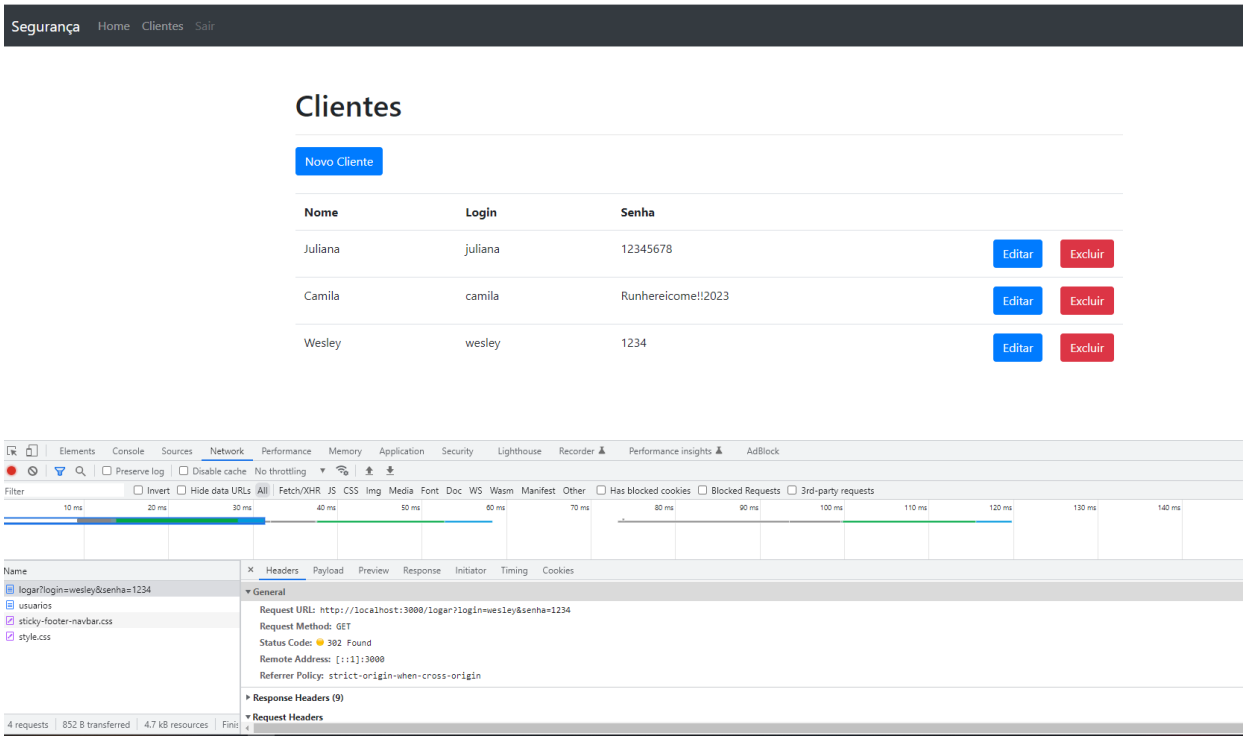
Figura 13 – CREDENCIAIS DO USUARIO DO SISTEMA.



Fonte: Autoria Propria (2022)

Após o usuário informar suas credenciais de acesso, o sistema permitirá entrar na área logada, porque as credenciais são válidas e legitimas, e na figura 14 pode-se observar que as informações de login estão expostas na url do browser.

Figura 14 – DADOS EXPOSTOS NA URL.



Fonte: Autoria Propria (2022)

Para entender melhor a causa da vulnerabilidade as figuras 15, 16 e 17 mostrará o código fonte do sistema que está causando essa vulnerabilidade.

Figura 15 – METODO GET CONFIGURADO NA VIEW.

```
<body class="text-center">
  <form class="form-signin" action="/logar" method="GET">
    <img class="mb-4" src="https://getbootstrap.com/docs/4.0/assets/brand/bootstrap-solid.svg" alt="" width="72" height="72">
    <h1 class="h3 mb-3 font-weight-normal">Login</h1>
    <label for="login" class="sr-only">Login</label>
    <input type="login" id="login" name="login" class="form-control" placeholder="login" required autofocus>
    <label for="senha" class="sr-only">Senha</label>
    <input type="password" id="senha" name="senha" class="form-control" placeholder="Senha" required>
    <% if(erros != ""){ %>
      <br>
      <div class="alert alert-danger" role="alert">
        <%= erros %>
      </div>
    <% } %>
    <br>
    <button class="btn btn-lg btn-primary btn-block" type="submit">Logar</button>
  </form>
</body>
</html>
```

Fonte: Autoria Propria (2022)

Como explicado anteriormente no capítulo 3, a aplicação foi desenvolvida na arquitetura MVC, o código exibido acima na figura 15, é a view onde, existe a iteração entre o usuário e o sistema, como se pode observar o método configurado para enviar os dados para o servidor é o GET e por isso os dados trafegam na url.

Figura 16 – METODO GET CONFIGURADO NA ROUTE LOGAR.

```
router.get('/login', LoginController.index);
router.get('/logar', LoginController.logar);
router.get('/sair', LoginController.deslogar);
```

Fonte: Autoria Propria (2022)

Da mesma forma pode-se ver na figura 16 que a rota que redireciona para o controlador responsável pela requisição HTTP também está configurada para receber um GET.

Figura 17– METODO GET CONFIGURADO NO CONTROLLER LOGAR.

dex.ejs | routes.js | usuario.js | login_controller.js

controllers > login_controller.js > ...

```
const Usuario = require("../models/usuario")
const Cookie = require("../helpers/cookie")

module.exports = {
  index: async (req, res) => {
    const erros = await req.consumeFlash('erro');
    res.render('login/index', { erros: erros });
  },
  deslogar: async (req, res) => {
    Cookie.remove(res, "usuario")
    res.redirect("/")
  },
  logar: async (req, res) => {
    let usuario = await Usuario.login(req.query.login, req.query.senha);
    if(usuario){
      Cookie.set(res, "usuario", usuario)
      res.redirect("/usuarios")
    }
    else{
      await req.flash('erro', 'Usuário ou senha inválidos');
      res.redirect("/login")
    }
  }
}
```

Fonte: Autoria Propria (2022)

Por último pode-se ver que na função logar do controlador está preparado para receber das requisições do tipo query, ou seja que trafegada no url das

mensagens, essas formas de desenvolvimento de aplicações são perigosas porque como observamos os dados estão expostos e podem ser visto por qualquer usuário e pode piorar se não houver criptografia no trafego das informações entre o navegador e o servidor, permitindo as atacantes colocarem sniffer de redes e interceptarem a mensagem, podendo ler todo o conteúdo transmitido, pois o mesmo encontra-se em texto claro.

