HÍCARO ROBERTO SILVA OLIVEIRA

SEGURANÇA DA INFORMAÇÃO NO COMÉRCIO ELETRÔNICO BRASILEIRO PÓS IMPLANTAÇÃO DA LEI GERAL DE PROTEÇÃO DE DADOS

LAVRAS - MG

2021

HÍCARO ROBERTO SILVA OLIVEIRA

SEGURANÇA DA INFORMAÇÃO NO COMÉRCIO ELETRÔNICO BRASILEIRO PÓS IMPLANTAÇÃO DA LEI GERAL DE PROTEÇÃO DE DADOS

Monografia de Graduação apresentada à Universidade Federal de Lavras como parte das exigências do curso de Sistemas de Informação, para a obtenção do título de Bacharel.

Prof. Dr. Joaquim Quinteiro Uchôa
Orientador

LAVRAS - MG

2021

HÍCARO ROBERTO SILVA OLIVEIRA

SEGURANÇA DA INFORMAÇÃO NO COMÉRCIO ELETRÔNICO BRASILEIRO PÓS IMPLANTAÇÃO DA LEI GERAL DE PROTEÇÃO DE DADOS

Monografia de Graduação apresentada à Universidade Federal de Lavras como parte das exigências do curso de Sistemas de Informação, para a obtenção do título de Bacharel.

APROVADA em 3 de dezembro de 2021.

Prof. Dr. Joaquim Quinteiro Uchôa	UFLA
Prof. Dr. Paulo Afonso Parreira Junior	UFLA
Profa. Dra. Renata Teles Moreira	UFLA

Prof. Dr. Joaquim Quinteiro Uchôa
Orientador

LAVRAS - MG
2021

AGRADECIMENTOS

Agradeço à minha família pelo apoio incondicional, mesmo nos momentos mais difíceis, agradeço também aos meus amigos, especialmente aos amigos Igor e Witor, que, sem eles, eu jamais teria chegado até aqui. Agradeço aos professores da minha graduação e ao meu orientador Joaquim, por ter me guiado durante o trabalho.

RESUMO

Na última década, a integração ao mundo conectado via *W*eb consolida o aumento dos usuários e dos prestadores de serviços, para atender esta geração conectada. O uso de aplicações *W*eb torna-se, então, algo bastante comum, especialmente na área do comércio, que, devido aos recentes eventos pandêmicos que resultaram nas mudanças de hábitos e no surgimento de uma nova realidade de funcionamento do mundo, se viu obrigado a adaptar-se rapidamente ao comércio eletrônico. O presente trabalho aborda questões de segurança em tais aplicações Web de comércio eletrônico, mais especificamente no cenário Brasileiro, onde a Segurança de Dados tornou-se mais rígida e sensível após a Lei Geral de Proteção de Dados (LGPD) entrar em vigor em agosto de 2021. Sendo assim, seguindo o guia de testes proporcionado pela OWASP, foram realizados testes de segurança em aplicativos web de grande influência no comércio eletrônico brasileiro.

Palavras-chave: Segurança da Informação; Lei Geral de Proteção de Dados (LGPD); Testes de Segurança; OWASP (*Open Web Application Security Project*); Comércio Eletrônico (*e*Commerce)

SUMÁRIO

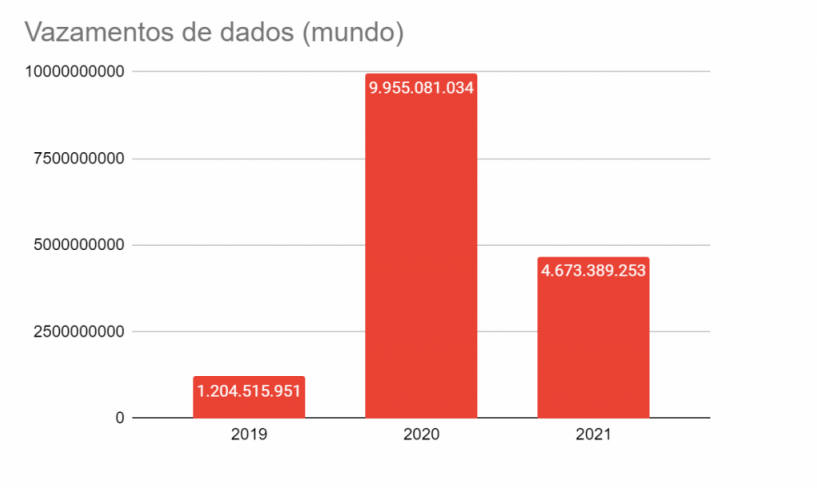
1 INTRODUÇÃO

Com a evolução das aplicações *web* e a abrangência global da *Internet*, muitas organizações passaram a oferecer seus serviços, ou até mesmo a operar completamente dentro dela. Com 5 bilhões de usuários em todo o mundo[1], o fluxo de informação e dados tomou proporções imensas, recebendo bastante atenção de entidades operantes com boas e más intenções.

Os dados dos usuários tornaram-se o 'novo petróleo', e, devido a seu alto potencial lucrativo, estão constantemente na mira de cibercriminosos.[2] Informações valem milhões, e o impacto dos seus vazamentos vão além de prejuízo financeiro bruto à empresas; atentados pessoais, como fraudes, também estão afetando os usuários comprometidos. Bilhões de dólares americanos são roubados todos os anos devido à fraudes e brechas de segurança em aplicações, estima-se que o custo total de ataques cibernéticos à economia mundial é cerca de um trilhão de dólares americanos[3].

O número de vazamento de credenciais aumentou de maneiras alarmantes desde o ano 2019 (Figura 1.1), isso pode ser atribuído, em parte, ao grande aumento da demanda e fluxo de dados, devido à grande quantidade de acessos a serviços *on-line*, e o possível despreparo de empresas para lidar com a segurança e integridade de tais dados, devido à abrupta adaptação digital.

Figura 1.1 – Comparativo no número de credenciais vazadas ano a ano



Fonte: PSafe/ CyberLabs

[1] https://www.worldometers.info
[2] https://www.psafe.com/blog/levantamento-mostra-que-numero-de-vazamentos-de-dados-em-2021-deve-superar-2020-aponta-psafe/
[3] https://parachutetechs.com/2021-cyber-attack-statistics-data-and-trends/

1.1 Motivações

O uso de aplicações *web* no ramo de comércio eletrônico brasileiro tem crescido muito nos últimos anos. Em 2020, o Brasil ocupava a décima quinta posição[4] no mercado eletrônico mundial, com uma receita de 22 bilhões de dólares americanos, e, 27% dessa receita, vem do mercado de Eletrônicos e Mídia.

O Brasil está em quinto lugar na lista dos países com mais usuários de internet no mundo[5], e cerca de 45 por cento da população brasileira comprou ao menos um produto online em 2020[6]. Isso faz com que a questão da integridade de dados do usuário torne-se uma grande prioridade, pricipalmente após entrar em vigor, em agosto de 2021, da Lei Geral de Proteção de Dados (LGPD, 2018), que define regras para o uso, coleta, armazenamento e distribuição de dados dos usuários por empresas públicas e privadas.