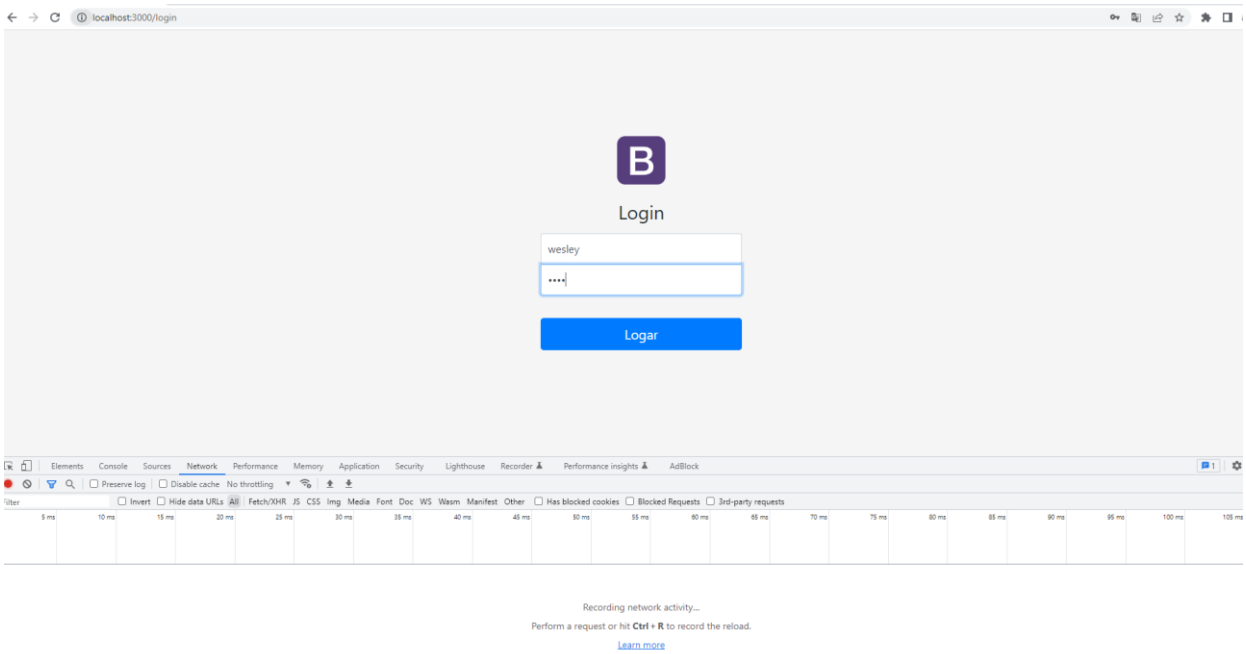
A seguir será mostrado possível correção para esta vulnerabilidade.

5.6.4. Possivel correção da vulnerabilidade dados expostos na aplicação

Para resolver este tipo de vulnerabilidade uma possível correção adotada é trocar a forma de transporte de dados através do protocolo HTTP, ao invés de usar o método GET para enviar requisições através da url do navegar, enviando as informações pelo corpo do HTTP, também conhecido como body, através do método POST.

Na figura 18 será mostrado a execução do login pelo usuário novamente e nas figuras 19, 20, 21 e 22 mostrará as correções impedindo de existir a vulnerabilidade de exposição de dados.

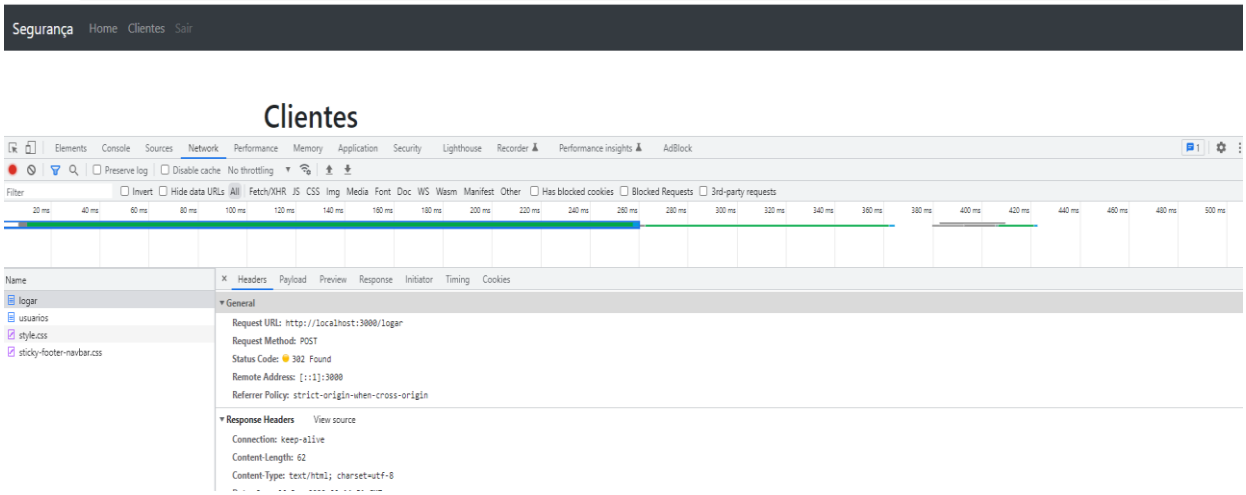
Figura 18 – LOGIN DO USUÁRIO.



Fonte: Autoria Propria (2022)

Na figura 18 é possível ver que os dados não estão sendo enviados pela url da aplicação e tampouco no cabeçalho (header) do protocolo HTTP, dessa forma os dados serão trafegados na rede de forma segura para o servidor de aplicação, uma vez que é utilizado o protocolo HTTPS, onde os dados são trafegados criptografados pela rede.

Figura 19 – APLICAÇÃO UTILIZANDO MÉTODO POST.



Fonte: Autoria Propria (2022)

Na figura 20 foi feita a modificação no view para utilizar o método POST para enviar os dados para servidor, fazendo com que os dados sejam trafegados pelo corpo da aplicação e não pela url do browser.

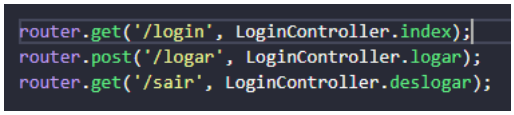
Figura 20 – MODIFICAÇÃO NO ENVIO DOS DADOS PARA MÉTODO POST.

```
<body class="text-center">
  <form class="form-signin" action="/logar" method="POST">
    <img class="mb-4" src="https://getbootstrap.com/docs/4.0/assets/brand/bootstrap-solid.svg" alt="" width="72" height="72">
    <h1 class="h3 mb-3 font-weight-normal">Login</h1>
    <label for="login" class="sr-only">Login</label>
    <input type="login" id="login" name="login" class="form-control" placeholder="login" required autofocus>
    <label for="senha" class="sr-only">Senha</label>
    <input type="password" id="senha" name="senha" class="form-control" placeholder="Senha" required>
    <% if(erros != ""){ %>
      <br>
      <div class="alert alert-danger" role="alert">
        <%= erros %>
      </div>
    <% } %>
    <br>
    <button class="btn btn-lg btn-primary btn-block" type="submit">Logar</button>
  </form>
</body>
</html>
```

Fonte: Autoria Propria (2022)

Também foi realizado mudança de GET para POST na route da função logar que envia os dados para o controlador, conforme mostrado pela figura 21.

Figura 21 – MODIFICAÇÃO DE GET PARA POST NO ROUTE DA APLICAÇÃO.



Fonte: Autoria Propria (2022)

Por fim para finalizar está parte, também foi alterado a forma que o controlador recebe requisições feita pelo browser na função logar, ao invés de receber pela query, foi alterado para receber as requisições através do body, conforme mostrada pela figura 21.

Figura 22 - MODIFICAÇÃO NA REQUISIÇÃO DO CONTROLLER.

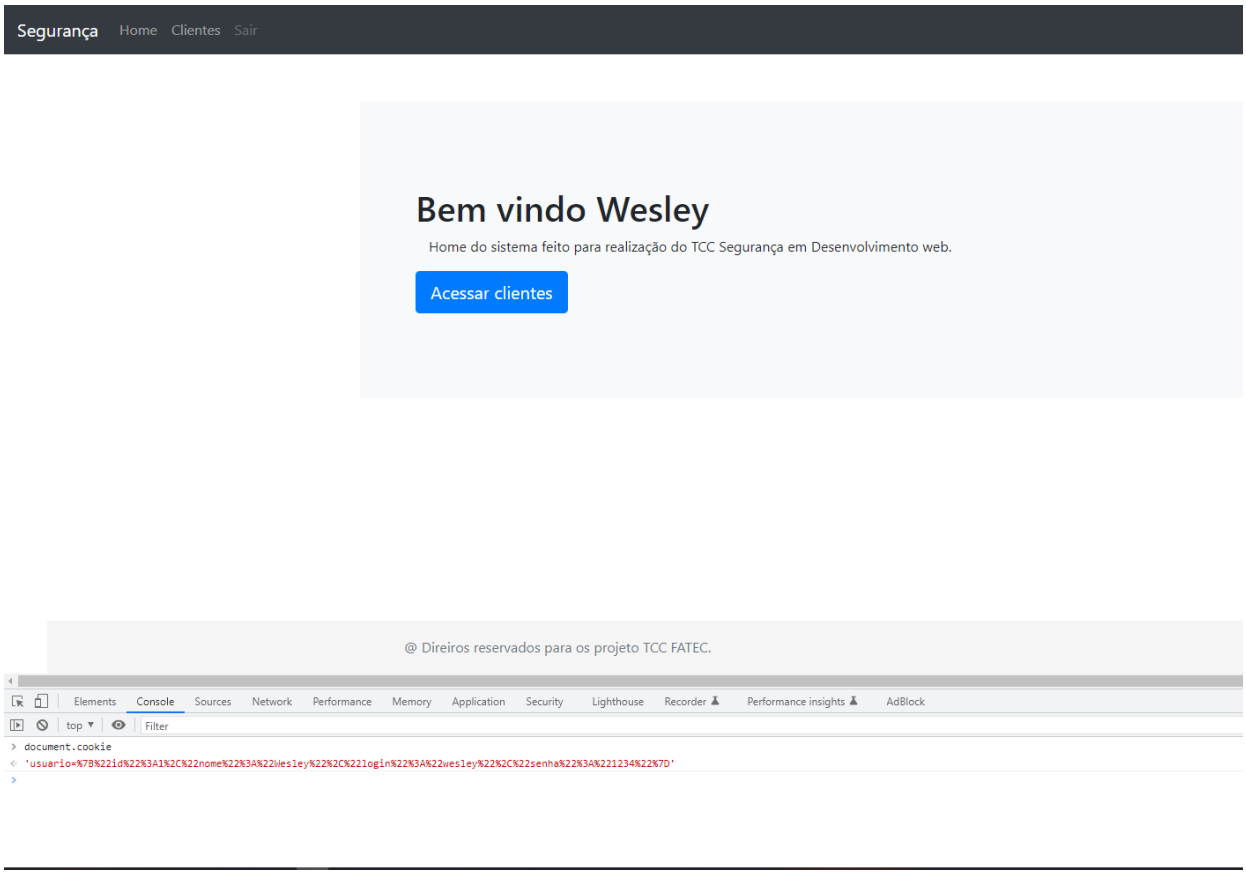
Fonte: Autoria Propria (2022)

5.6.5. Aplicação com vulnerabilidade de roubo de sessão

Roubo de sessão é uma vulnerabilidade relacionado aos cookies do sistema, essa vulnerabilidade permite o atacante ler e/ou alterar informações do sistema, por exemplo o usuário pode logar com um usuário que possui pouco privilégio de acesso ao sistemas e após conseguir acesso ele altera o cookie para um usuário com muito privilégio na aplicação e começa a realizar diversos tipos de ataque ao sistema, esse tipo de vulnerabilidade acontece por dois motivos, o primeiro devido a uma falha no momento de escrever o código e as configurações do cookie ficam na

responsabilidade do navegador e não do lado servidor, como será mostrado nas figuras 23, 24, 25 e 26, o outro motivo é não ter criptografia.

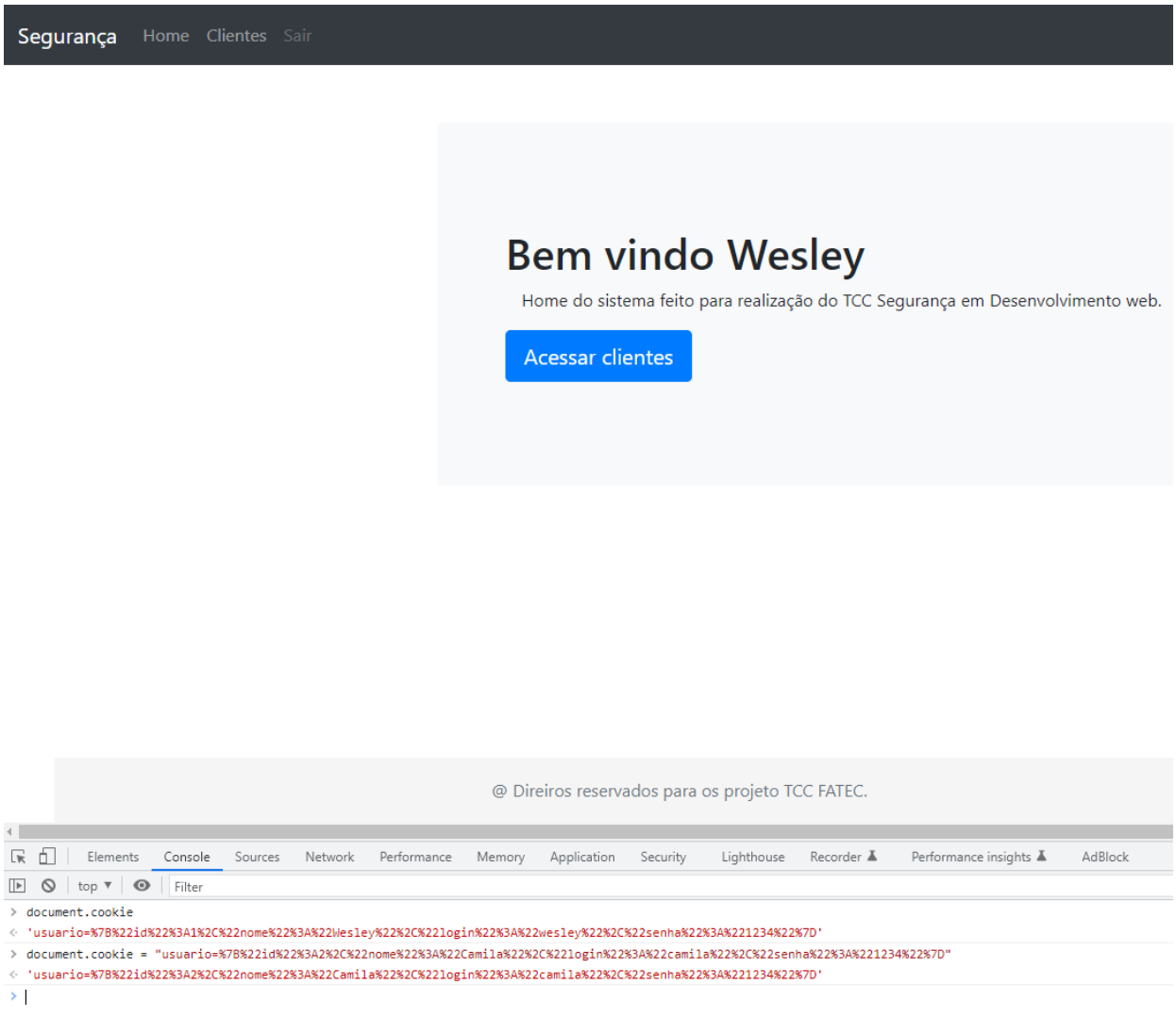
Figura 23 – VISUALIZANDO COOKIE DO USUÁRIO.