A Lei Geral de Proteção de Dados do Brasil foi fundamentada nas diretrizes da lei de proteção de dados vigente na União Europeia, criada em janeiro de 2012 e entrando em vigor em maio de 2018, após o escândalo da exploração de dados pessoais de grandes empresas da *Internet*, onde os dados dos usuários eram utilizados comercialmente em segredo. A legislação Europeia ainda é mais detalhada e completa que a versão brasileira, mas a tendência é que cada vez mais as proteções à dados sensíveis seja reforçada.

Com a LGDP, companhias agora devem garantir que os dados de seus usuários estão devidamente seguros contra acessos, exposição ou alteração indevida dos mesmos, o que requer um investimento significativo com segurança da informação. A lei demanda, também, que todas as empresas sejam completamente transparentes com o uso e armazenamento dos dados para com seus proprietários, que podem, a qualquer momento, solicitar a remoção dos mesmos.

No Brasil, a vigência desta lei estava sendo postergada indefinidamente, e um dos grandes fatores que levou esta lei a entrar em vigor, além da necessidade de se adequar às exigências para celebração de tratados e acordos internacionais, especialmente com a própria União Europeia, foi o grande aumento de invasões e vazamento de dados, além da grande perda financeira de cerca de 80 bilhões de reais em 2019, devido à ataques cibernéticos, de acordo com o levantamento da União Internacional de Telecomunicações (ITU) da Organização das Nações Unidas (ONU).

[4] https://www.ecommercedb.com/en/markets/br/all
[5] https://www.statista.com/statistics/262966/number-of-internet-users-in-selected-countries/)
[6] https://www.ecommercedb.com/en/markets/br/all

O número de ataques fraudulentos contra brasileiros chegou a 1,9 milhão no primeiro semestre de 2021[7], o que corresponde a um aumento de 15,6% com relação ao mesmo período do ano passado. A alta foi puxada principalmente pelas ações contra pessoas de até 25 anos, que tiveram crescimento de 19,3%, de acordo com o Indicador de Tentativas de Fraude da Serasa Experian. A entidade estima que houve uma movimentação fraudulenta a cada oito segundos[8].

Uma aplicação *web* comprometida poderia vir a resultar em um enorme vazamento de dados, o que implicaria, consequentemente, em perdas financeiras severas, além de consequências éticas e legais (LI; XUE, 2014).

1.2 Objetivos e Metodologia

Este trabalho tem por objetivo principal por a teste as funcionalidades de quatro aplicações *web* de comércio eletrônico, a fim de descobrir se a integridade dos dados sensíveis dos usuários das plataformas pode ser comprometida.

As aplicações testadas operam em larga escala no mercado brasileiro e lidam diariamente com um grande contingente de dados e operações financeiras. Das quatro aplicações, três delas são operantes no segmento de mercado B2C (*Business to Customer* - Negócio para Consumidor), que trata-se da venda, por uma loja, de produtos físicos para um cliente final privado, incluindo compras feitas em um computador e dispositivos móveis (Smartphones, Tablets,etc), a última aplicação opera no segmento C2C (*Customer to Costumer* - Consumidor para Consumidor) onde consumidores finais realizam transações entre si, mediados por uma plataforma digital[9].

Para realizar os testes de segurança nas aplicações, foi utilizada a última versão do Guia de Testes de Segurança Web disponibilizado pela OWASP (*Open Web Application Security Project*)[10] .

[7] https://boainformacao.com.br/2021/08/numero-de-ataques-fraudulentos-cresce-mais-de-15-no-primeiro-semestre/

[8] https://boainformacao.com.br/2021/08/numero-de-ataques-fraudulentos-cresce-mais-de-15-no-primeiro-semestre/

[9] https://www.suno.com.br/artigos/c2c/

[10] https://owasp.org/www-project-web-security-testing-guide/stable/

1.3 Estrutura do Texto

O capítulo 2 trata-se da descrição de termos e conceitos utilizados na realização deste trabalho, sendo estes acerca de tecnologias usadas e de vulnerabilidades e explorações de segurança.

O capítulo 3 aborda o conceito e aplicação da Segurança da Informação e fala da sua importância crítica na atualidade.

No capítulo 4, a Metodologia utilizada na realização dos testes é apresentada, juntamente com os testes realizados e ferramentas utilizadas.

Por fim, capítulo 5 apresenta e debate sobre os resultados obtidos em cada teste realizado e a resposta de cada aplicação de comércio eletrônico a eles.

2 REFERENCIAL TEÓRICO

Neste capítulo serão descritos conceitos e termos a serem utilizados no trabalho. Este referencial teórico foi estruturado em dois tópicos, sendo estes: Termos técnicos e Vulnerabilidades de Segurança.

2.1 Termos técnicos

Nesta seção serão apresentados termos necessários para melhor compreensão dos testes realizados. Todos os testes foram feitos utilizando a abordagem Caixa Preta (*Black Box*) onde o testador não possui conhecimento prévio da estrutura do código interno, dos detalhes da implementação ou de caminhos internos do software a ser testado (HOOGENRAAD, 2019).

Para realizar e interpretar corretamente os testes em aplicações *web*, é necessário entender como funciona o *Hypertext Transfer Protocol* - HTTP (Protocolo de Transferência de Hipertexto), que é um protocolo de nível de aplicação que funciona através de Requisição/-Resposta entre um cliente e um servidor e constitui basicamente o padrão de operações através da *web* (FIELDING, 1999); e entender como a linguagem de programação *Hypertext Markup Language* - HTML (Linguagem de Marcação de Hipertexto) cria os documentos eletrônicos (páginas) que são mostrados na *web*. Código HTML garante a formatação padrão de texto e imagens processados por um Navegador *Web*.

A utilização e funcionalidades do HTTP e HTML englobam operações realizadas no Lado-Cliente, também chamado de *front-end*, que refere-se a tudo em uma aplicação *web* que é mostrado ou que ocorre no dispositivo do usuário final. Isso inclui tudo que ele vê graficamente e todas as ações que a aplicação pode performar, dispostas na interface de usuário (UI)[1].

No Lado-Servidor, também chamado *back-end*, ocorrem operações que, muitas vezes, requerem acessos e informações que não podem ser acessadas via cliente, englobando operações com banco de dados e validação de entradas para verificar se os dados estão no formato correto[2]. Para operar sistemas de gerenciamento de banco de dados relacionais é utilizada, por padrão determinado pelo Instituto Nacional de Normas Americano (ANSI) e a Organização Internacional de Normas (ISO), a *Structured Query Language* - SQL (Linguagem de Consulta

[1] https://www.cloudflare.com/pt-br/learning/serverless/glossary/client-side-vs-server-side/
[2] https://techtipbits.com/security/input-validation-in-web-applications/

Estruturada)[3], linguagem de programação que lida com formação e processamento de consultas a banco de dados.

O processo de interação entre a aplicação do lado cliente com o lado servidor ou, no caso, requisições e transações HTTP, é recebe o nome de sessão quando performados pelo mesmo usuário. Entender o processo de gerenciamento de sessão[4] de uma aplicação ou serviço, sendo este o processo de lidar de forma segura com múltiplas requisições vindas de um único usuário ou entidade, é necessário para validar e testar procedimentos de segurança.

2.2 Vulnerabilidades de Segurança

Nesta seção serão descritos funcionalidades em que vulnerabilidades de segurança podem ser encontradas em uma aplicação *web*.