Fonte: Autoria Propria (2022)

Como pode ver na figura 23 o cookie com as informações do usuário está visível e pode ser editado pela ferramenta de inspeção fornecida pelo navegador, qualquer usuário que tenha conhecimento em Javascript pode fazer alteração no cookie através da ferramenta console, para isso acontecer é necessário alterar os parâmetros e executar o comando em Javascript como será mostrado na figura 24.

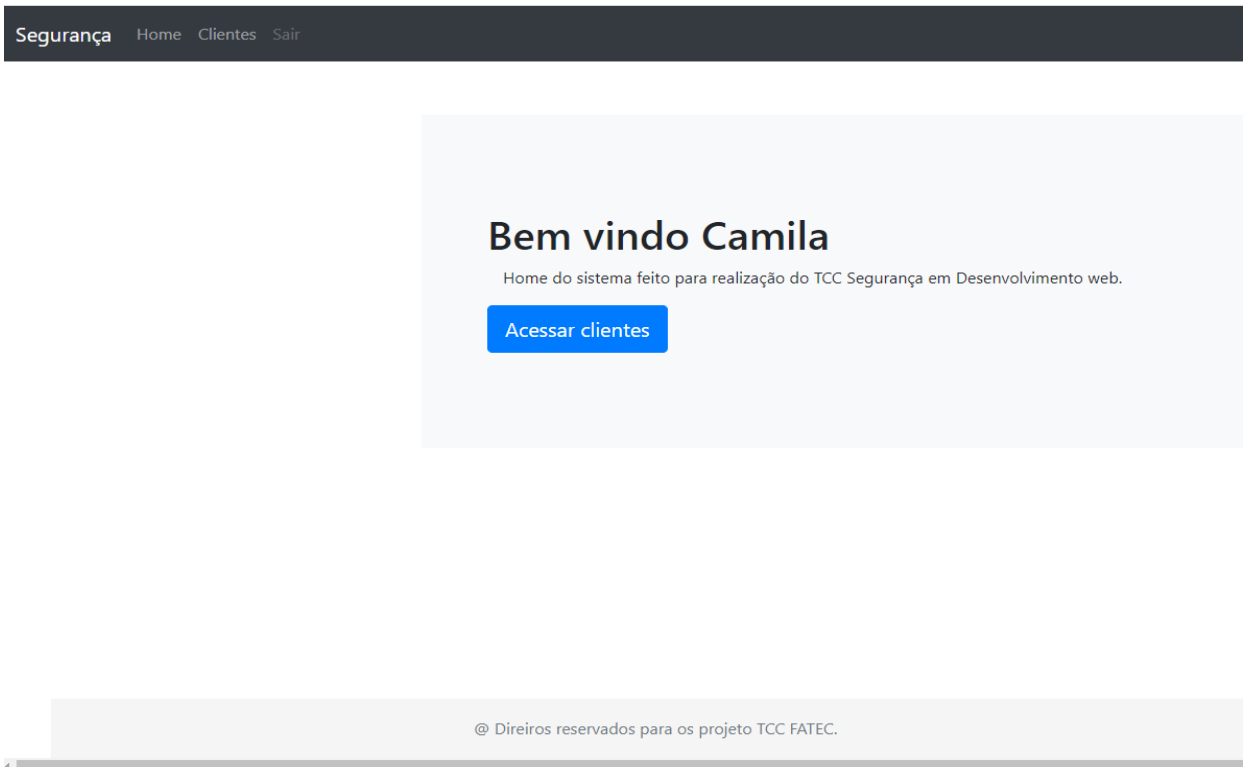
Figura 24 – ALTERANDO PARAMETROS DO COOKIE DO USUÁRIO.



Fonte: Autoria Propria (2022)

Após a alteração nos parâmetros do usuário, o atacante conseguiu obter acesso ao sistema através das credenciais de outro usuário, o resultado desse procedimento será visto na figura 25.

Figura 25 – RESULTADO DO ROUBO DE SESSÃO DO USUÁRIO.



Fonte: Autoria Propria (2022)

Como pode ser visto na figura 25, o ataque foi bem-sucedido e agora o sistema está logado com outro usuário. Um dos motivos deste problema estar acontecendo é devido a falha no momento de desenvolvimento, porque foi adota uma codificação que permite que cookies sejam alterados do lado do cliente através da configuração httpOnly: false, que deveria ser utilizado como httpOnly: true, como será mostrado na figura 26.

Figura 26 – CODIFICAÇÃO RESPONSÁVEL PELO ROUBO DE SESSÃO DO USUÁRIO.

```
get: (request, key) => {
  let cookieParse = {},
    rc = request.headers.cookie;

  rc && rc.split(';').forEach(function( cookie ) {
    let parts = cookie.split('=');
    cookieParse[parts.shift().trim()] = decodeURI(parts.join('='));
  });

  return cookieParse[key];
},
set: (response, key, value) => {
  response.cookie(key, JSON.stringify(value), { maxAge: (1 * 60 * 60 * 1000), httpOnly: false }); // 1 hora
},
```

Fonte: Autoria Propria (2022)

Esse problema de vulnerabilidade pode ser sanado alterando o tratamento de cookie para o servidor e implementando criptografia nele, fazendo com que o atacante não consiga decifrar os parâmetros da mensagem como será mostrado no tópico seguinte.

5.6.6. Possivel correção da vulnerabilidade de roubo de sessão

Para resolver a vulnerabilidade de sessão será alterado a forma de tratamento dos cookies, as modificações nos cookies serão permitidas apenas no lado do servidor e não do lado cliente, desta forma o cookie não poderá ser escrito no cliente por códigos de Javascript.

A correção do código pode ser vista na figura 27.