- Forja de Requisição entre Sites (*Cross-Site Request Forgery*): Essa vulnerabilidade pode levar a um ataque que força um usuário final a executar ações não intencionais em uma aplicação onde estão atualmente autenticados, através de alterações nas requisições da sessão[5]. Uma CSRF bem suscedida pode, por exemplo, fazer com que transações monetárias tenham seus valores e/ou seu destino alterados, e será executada sem verificação adicional, pois está vindo de um usuário legítimo.

- Injeção SQL (*SQL Injection*): Ocorre quando comandos SQL podem ser injetados em uma entrada de dados vinda do cliente utilizando a aplicação, com a intenção de afetar a execução de comandos SQL predefinidos. Em caso de sucesso, será possível modificar dados no banco de dados (Inserir, Atualizar e Deletar), executar operações administrativas (como desligar o Sistema Gerenciador de Banco de Dados) e acessar o conteúdo de arquivos presentes no sistema[6].

- *Cross-Site Scripting* (XSS): Trata-se de uma injeção de dados, na qual códigos maliciosos são inseridos e executados em sites outrora confiáveis[7]. Um ataque bem sucedido utilizando XSS pode levar a um *Clickjacking*, termo criado por Jeremiah Grossman e Robert Hansen em 2008 que é um acrônimo das palavras "*Click*" (Clique) e *"Hijacking"*

[3] https://www.w3schools.com/sql/sql-intro.asp
[4] https://www.veracode.com/security/session-management
[5] https://owasp.org/www-project-web-security-testing-guide/stable/4-Web-Application-Security-Testing/06-Session-Management-Testing/05-Testing-for-Cross-Site-Request-Forgery
[6] https://owasp.org/www-community/attacks/SQL-Injection
[7] https://owasp.org/www-community/attacks/xss/

(Sequestro), que literalmente significa sequestrar o clique de um usuário, enganando e induzindo-o a interagir com algo que não é o que o usuário pensa que está interagindo[8]. Ataques desse tipo são bastante comuns em fraudes de *websites*, utilizando de mímica e redirecionamento de página para pegar usuários desatentos e enviá-los para um site malicioso que lembra o site confiável em que estavam antes.

- Injeção HTML: Explora a vulnerabilidade de uma página *web* de permitir que um usuário controle um ponto de entrada e possa inserir arbitrariamente código HTML. Caso uma aplicação *web* seja vulnerável à Injeção HTML, o atacante pode ter acesso a *cookies* ou modificar o conteúdo da página a ser visto por vítimas.

- *Design* Inseguro: Engloba quaisquer brechas de segurança e falhas em arquitetura do sistema. Um *design* inseguro pode levar à vulnerabilidades que podem ser exploradas, mesmo que a implementação do sistema tenha sido feita de maneira segura. Mensagens de erro não tratadas, sessões sem *timeout* (encerramento automático em caso de inatividade), métodos ou valores de *tokens* expostos, entre outros, são todos exemplos de uma implementação insegura no *design* de uma aplicação.

- Falhas de Criptografia: Também denominada Exposição de Dados Sensíveis, esta vulnerabilidade está relacionada à implementação de criptografia em um sistema, ou à falta dela. Algoritmos ultrapassados, padrões facilmente identificáveis ou utilização de senhas fixas em código são exemplos de erros que levam à falhas de criptografia.

[8] https://owasp.org/www-project-web-security-testing-guide/stable/4-Web-Application-Security-Testing/11-Client-side-Testing/09-Testing-for-Clickjacking

3 SEGURANÇA DA INFORMAÇÃO

Segurança da Informação, como área de conhecimento, é dedicada à proteção de todos os ativos da informação contra acessos ou alterações não autorizados, ou sua indisponibilidade (SEMOLA, 2003).

Neste capítulo, será discutida a importância da Segurança da Informação nos dias atuais, onde dados são um recurso abundante e valioso e estão sob constante ameaça de atacantes, muitos deles tendo sucesso.

Os princípios de Segurança da Informação são derivados de três conceitos fundamentais (PFLEEGER; PFLEEGER, 2006):

- Confidencialidade: dados só podem ser vistos por entidades autorizadas.
- Integridade: dados só podem ser alterados ou manipulados por entidades autorizadas legalmente.
- Disponibilidade: dados estarão disponíveis para entidades autorizadas sempre que solicitados.

Tais princípios se alinham com os direitos tutelados pela Lei Geral de Proteção de Dados, onde a privacidade, a liberdadee a proteção quanto ao livre desenvolvimento da personalidade da pessoa natural.

Desta forma, é recomendado que as práticas e princípios de segurança devem estar completamente solidificados em uma aplicação, seguindo o artigo 46 da Lei Geral de Proteção de Dados, onde é dito que os detentores da informação devem adotar medidas de segurança aptas a proteger os dados pessoais de acessos não autorizados, destruição, perda, alteração, comunicação ou qualquer forma de tratamento inadequado, ilícito ou desconsentido.

3.1 Segurança e vulnerabilidades em Aplicações *W*eb

Aplicações *web* são frequentemente implementadas com um período curto para entrega final do projeto desenvolvido, práticas como esta são contra a implementação de boas arquiteturas de segurança (GUPTA; GUPTA, 2017), juntamente com o desconhecimento destas boas práticas. Isso pode resultar em brechas de segurança que podem ser detectadas e exploradas por um invasor, o que pode comprometer a integridade dos dados do sistema, ou o sistema como um todo, causando danos muitas vezes irreparáveis.

Alguns dos ataques mais comuns à aplicações web, especialmente no comércio eletrônico, são: *Cross-site Scripting (XXS); Cross-site Request Forgery (CSRF) e SQL Injection* (ESPINDOLA, 2017), estes que estão posicionados no topo da lista de vulnerabilidades da OWASP.

3.1.1 OWASP

A OWASP (*Open Web Application Security Project*) (OWASP, 2021) é uma organização independente e sem fins lucrativos que trabalha com o intuito de melhorar a segurança de *software*, especificamente das aplicações *web*.

A organização promove diversos eventos e em seu *site* dispõe de diversos artigos sobre desenvolvimento seguro, sobre as principais ameaças e vulnerabilidades e provê um guia de testes de segurança para identificá-las e previní-las, e também possuem ferramentas desenvolvidas para o auxílio ao aprendizado e desenvolvimento de aplicações seguras

3.1.2 Guia de Testes de Segurança *Web*

O projeto Web Security Testing Guide (WSTG) (WSTG, 2021) da OWASP consiste no principal recurso de teste de segurança cibernética para desenvolvedores de aplicações *web* e profissionais de segurança em um contexto geral.

O manual consiste de definições de ameaças e vulnerabilidades, seus impactos, explorações e prevenções ao desenvolver uma aplicação. Estas definições estão listadas em categorias relacionadas a cada funcionalidade de um sistema, incluindo o Lado-Cliente, Lado-Servidor e todas entre estes.

As práticas de programação seguras promovidas pela OWASP ajudam desenvolvedores de todo o mundo a implementar aplicações confiáveis e principalmente a saber como previnir e lidar com ameaças à segurança, entendendo seu funcionamento, exploração e consequências.