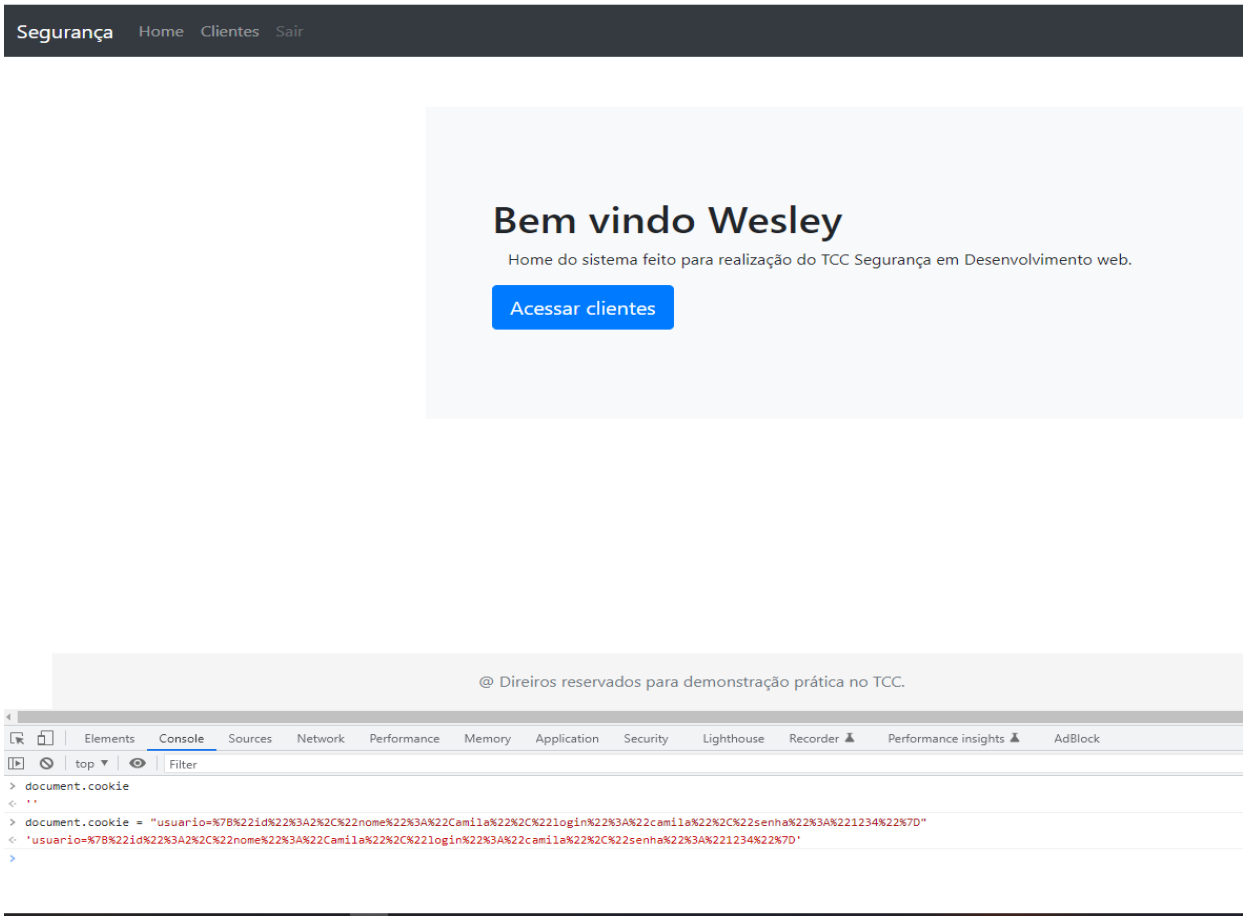
Figura 27 – VISUALIZANDO COOKIE DO USUÁRIO.

```
module.exports = {
  get: (request, key) => {
    let cookieParse = {},
      rc = request.headers.cookie;

    rc && rc.split(';').forEach(function( cookie ) {
      let parts = cookie.split('=');
      cookieParse[parts.shift().trim()] = decodeURI(parts.join('='));
    });

    return cookieParse[key];
  },
  set: (response, key, value) => {
    response.cookie(key, value, { maxAge: (1 * 60 * 60 * 1000), httpOnly: true }); // 1 hora
  },
  remove: (response, key) => {
    response.cookie(key, "", { maxAge: -1, httpOnly: true });
  }
};
```

Fonte: Autoria Propria (2022)

Após está correção será possível que não será possível modificar os cookies via Javascript pelo browser, como mostrado na figura 27.

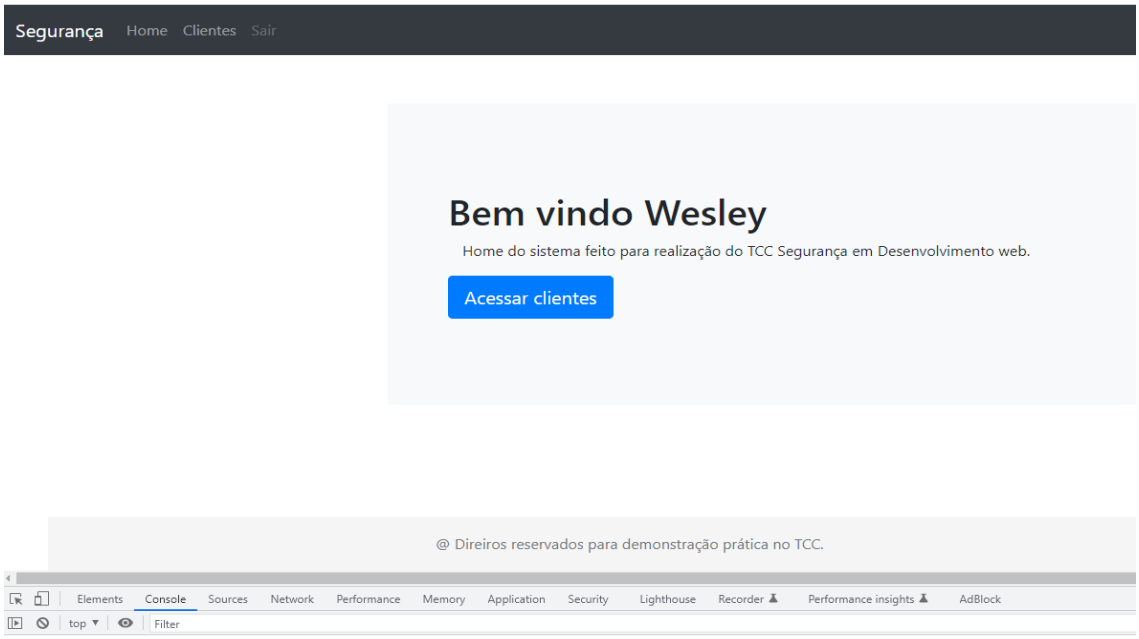
Figura 28 – TENTATIVA DE ALTERAÇÃO DE COOKIE NO BROWSER.



Fonte: Autoria Propria (2022)

Como pode ser visto na figura 28, foi realizada a tentativa de alterar os parâmetros de login dos usuários nos cookies por Javascript.