No presente trabalho, as vulnerabilidades de foco estarão nas categorias representadas dentre as quatro primeiras na versão de 2021 do TOP 10 OWASP (TOP10, 2021), que lista as vulnerabilidades mais exploradas e com maiores impactos em aplicações *web*, estas sendo:

- A vulnerabilidade à *Cross-Site Request Forgery* subiu da quinta posição para a primeira posição, listada como Controle de Acesso Corrompido (*Broken Acess Control)*[1];

[1] https://owasp.org/Top10/A01_2021-Broken_Access_Control/

- A vulnerabilidade à falhas de criptografia subiu da terceira para a segunda posição.
- A vulnerabilidade à *Cross-Site Scripting* atinge a terceira posição, sendo a mais comum no campo de injeções, seguida por injeções SQL[2];
- Em 2021, *Design* Inseguro entrou na quarta posição, como uma nova categoria de vulnerabilidade[3].

[2] https://owasp.org/Top10/A03_2021-Injection/
[3] https://owasp.org/Top10/A04_2021-Insecure_Design/

4 METODOLOGIA

Neste capítulo serão apresentados os objetos de estudo nos quais os testes foram feitos e também serão discutidos estes testes. O período de avaliação das aplicações compreende um intervalo de seis meses, tendo início ao decorrer de março e abril de 2021 e concluídos ao decorrer de setembro e outubro de 2021.

Todos os testes foram executados através do sistema Kali Linux[1], um sistema baseado em *Derbian* que possui, por padrão, ferramentas que auxiliam na execução de testes de segurança de todos os tipos. As ferramentas utilizadas para a execução dos testes, escolhidas devido à familiaridade do autor com as mesmas, foram:

- *Network Mapper (Nmap)*[2] (Figura 4.1), que trata-se de um *scanner* de rede, afim de mapear os serviços oferecidos, *hosts*, sistemas operacionais, entre outros, para obter o máximo de informações acerca do objeto de estudo. A ferramenta foi utilizada principalmente para identificar servidores e métodos HTTP utilizados pelas aplicações em questão.

- *Burp Suite*[3] (Figura 4.2), usada para testes de penetração em aplicações *web* utilizando interceptação de *proxy* e automação de processos de testes. Esta ferramenta permite entender a funcionalidade da aplicação e interpretar corretamente as queries e valores gerados, além de facilitar procedimentos de teste que precisavam de acesso a código.

A realização dos testes foi executada utilizando o método Caixa Preta (*Black Box*) e baseados nas vulnerabilidades que estão no topo da lista da OWASP, responsáveis pela maioria dos incidentes reportados. A Metodologia de Testes seguiu três diferentes abordagens:

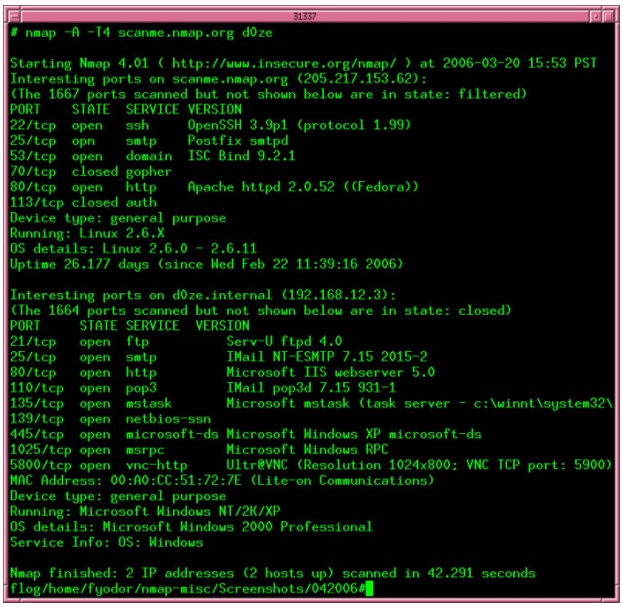
- Usuário comum não familiarizado com a aplicação;

- Um usuário mal intencionado que tenta acessar conteúdos ocultos ou forçar erros na página para extrair informações sobre o sistema para tentar encontrar maneiras de afetá-lo (*Hacker*).

- Um usuário mal intencionado que tem a intenção de enganar usuários comuns a fim de obter vantagem própria (Golpista).

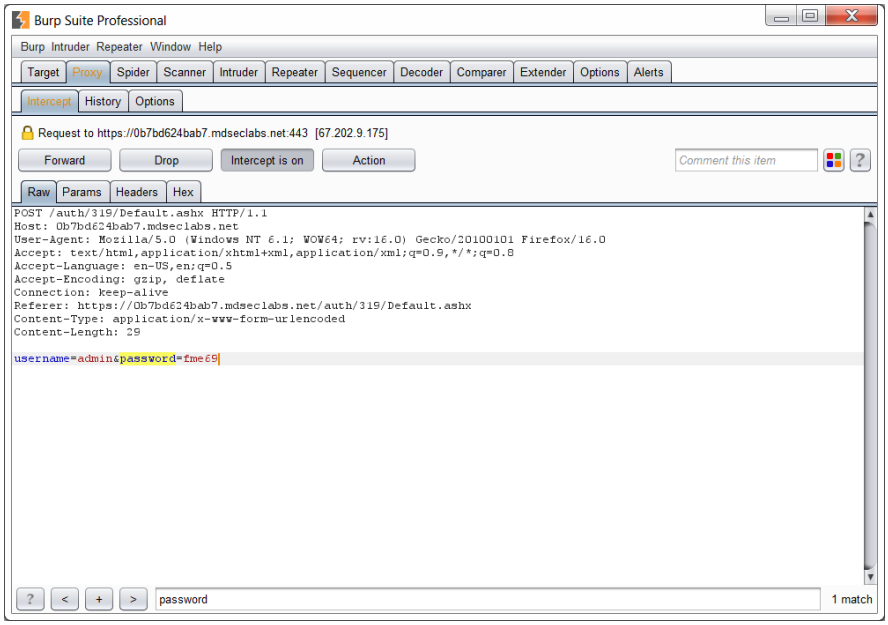
[1] https://www.kali.org
[2] https://nmap.org
[3] https://portswigger.net/burp

Figura 4.1 – Scan de Rede pela Nmap



Fonte: *Website* da Ferramenta

Figura 4.2 – Interceptação de Proxy pela Burp Suite



Fonte: *Website* da Ferramenta

Os Objetos de Estudo nos quais os testes foram realizados foram determinados através de estatísticas obtidas do Banco de Dados do Comércio Eletrônico[4] e tratam-se de quatro plataformas de comércio eletrônico operantes no mercado brasileiro, que mais apresentaram crescimento em receita e acessos em 2020 e 2021, estas que serão referidas, por questões éticas, como Objetos A, B, C e D, onde este último é o único a estar na categoria C2C.

[4] https://ecommercedb.com

4.1 Objetos de Estudo

4.1.1 Objeto A

O Objeto de estudo A, testado previamente por (TEIXEIRA, 2012), foi revisitado seguindo a última versão do WTSG, aplicando os novos testes e avaliando as novas tecnologias de segurança empregadas em sua plataforma. O Objeto trata-se de uma loja subsidiária especializada em Eletrônicos e Informática, sendo de grande renome no mercado brasileiro.

4.1.2 Objeto B

O Objeto de Estudo B trata-se de uma grande empresa de capital aberto que possui muitas subsidiárias e *B* é um dos maiores nomes no comércio nacional.