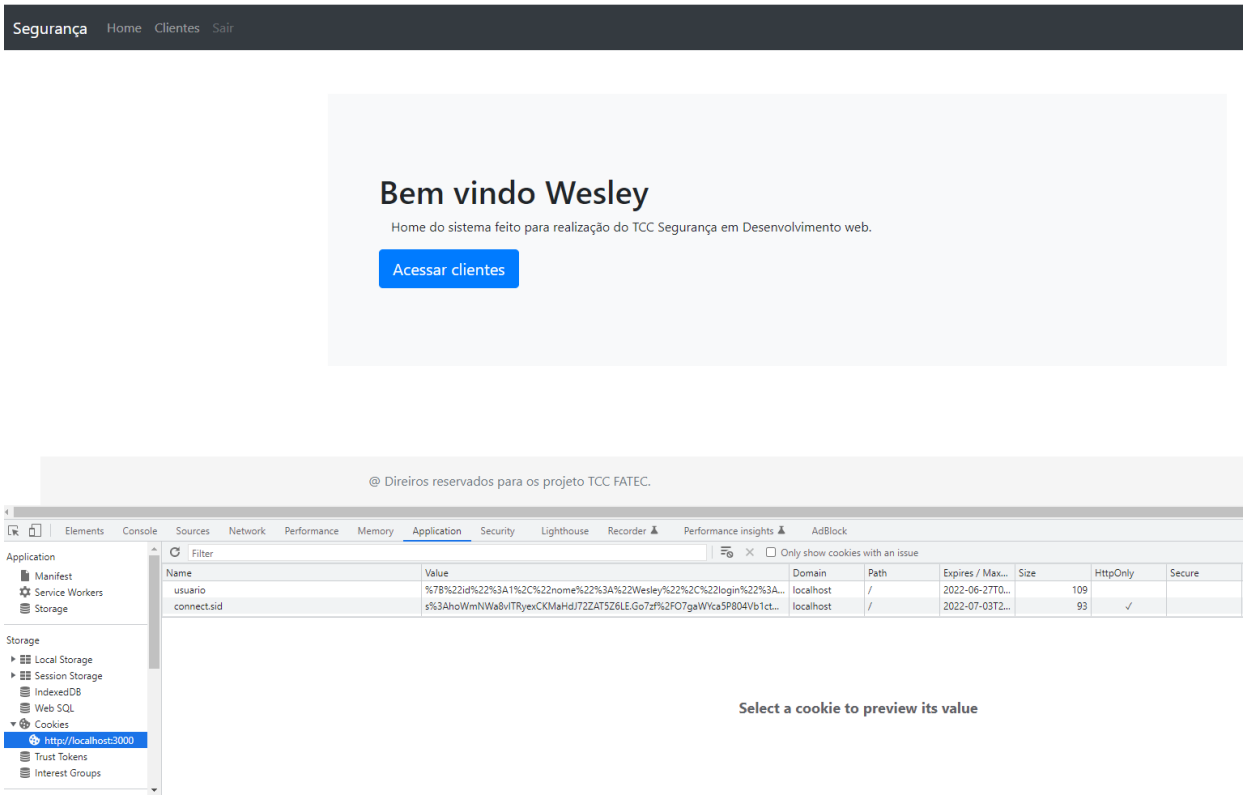
Figura 29 – FALHA NA TENTATIVA DE ALTERAÇÃO DE COOKIE NO BROWSER.



Fonte: Autoria Propria (2022)

Depois da alteração realizada no código da aplicação não é possível realizar roubo de sessão via Javascript no browser da aplicação, no entanto ainda é possível atacar essa vulnerabilidade de outra maneira, como pode ser vista na figura 30.

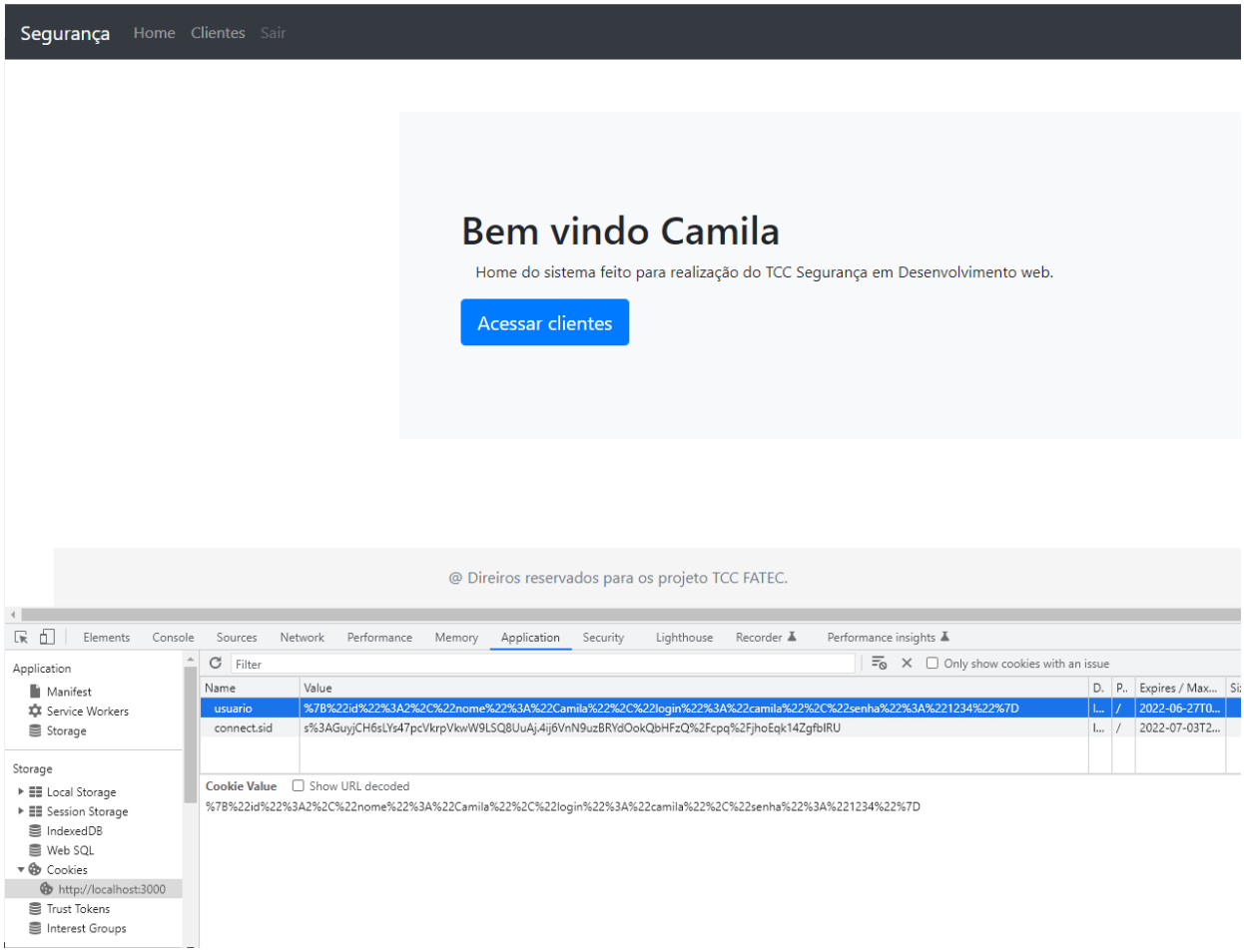
Figura 30 – ALTERAÇÃO DE COOKIE NO BROWSER.



Fonte: Autoria Propria (2022)

Para executar o roubo de sessão com alteração do cookie o atacante vai utilizar a ferramenta inspecionar fornecido pelo browser e irá até a aba aplicação, chegando até o link de cookies e encontrará o cookie do qual ele deseja alterar, no caso vai ser alterado o cookie com as credenciais do usuário e como pode ser visualizado na figura 31, o ataque foi bem-sucedido.

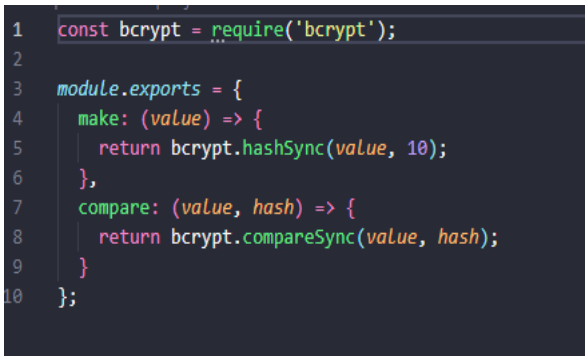
Figura 31 – ATAQUE BEM-SUCEDIDO.