4.1.3 Objeto de Estudo C

O Objeto de Estudo C trata-se de uma grande loja que é subsidiária de uma empresa de capital aberto e é uma das mais antigas operantes no mercado nacional.

4.1.4 Objeto de Estudo D

O Objeto de Estudo D trata-se de um grande *marketplace* C2C de capital aberto que, apesar de não ser de origem brasileira, tem uma operação muito forte e consolidada em territorio nacional.

As vulnerabilidades testadas serão dispostas em categorias de acordo com o grupo de que fazem parte nas definições da OWASP.

4.2 Testes Realizados

4.2.1 Controle de Acesso Corrompido

Controle de acesso, ou autorização de acesso, é a forma com que uma aplicação concede acesso a conteúdos e funcionalidades para alguns usuários privilegiados na mesma, como administradores e moderadores. Essa checagem é feita após a autenticação do usuário e determinam as ações pemitidas a ele[5]. Um controle de acesso corrompido pode levar usuários a performar

[5] https://owasp.org/www-community/Broken_Access_Control

ações não intencionais ou tomar controle do acesso do mesmo, realizando operações que serão tidas como vindas do usuário legítimo.

4.2.1.1 Forja de Requisição Entre Sites

Uma tentativa de forja de requisição é baseada em:

- Comportamento do Navegador quanto a informações de sessão em Cookies e autenticação HTTP;
- Conhecimento do invasor de *URL's*, funcionalidades ou requisições válidas na aplicação;
- Gerenciamento de Sessão dependendo apenas em informações do navegador;
- A existência de *tags HTML (HyperText Markup Language)* cuja presença pode causar acesso imediato a recurso HTTP[S].

Para testar a vulnerabilidade dos Objetos à CSRF, foi utilizada manipulação de código de requisições HTML por interceptação de *proxy*, sendo utilizada a ferramenta *Burp Suite*.

Os testes consistem em alterar dados da sessão após feita uma requisição e tentar performar ações indevidas ou acessar, sem autorização, funcionalidades ou informações em uma sessão atualmente autenticada na aplicação.

Dentre os dados alterados nos testes estão: Valores de consultas, Tokens de Acesso, Valores de Cookies.

Ao capturar a requisição via *proxy*, estes valores são alterados e a requisição é reenviada, sendo observados os valores da resposta e comparando com a resposta anterior.

4.2.2 Falhas de Criptografia

4.2.2.1 Uso de senhas fixas em código (*Hard-coded Passwords*)

O uso de senhas fixas é geralmente uma falha na implementação da arquitetura de segurança de um sistema. Geralmente esta vulnerabilidade passa despercebida e é difícil de ser tratada.

Para testar se uma aplicação possui tal brecha em sua segurança, é necessário procurar quaisquer pontos em que senhas podem estar fixas, isto acontece, comumente, caso os implementadores não alterem ou removam valores padrão do sistema.

Considere o seguinte código de acesso a um banco de dados:

- *DriverManager.getConnection(url, "hicaro", "senhapadraoadmin");*

Este código é executado no lado-servidor da aplicação, qualquer um com acesso ao mesmo teria acesso à senha, e, consequentemente, ao sistema. Um atacante poderia também, caso tivesse acesso ao *bytecode* da página, utilizar um comando para "desmontar" o código, tendo acesso aos dados fixos isso pode ocorrer quando o programador deixar dados de acesso a API's no código *front-end*. É possível extrair estas informações usando ferramentas de desenvolvedor de qualquer navegador.

4.2.2.2 Entropia Insuficiente

Esta vulnerabilidade trata-se de quando uma aplicação utiliza baixa complexidade ou deixa à mostra seus padrões gerados para quaisquer valores, como tokens, ID's etc.

Para testar a entropia de dados de uma aplicação, basta observar quando quaisquer valores são gerados, por exemplo, considere a função:

- *"function gerarSessionID($idUsuario){ srand($idUsuario);*

  return rand();"

Como a *seed* utilizada na função aleatória é a ID do usuário, o valor da ID da sessão vai sempre ser a mesma, o que pode fazer com que um atacante tente adivinhar possíveis ID's e sequestrar uma sessão de um usuário.

4.2.3 Injeção

4.2.3.1 Injeção SQL (SQLi)

Todos os testes de injeção SQL feitos foram realizados manualmente, buscando pontos de injeção nas aplicações, que são qualquer ponto de entrada de dados cujos valores são enviados a um banco de dados, tais como campos de busca ou autenticação/cadastro; ou foram realizados utilizando interceptação de *proxy* via *Burp*, alterando os elementos de pesquisa através das páginas que utilizam o método *GET*.

Os objetos de estudo foram testados utilizando injeções de comandos SQL para tentar:

- Quebra de Acesso/Autenticação: Foi utilizada injeção SQL Booleana, onde são inseridos trechos parciais ou totais de consultas SQL nos campos de texto a fim passar por processos de autenticação de forma indevida.

Suponha que a operação de *login* de um site realize a seguinte consulta no banco de dados da aplicação:

- *"SELECT 'userToken' FROM users WHERE user=userid AND password=passwd;"*

Caso a entrada de dados não seja higienizada, é possível realizar uma quebra de autenticação adicionando elementos de consulta SQL juntamente com as entradas válidas com o seguinte comando:

- *"SELECT 'usrToken' FROM users WHERE user='usr' AND password='abc" OR '1'='1'- -+;"*

A adição de *"' OR '1'='1'- -+"* no campo de senha faz com que a consulta SQL ignore as aspas simples colocadas automaticamente no valor de entrada da senha, interpretando-o como parte da sintaxe da consulta realizada.

Como 1=1 é sempre verdadeiro, ao combinarmos este valor com o operador "OR", não é necessário saber ou prover valores válidos para "*username*"e "*password*", de tal forma, independente de quais valores forem informados, 1=1 é sempre verdadeiro, portanto o acesso ao sistema seria autorizado, e é possível obter privilégios de acesso como, por exemplo, de administrador da aplicação.

- Listagem de Banco de Dados: O teste de Injeção SQL utilizado foi a Injeção Baseada em União, cujo objetivo é tentar determinar quantas colunas há em uma tabela e encontrar qual delas possui valores *String*, para, então, listá-los ou acessar outras tabelas.

  O operador “UNION” da SQL permite executar um ou mais comandos "SELECT", anexando os resultados à consulta original. Considere a seguinte consulta:

  - *"SELECT 'x', 'y' FROM tabela1 UNION SELECT 'z', 'w' FROM tabela2"*

  A consulta vai retornar um único resultado, composto dos valores das colunas ’x’ e ’y’, da tabela1; e ’z’ e ’w’, da tabela2.

  Para uma consulta utilizando “UNION” ser executada, as consultas individuais devem retornar o mesmo número de colunas e os seus tipos de dado devem ser compatíveis.

4.2.3.2 *Cross-site Scripting* (XSS)

Um ataque XSS consiste em manipular uma aplicação *web* para enviar uma saída maliciosa a um outro usuário, e, geralmente, o navegador vai executá-lo, pois parece que o código veio de uma fonte confiável.

O tipo mais comum de XSS é o refletido, que é performado ao manipular os parâmetros de um URL e fazendo com que um usuário acesse-o, potencialmente levando-o a performar ações não intendidas ou expor dados de sessão.