Fonte: Autoria Propria (2022)

Como pode ser observado, o ataque foi bem-sucedido e mais uma vez o sistema foi comprometido, onde o usuário mal-intencionado conseguiu alterar os valores do cookie do usuário, conseguindo navegar além de suas permissões. Para resolver este problema será necessário utilizar criptografia para proteger as informações contidas nos cookies, e para isso será necessário instalar uma biblioteca que faz esse gerenciamento para sistema. É sempre importante lembrar que se deve utilizar ferramentas confiáveis validadas e testadas por especialistas e não criar algo onde se tem conhecimento básico sobre o assunto, porque fazendo desta maneira pode conter vulnerabilidades que quando atacada e acessada por atacantes, causará danos severos tanto para os usuários quanto para a empresa.

Figura 32 - CRIPTOGRAFIA NA GERAÇÃO DE COOKIES



```
const bcrypt = require('bcrypt');

module.exports = {
  make: (value) => {
    return bcrypt.hashSync(value, 10);
  },
  compare: (value, hash) => {
    return bcrypt.compareSync(value, hash);
  }
};
```

Fonte: Autoria Propria (2022)

Como pode ser visualizado na figura 32, para resolver o problema de roubo de sessão foi implementado uma função para criptografar os dados no cookie, essa função tem responsabilidade de gerar as chaves criptográficas e toda vez que receber uma nova informação fazer a comparação para ver se as chaves são iguais.

Também foi implementado criptografia para comparar os valores do cookie e do validador implementado no sistema, se a comparação entre ambos for diferente a aplicação retornará para a página de login e obrigará o usuário a logar novamente. As alterações poder ser visualizada nas figuras 33 e 34;

Figura 33 – COMPARAÇÃO ENTRE O COOKIE DO USUARIO E O HASH DA CRIPTOGRAFIA**.**

```
const Cookie = require("../helpers/cookie")
const Criptografia = require('../helpers/cripto')

module.exports = (req, res, next) => {
  let user_validator = Cookie.get(req, "usuario_validador");
  let user = Cookie.get(req, "usuario");

  if(!user_validator || !user ) return res.redirect("/login");

  user_validator = unescape(user_validator);
  user= unescape(user);

  let valid = Criptografia.compare(user, user_validator);


  if(!valid) return  res.redirect("/login");

  try{
    user = JSON.parse(user);
  }
  catch(e){
    return res.redirect("/login");
  }

  req.usuarioLogado = user;
  next();
}
```

Fonte: Autoria Propria (2022)

Como pode ser visualizado foi adicionado um comparador para verificar se os dados criptografados são iguais as informações contidas no cookie.

Figura 34 – CRIPTOGRAFANDO COOKIE NA CRIAÇÃO DA SESSÃO.

```
const Usuario = require("../models/usuario")
const Cookie = require("../helpers/cookie")
const Criptografia = require('../helpers/cripto')

module.exports = {
  index: async (req, res) => {
    const erros = await req.consumeFlash('erro');
    res.render('login/index', { erros: erros });
  },
  deslogar: async (req, res) => {
    Cookie.remove(res, "usuario")
    res.redirect("/")
  },
  logar: async (req, res) => {
    let user = await Usuario.login(req.body.login, req.body.senha);
    if(user){
      let string_user = JSON.stringify(user);
      let validator = Criptografia.make(string_user);
      Cookie.set(res, "usuario", string_user)
      Cookie.set(res, "usuario_validador", validator)
      res.redirect("/usuarios")
    }
    else{
      await req.flash('erro', 'Usuário ou senha inválidos');
      res.redirect("/login")
    }
  }
}
```

Fonte: Autoria Propria (2022)

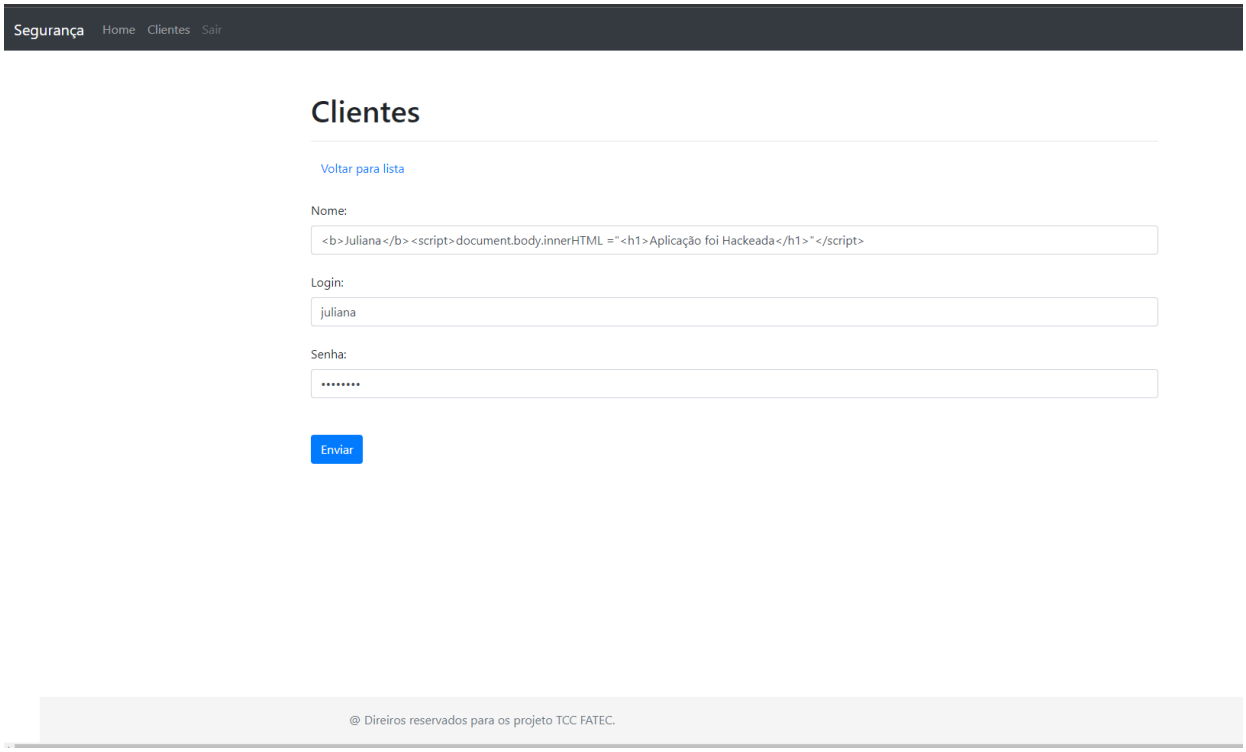
Da mesma forma foi adicionado no controlador uma variável para criptografar o cookie com finalidade de proteger a aplicação contra roubo de sessão de usuários.

5.6.7 Aplicação com vulnerabilidade de cross-site scripting

As vulnerabilidades de cross-site scripting, ou também conhecido como XSS, permite que usuários mal intencionados injetem códigos maliciosos dentro da aplicação, este tipos de script podem alterar o conteúdo que será entregue para o usuário final, esse tipo de ataque pode roubar dados dos clientes, acessar tokens de validação, roubar sessão, quebrar a aplicação pois altera todo o funcionamento dela, este ataque utiliza linguagens de programação do lado do cliente e a mais utilizada é o Javascript, na figura 35 será mostrado como um atacante injeta esse

tipo de conteúdo na aplicação.

Figura 35 – TENTATIVA ATAQUE DE CROSS-SITE SCRIPTING.