Os testes consistem em encontrar pontos de entrada em um URL e tentar executar um *script* ao acessar a página ou fazer com que o URL leve o usuário a um lugar vulnerável.

Considere o URL de busca: *www.site.com/busca='busca'*

Um site é vulnerável a *Cross-Site Scripting* quando, por exemplo, não filtra os campos de texto e permite com que o termo de busca possa ser um comando ou um hiperlink, como:

- *www.site.com/busca=<script>alert(XSS)</script>*: se ao acessar o URL, o usuário ver um *pop-up* com o alerta, a aplicação é vulnerável a XSS Refletido.
- *www.site.com/busca=www.outrosite.com/malware*: se ao acessar o URL, o usuário ser redirecionado para o outro endereço, a aplicação é vulnerável a XSS Refletido.

O tipo mais perigoso de XSS é o armazenado[6]. Aplicações *web* que permitem ao usuário armazenar dados que podem ser acessados posteriormente, como páginas de perfil, carrinho de compras, configurações, entre outros, podem ser utilizadas como hospedeira para o ataque. Caso a entrada não seja verificada, é possível inserir código malicioso que será executado sempre que a página for acessada.

Considere uma aplicação em que é possível um usuário fazer comentários. Caso a aplicação seja vulnerável a XSS Armazenado, um usuário poderia, por exemplo, colocar um script que captura valores de *cookie*, utilizando a função *document.cookie()*, e tentar fazer com que outro usuário o execute. Isso poderia levar, por exemplo, a um usuário malicioso, alterar os valores do próprio *cookie*, roubando a identidade de outro usuário dentro da aplicação e executar ações como o mesmo.

[6] https://owasp.org/www-project-web-security-testing-guide/stable/4-Web-Application-Security-Testing/07-Input-Validation-Testing/02-Testing-for-Stored-Cross-Site-Scripting

4.2.4 *Design* Inseguro

4.2.4.1 Geração de mensagens de erro contendo informações sensíveis

Ao performar requisições ou ações que propositalmente levem a um erro na aplicação, na maioria das vezes, o erro é tratado corretamente e uma mensagem é mostrada ao usuário indicando que não foi possível concluir a ação performada. Há casos, entretanto, em que a mensagem de erro não é tratada e virá contendo informações sobre o ambiente da aplicação, usuários ou dados associados com a ação.

Os testes consistem em prover dados em formatos ou tipos incorretos para campos de entrada, alterar caminhos de URL e induzir erros de sintaxe em interpretadores. Esta vulnerabilidade pode facilitar a execução de ataques ao sistema, dada a exposição de informações úteis sobre seu funcionamento.

4.2.4.2 Tempo de Desligamento de Sessão

O teste foi realizado ao autenticar um usuário válido na plataforma e deixá-la inativa, retornando periodicamente e tentando performar uma ação, especialmente ações que só podem ser realizadas ao estar autenticado na plataforma, como visualizar os dados do usuário, ou acessar a página de pagamentos. Também foi testada a possibilidade de ressuscitar uma sessão, em que, após encerrada, é feita uma tentativa de regressar a um estado em que a sessão ainda estava ativa, utilizando a função “voltar” do navegador.

5 RESULTADOS OBTIDOS

Neste capítulo estão os resultados obtidos em cada teste realizado, consistindo na resposta de cada objeto aos testes e onde há, bem como onde não há, vulnerabilidades que possam ser exploradas a fundo e comprometer a integridade do sistema.

5.1 Resultados dos Testes

5.1.1 Tempo de Desligamento de Sessão

Figura 5.1 – Tabela Comparativa de Resultados

Objeto A	Possui Session Timeout
Objeto B	Possui Session Timeout
Objeto C	Possui Session Timeout
Objeto D	Não Possui Session Timeout

Fonte: Autor

O período de esgotamento de tempo observado foi entre 5 e 10 minutos de inatividade para os Objetos A, B e C.

O Objeto D demonstra ser o mais vulnerável a uma ressuscitação de sessão, visto que o sistema não encerra automaticamente a sessão autenticada atual, permitindo à mesma ser acessada posteriormente, por qualquer usuário, sem necessidade de reautenticação. Além disso, não há encerramento de sessão ao fechar o navegador ou a página *web*, mesmo não havendo opção de "Manter-se Conectado" ao autenticar-se na plataforma, fazendo com que um usuário tenha sua conta acessada por um outro usuário, caso este reabra a página acessada.

5.1.2 Cross-Site Request Forgery

Todos os objetos de estudo se mostraram capazes de lidar com tentativas de Forja de Requisição através de interceptação e alteração de requisições em um nível mais básico. Os objetos tratavam as requisições manipuladas como inválidas ou não autorizadas.

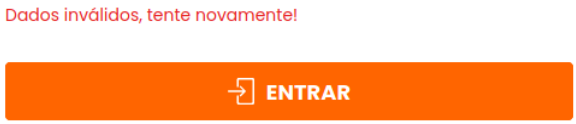
5.1.3 SQL Injection

5.1.3.1 Quebra de Autenticação

As respostas às tentativas de quebra de autenticação via injeção SQL foram praticamente as mesmas em todas os objetos, sendo observada mudanças apenas na mensagem de erro ao usuário. O objeto A informa apenas que os dados inseridos são inválidos (Figura 5.2); O objeto B pede que as informações sejam verificadas novamente (Figura 5.3); O objeto C apresenta um *pop-up* de erro informando que algum dos dados informados está errado (Figura 5.4); O objeto D pede que as credenciais sejam revisadas antes de continuar (Figura 5.5).

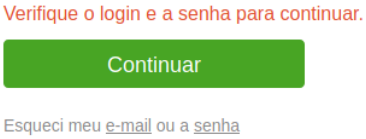
Analisando as repostas, pode-se inferir que operadores SQL são tratados nos campos de entrada, sendo barrados ou higienizados pela aplicação.

Figura 5.2 – Resposta à tentativa de Injeção SQL nos campos de autenticação do Objeto A



Fonte: *Website do Objeto A*

Figura 5.3 – Resposta à tentativa de Injeção SQL nos campos de autenticação do Objeto B

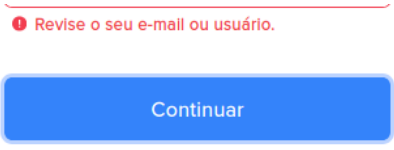


Fonte: *Website do Objeto B*

Figura 5.4 – Resposta à tentativa de Injeção SQL nos campos de autenticação do Objeto C

Fonte: *Website do Objeto C*

Figura 5.5 – Resposta à tentativa de Injeção SQL nos campos de autenticação do Objeto D



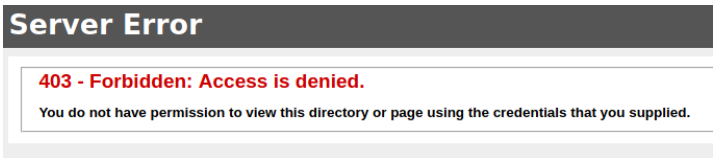
Fonte: *Website do Objeto D*

5.1.3.2 Listagem e Obtenção de dados do Banco de Dados

A arquitetura da aplicação *web* dos objetos A, B e D não deixam elementos parametrizados à mostra em páginas que possuem pontos de entrada de dados, dificultando o acesso direto ao método *GET* e consequentemente impossibilitando a execução de uma injeção SQL visando acessar dados das tabelas por requisições de *URL*, porém, ainda é possível utilizar injeções para forçar erros ou tentar obter mais resultados. Os objetos A, B e D tratam todos os códigos que possuem sintaxe SQL como texto pleno, adicionando caracteres para diferenciá-los na hora de realizar a operação.

A resposta do Objeto C à tentativa da injeção na sua página de busca resulta em um código 403, o que significa que não é possível acessar, sem permissão, o servidor para executar a requisição; o código aparece juntamente com um erro de servidor (Figura 5.6). A presença de um código 403 mostra que o *site* agiu corretamente em vista da requisição não autorizada, mas pode também significar que, de alguma outra maneira, um atacante ainda pode conseguir passar pela barreira.

Figura 5.6 – Página de Acesso Negado do Objeto C (Código 403: *Forbidden*)



Fonte: *Website do Objeto C*

Quando uma tentativa de injeção resulta em um erro direto dentro do servidor, pode-se inferir que a consulta foi realmente processada. Um código 403 é um código de redirecionamento, o que significa que é possível acessar uma página que um usuário sem acesso privilegiado não deveria.

5.1.4 Cross-Site Scripting

Ao tentar executar código pelos pontos de entrada da páginas de busca e dados cadastrais, o comportamento observado dos objetos foi:

- Objetos A e D higienizam as entradas do usuário antes de executá-las, fazendo com que as mesmas não afetem em nada a aplicação. A higienização é feita interpretando os operadores como texto pleno ou simplesmente removendo-os antes de executar a ação.

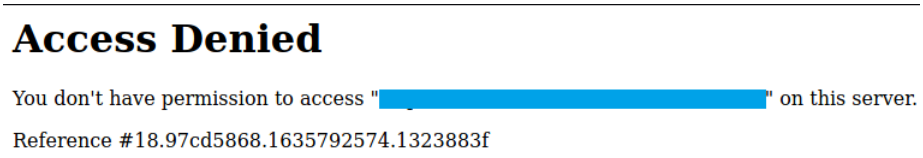
- Objetos B e C retornam uma página de código 400 (Figura 5.7) e 403 (Figura 5.8), respectivamente, o que é a resposta recomendada do servidor quando uma tentativa de ataque é performada.

Nos testes para *Cross-Site Scripting*, foram também realizadas instâncias que utilizavam Injeção HTML e Injeção PHP, onde todos os Objetos se mostraram não vulneráveis às mesmas, apresentando o mesmo tratamento e resposta observados.

Figura 5.7 – Página de Acesso Negado do Objeto B (Código 400: *Bad Request*)

Fonte: *Website do Objeto B*

Figura 5.8 – Página de Acesso Negado do Objeto C (Código 403: *Forbidden*)



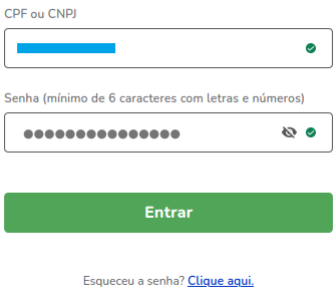
Fonte: *Website do Objeto C*

O Objeto C, todavia, mostrou-se vulnerável à XSS refletido na página de *login* da aplicação (figura 5.9), onde a URL de redirecionamento pode ser interceptada via *proxy*, ou até mesmo pelo próprio navegador, e o endereço de redirecionamento pode ser alterado, fazendo com que o usuário seja redirecionado para qualquer página que o alterador quiser.

Um outro ponto importante a se destacar é que o certificado de conexão segura do *site* não é alterado (Figura 5.10), pois o URL alterado faz parte da própria página genuína da aplicação, sendo assim, para o servidor, qualquer operação é interpretada como vinda de uma fonte confiável.

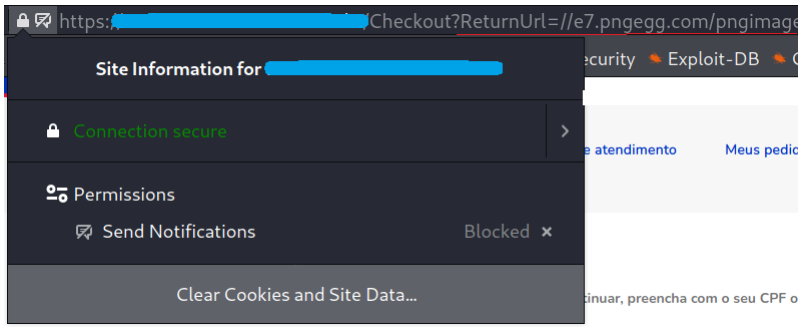
A vulnerabilidade à XSS encontra-se no parâmetro *ReturnUrl*, onde qualquer URL pode ser armazenado e acessado. Com o endereço de redirecionamento alterado, mesmo estando na página oficial do Objeto, ao efetuar o *login*, o usuário vítima será enviado, pela página confiável, à uma página de confiança questionável (Figura 5.11). Este teste caracteriza um

Figura 5.9 – Tela de Login do Objeto C com endereço de redirecionamento alterado



Fonte: *Site* do Objeto C

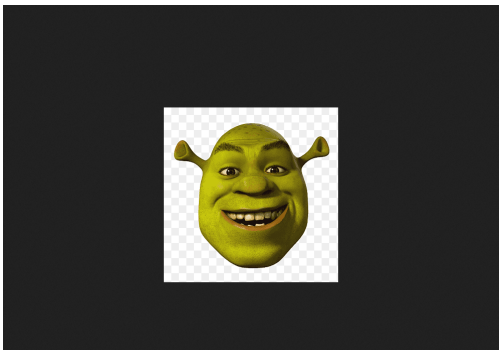
Figura 5.10 – Certificado de Segurança inalterado com URL alterado



Fonte: *Website* do Objeto C

exemplo, também, de *Clickjacking*, onde o usuário interage com a aplicação do Objeto C e é enviado sem consentimento à outra página, podendo ser induzido a performar mais ações não intendidas.

Figura 5.11 – Pagina redirecionada para Imagem do Shrek ao se autenticar com sucesso.



Fonte: *PngEgg*[1]

Uma vulnerabilidade como esta, faz com que usuários maliciosos possam enganar outros usuários e enviá-los para páginas inseguras, acessar as credenciais digitadas ou até mesmo fazê-los baixar arquivos automaticamente, bastando apenas fazer com que os mesmos acessem o URL modificado. Uma consequência de uma exploração de tal vulnerabilidade inclui exposição

de dados sensíveis do usuário, o que viola diretamente o artigo 46 da Lei Geral de Proteção de Dados, podendo acarretar, também, consequências legais para a empresa.

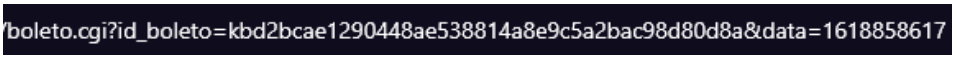
5.2 Considerações Finais

As aplicações testadas mostraram-se atentas com as informações sobre a aplicação mostradas ao usuário em casos de ações indevidas ou simplesmente erradas. Nenhuma informação útil a um atacante pôde ser obtida através de mensagens vindas da aplicação e também são omitidas informações úteis ao usar *scanners*, como o *Nmap*, utilizado neste trabalho.