Fonte: Autoria Propria (2022)

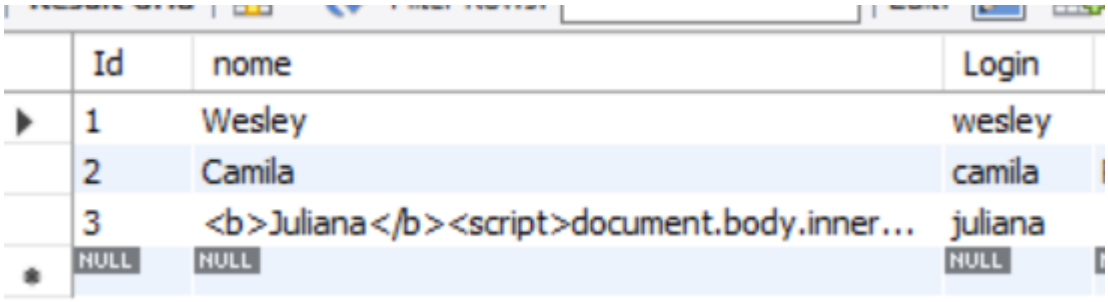
Após a inserção de script, a aplicação armazenou os dados na base de dados e quando redirecionada para página de usuários é possível ver na figura 36 que a aplicação quebrou e não é possível acessá-la mais, se este tipo de ataque ocorresse em uma aplicação de grande porte, como por exemplo um e-commerce, a empresa não poderia efetuar suas vendas e perderia muito dinheiro.

Figura 36 – ATAQUE DE CROSS-SITE SCRIPTING.

Fonte: Autoria Propria (2022)

Figura 37 mostra esses dados registrados na base de dados e a aplicação só será possível após a correção deles.

Figura 37 – BASE DE DADOS COM SCRIPT DE CROSS-SITE SCRIPTING.



Id	nome	Login
1	Wesley	wesley
2	Camila	camila
3	Juliana<script>document.body.inner...	juliana
NULL	NULL	NULL

Fonte: Autoria Propria (2022)

Para corrigir esse a vulnerabilidade de cross-site scripting, foi inserida uma regular expressão (regex) na função de salvar as informações na base de dados, essa regex tem responsabilidade de verificar se os valores recebidos da requisição do navegador contêm scripts e caso tenha ela os tirará e manterá apenas as informações necessárias. A correção pode ser visualizada na figura 38.

Figura 38 – CORREÇÃO DA VULNERABILIDADE DE CROSS-SITE SCRIPTING

```
  async salvar(){
    try {
      if(!this.senha || this.senha =="")
      {
        throw {message: "Senha Obrigatória"}
      }
      this.nome = this.nome.replace(/<script\b[^<]*(?:(?!<\/script>)<[^<]*)*<\/script>/gi, "");
      await db.exec(`insert into clientes(nome, login, senha) values(?, ?, ?)`, [this.nome, this.login, this.senha])
    }
    catch(e){
      if(e.message.indexOf("Duplicate entry") !== -1){
        throw { message: "Login já existe, crie com um login diferente de: " + this.login }
      }
      throw e
    }
  }

  async atualizar(){
    try {
      if(!this.senha || this.senha =="")
      {
        throw {message: "Senha Obrigatória"}
      }
      this.nome = this.nome.replace(/<script\b[^<]*(?:(?!<\/script>)<[^<]*)*<\/script>/gi, "");
      await db.exec(`update clientes set nome=?, login=?, senha=? where id = ?`, [this.nome, this.login, this.senha, this.id])
    }
    catch(e){
      if(e.message.indexOf("Duplicate entry") !== -1){
        throw { message: "Login já existe, crie com um login diferente de: " + this.login }
      }
      throw e
    }

  }
}
```

Fonte: Autoria Propria (2022)

Essas correções de algumas vulnerabilidades mostradas no trabalho, são necessárias para construirmos a base de uma aplicação minimamente voltada para segurança da informação.

6. CONCLUSÃO

Por meio do desenvolvimento deste trabalho e pelos resultados que foram obtidos, pode-se concluir que o processo de desenvolvimento seguro de software é de grande importância para garantir a integridade, disponibilidade e confidencialidade das informações, que são os pilares básicos da segurança da informação.

Esta forma de pensar e desenvolver softwares não é apenas importante para ter segurança, mas também é de grande valia para todas as áreas, porque visa facilitar a manutenção futura do software, pois com o pensamento em melhorar a qualidade do código escrito, separar as responsabilidades do sistema, desenvolver arquitetura limpa também contribui para melhor entendimento para quem dará a manutenção futura tanto para resolver as possíveis vulnerabilidades e/ou bugs existentes quanto para implementação de novas funcionalidades que serão acrescentadas ao longo da vida útil da aplicação.

Outro fator que também ganha com essa prática de desenvolvimento é que os custos gastos por parte da empresa também pode ser diminuído, uma vez que não será necessário refazer todo o projeto devido a falhas graves encontradas no futuro, porque durante a análise de requisito e definição de arquitetura do projeto já é feito definido todo o escopo arquitetural do sistema, também o problema de vazamento de informação sigilosa dos usuários será mais difícil de acontecer, fazendo com que a empresa diminua consideravelmente a possibilidade de pagamento de indenização devido a dados expostos de clientes.

Por fim é importante lembrar que não existe ambiente 100% seguro, mas com o desenvolvimento focado em segurança a probabilidade de incidentes são minimizados, e sempre com foca na manutenção da aplicação, aplicando as atualizações sempre que disponíveis, instruindo os colaboradores e usuários a adotarem práticas seguras será possível conseguir alcançar níveis excelentes em segurança da aplicação.

REFERÊNCIAS BIBLIOGRÁFICAS

CORREIA, Miguel Pupo; SOUSA, Paulo Jorge. **Segurança no Software**. 2. ed. Lisboa: Ed. FCA, 2017.

SÊMOLA, Marcos. **Gestão da segurança da informação:** Uma Visão Executiva. 2 edição. Rio de Janeiro . Ed. Elsevier Brasil, 2014.

TAKE BLIPBLOG: O que é software? Entenda o conceito, como funciona e seus 6 tipos, Disponível em: https://www.take.net/blog/tecnologia/software/#:~:text=Um%20software%20%C3%A9%20um%20servi%C3%A7o,%2C%20televisores%2C%20entre%20outros). Acesso em: 29 de Abril de 2022 18:30.

BLOG BETRYBE: Aplicações web: entanda o que são e como funcionam! Disponível em: https://blog.betrybe.com/desenvolvimento-web/aplicacoes-web/ . Acesso em 10 de Junho de 2022 10:15

CYNOTECH: Web Application Architecture, Disponível em: https://cynoteck.com/wp-content/uploads/2021/12/Web-Application-Architecture-diagram-3.png .Acesso em: 03 de Junho de 2022 13:17.

BLOG KASPERSKY: O que é um ataque de cross-site scripting? Definição e explicação, Disponível em: https://www.kaspersky.com.br/resource-center/definitions/what-is-a-cross-site-scripting-attack . Acesso em 01 de Junho de 2022 20:19.

MDN WEB DOCS: Uma Visão geral do HTTP, Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Overview. Acesso em 25 de Maio de 2022 16:10.

ILUSTRADEV: O que é MVC? Explicação simples, Disponível em: https://ilustradev.com.br/o-que-e-mvc-explicacao-simples/. Acesso em: 22 de Maio de 2022 16:23