A aplicação A demonstrou uma grande melhoria em sua segurança em comparação ao trabalho anterior, a maioria dos pontos problemáticos levantados por (TEIXEIRA, 2012) foram corrigidos. Um dos pontos, entretanto, sofreu pouca alteração, este sendo o fato de o acesso à boletos gerados para um usuário não depender de nenhum dado de sessão ou autenticação deste usuário.

As páginas podem ser acessadas arbitrariamente por qualquer um que detenha um URL para um boleto, e a geração deste ainda é realizada pelos parâmetros *id_boleto* e data em UNIX *Timestamp* (Figura 5.12). Embora a geração do parâmetro *id_boleto* tenha se tornado complexa em comparação à versão antiga, ainda é possível acessar quaisquer boletos gerados no sistema, independente de quando foram gerados, caso a lógica para a geração de um *id* seja descoberta, e este processo pode ser facilmente automatizado com a utilização de robôs.

Figura 5.12 – Parâmetros para geração de boleto do Objeto A



Fonte: *Website do Objeto A*

As aplicações B e D mostraram-se seguras em relação às vulnerabilidades testadas, não apresentando falhas de segurança ou brechas visíveis ou de fácil acesso que possam ser exploradas em um ataque Caixa Preta, mas não quer dizer que estejam completamente seguras.

A aplicação C mostrou-se vulnerável à pontenciais fraudes e exposição de dados devido à brecha para XSS em sua função de *login*. Em questão às outras vulnerabilidades, a aplicação lidou bem com cada uma delas, mas uma brecha como esta pode por em risco todas as implementações de segurança acerca das outras e esta não pode ser facilmente detectável. Outro ponto importante é que não é necessário grande conhecimento de técnicas de invasão ou pro-

gramação, fazendo com que seja muito acessível e fácil explorar esta vulnerabilidade, sendo um grande atrativo para golpistas.

Pode-se concluir, a partir dos dados observados, que a maior parte das aplicações de comércio eletrônico testadas está devidamente segura dentro dos parâmetros de segurança da informação, mas ainda há brechas que podem resultar em grandes prejuízos para as empresas e seus usuários. Os resultados demonstram que estas plataformas estão se preocupando com sua integridade e protegendo a si e a seus usuários contra invasores, mas, riscos de invasão não são a única ameaça à integridade de uma plataforma.

6 CONCLUSÃO

O trabalho realizado abordou uma pequena parcela, ainda que importante, do mercado eletrônico brasileiro e esta foi testada apenas para as vulnerabilidades diretas que uma aplicação pode ter em sua implementação ou *design*. Muitos ataques cibernéticos não são voltados diretamente à uma aplicação, estes também procuram brechas em quaisquer vínculos com terceiros em que vulnerabilidades possam ser encontradas e exploradas.

Pode-se citar, como exemplo de tal ataque indireto, o recente vazamento de 1,7 bilhão de registros[1], ultrapassando 610 GB (*Giga Bytes)* de dados confidenciais expostos, ocorrido no dia 13 de outubro de 2021. A aplicação alvo foi uma plataforma de integração de *marketplace* e comércio eletrônico, a qual muitas das grandes empresas atuantes no cenário fazem parte.

O fator legal imposto pela LGPD acerca da garantia de segurança de dados tem um impacto ainda maior para as entidades que violarem suas diretrizes. Como visto no trabalho, há entidades com potenciais falhas que podem levar à uma exploração que viola o direito à privacidade, um dos pontos mais enfatizados pela lei.

A questão da Segurança da Informação engloba muitas variáveis que, muitas vezes, estão fora da capacidade de empresas de lidar diretamente, como falhas externas e humanas. Mas, um sistema seguro já garante uma base sólida para o desenvolvimento de políticas e aplicações que conseguem devidamente assegurar e proteger o seu recurso mais valioso, os dados de usuários.

Dos resultados obtidos durante a realização deste trabalho, pode-se inferir que boa parte das aplicações *web* de comércio eletrônico atuantes no cenário brasileiro estão se preocupando com a segurança dos seus usuários e de suas plataformas, embora ainda existam brechas que possam ser exploradas.

Apesar dos resultados terem sido positivos em sua maioria, os testes realizados não certificam tais aplicações como devidamente seguras, necessitando de testes mais aprofundados e abordando várias outras áreas que, embora não ocupando posições altas no TOP 10, ainda são vulnerabilidades críticas apontadas pela OWASP em seu Guia de Testes e que podem, potencialmente, estar presentes mesmo nas aplicações planejadas com o devido cuidado com uma implementação segura. A metodologia abordada neste trabalho, apesar de ter sido aplicada em aplicações de comércio eletrônico, pode ser utilizada em qualquer tipo de aplicação que arma-

[1] https://thehack.com.br/incorporadora-de-ecommerces-deixa-mais-de-1-75-bilhao-de-dados-expostos/

zena, coleta ou transporta dados sensíveis de usuários, visto que estas também estão inclusas na Lei geral de Proteção de Dados.

REFERÊNCIAS

ESPINDOLA, M. Segurança em comércio eletrônico. **n International Journal of Cloud Applications and Computing**, 2017. Disponível em: <https://repositorio.animaeducacao.com.br/handle/ANIMA/3715>.

FIELDING, R. **Hypertext Transfer Protocol–HTTP/1.1**. 1999. Disponível em: <https://www.w3.org/Protocols/HTTP/1.1/rfc2616.pdf>.

GUPTA, S.; GUPTA, B. Detection, avoidance, and attack pattern mechanisms in modern web application vulnerabilities: Present and future challenges. **n International Journal of Cloud Applications and Computing**, 2017. ISSN 2156-1834. Disponível em: <https://www.igi-global.com/gateway/article/182251>.

HOOGENRAAD, W. **Teste de caixa preta: Software no rack**. 2019. Disponível em: <https://pt.itpedia.nl/2019/01/23/black-box-testing-software-op-de-pijnbank/>.

LGPD. 2018. Disponível em: <http://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/Lei/L13709.htm>.

LI, X.; XUE, Y. A survey on server-side approaches to securing web applications. 2014. Disponível em: <https://doi.org/10.1145/2541315>.

OWASP. 2021. Disponível em: <https://owasp.org>.

PFLEEGER, C.; PFLEEGER, S. **Security in Computing (4th Edition)**. [S.l.: s.n.], 2006. ISBN 0132390779.

SEMOLA, M. **Gestão da Segurança da Informação: uma visão executiva**. [S.l.: s.n.], 2003. ISBN 8535271783.

TEIXEIRA, T. S. **TESTES DE SEGURANÇA EM APLICAÇÕES WEB SEGUNDO A METODOLOGIA OWASP**. 2012. Disponível em: <http://repositorio.ufla.br/bitstream/1/5059/1/MONOGRAFIA_Testes%20de%20segurança%20em%20aplicaç~oes%20web%20segundo%20a%20metodologia%20OWASP.pdf>.

TOP10. 2021. Disponível em: <https://owasp.org/Top10/>.

WSTG. 2021. Disponível em: <https://owasp.org/www-project-web-security-testing-guide/>.