PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
ESCOLA POLITÉCNICA E DE ARTES
GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

ANÁLISE DE ATAQUES CIBERNÉTICOS EM SERVIDORES WEB E ESTRATÉGIAS COMPUTACIONAIS PARA MITIGAÇÃO

HENRIQUE BARBOSA DE LIMA DIAS

GOIÂNIA – GO,
2024

HENRIQUE BARBOSA DE LIMA DIAS

ANÁLISE DE ATAQUES CIBERNÉTICOS EM SERVIDORES WEB E ESTRATÉGIAS COMPUTACIONAIS PARA MITIGAÇÃO

Trabalho de Conclusão de Curso apresentado à Escola Politécnica e de Artes, da Pontifícia Universidade Católica de Goiás, como requisito parcial para a obtenção do título de bacharel em Engenharia de Computação.

Orientador: Prof. Me. Rafael Leal Martins

GOIÂNIA – GO,

2024

HENRIQUE BARBOSA DE LIMA DIAS

ANÁLISE DE ATAQUES CIBERNÉTICOS EM SERVIDORES WEB E ESTRATÉGIAS COMPUTACIONAIS PARA MITIGAÇÃO

Trabalho de Conclusão de Curso, julgado adequado para obtenção do título de Bacharel em Engenharia de Computação, e aprovado em sua forma final pela Escola Politécnica e de Artes da Pontifícia Universidade Católica de Goiás, em 13/12/2024.

_____

Luiz Álvaro de Oliveira Júnior

Coordenador(a) de Trabalho de Conclusão de Curso

Banca examinadora:

_____

Orientador: Prof. Me. Rafael Leal Martins

_____

Prof. Me. Fernando Gonçalves Abadia

_____

Prof. Dr. Nilson Cardoso Amaral

GOIÂNIA – GO,

2024

AGRADECIMENTOS

À minha mãe, Iocyene, e à minha avó, Maria, pelo apoio incondicional e pelo esforço incansável para que eu pudesse chegar até aqui. Sua dedicação e sacrifícios sempre foram uma inspiração para mim. Sem o seu suporte, tanto emocional quanto prático, nada disso teria sido possível. Obrigado por acreditarem nos meus sonhos mesmo quando as dificuldades surgiam, sendo o alicerce que me sustentou em toda essa jornada.

Ao meu orientador, Rafael Leal Martins, pela orientação e pelas valiosas contribuições durante o desenvolvimento deste trabalho. Sua orientação e apoio foram fundamentais para a conclusão deste projeto.

Aos meus amigos Jean Carlos, Bruno Raphael e João Marcos, que estiveram comigo durante toda a trajetória do curso. Compartilhamos desafios, aprendizados e muitas conquistas, sempre ajudando uns aos outros nos momentos mais difíceis. Vocês tornaram essa caminhada mais leve e significativa, e sou grato por sua amizade, parceria e pelas incontáveis horas de estudo, conversa e motivação mútua.

A todos que, de alguma forma, me ajudaram a chegar até aqui, meu mais sincero obrigado. Este trabalho reflete, também, a contribuição e o apoio de cada um de vocês.

RESUMO

Este trabalho explora a natureza e a execução de ataques cibernéticos prevalentes contra servidores web, com foco nos ataques *Man-in-the-Middle (MitM)*, *SQL injection* e de força bruta. O ataque MitM envolve a interceptação e manipulação da comunicação entre duas partes, permitindo que o invasor altere ou capture dados sensíveis. O *SQL injection* possibilita que atacantes manipulem consultas SQL, dando acesso a informações confidenciais não autorizadas, enquanto os ataques de força bruta tentam adivinhar credenciais por meio de tentativas repetitivas. A pesquisa adota uma metodologia que combina revisão bibliográfica com experimentos práticos, utilizando ferramentas como Ettercap, SQLMap *e* Hydra para identificar vulnerabilidades em servidores web e testar a eficácia das defesas implementadas. Além da análise dos ataques, o trabalho também apresenta uma demonstração prática de defesa, incluindo a implementação de um site seguro com a validação de entradas, utilização do protocolo HTTPS e autenticação segura. Essas defesas são implementadas e testadas com o objetivo de proteger dados em trânsito, prevenir acessos não autorizados e mitigar os ataques simulados, oferecendo uma proteção eficaz contra as ameaças cibernéticas. A segurança do site é avaliada com base na resistência às ameaças identificadas, destacando a importância de práticas de segurança adequadas, como o uso de criptografia para garantir a confidencialidade das informações. Os resultados indicam que a combinação dessas medidas é altamente eficaz na proteção de servidores web, reforçando a segurança geral do sistema e oferecendo uma defesa sólida contra as vulnerabilidades exploradas. Conclui-se que a aplicação de defesas robustas é essencial para garantir a integridade, a confiabilidade e a proteção contínua dos sistemas.

Palavras-chave: Ataques Cibernéticos; Servidores Web; *Man-in-the-Middle*; *SQL Injection*; Força Bruta; Criptografia.

ABSTRACT

This work explores the nature and execution of prevalent cyberattacks against web servers, focusing on Man-in-the-Middle (MitM) attacks, SQL injection, and brute-force attacks. The MitM attack involves intercepting and manipulating communication between two parties, allowing the attacker to alter or capture sensitive data. SQL injection enables attackers to manipulate SQL queries, gaining unauthorized access to confidential information, while brute force attacks attempt to guess credentials through repetitive trial and error. The research adopts a methodology that combines bibliographic review with practical experiments, using tools such as Ettercap, SQLMap, and Hydra to identify vulnerabilities in web servers and test the effectiveness of implemented defenses. In addition to the analysis of attacks, the work also presents a practical defense demonstration, including the implementation of a secure website with input validation, the use of the HTTPS protocol, and secure authentication. These defenses are implemented and tested with the goal of protecting data in transit, preventing unauthorized access, and mitigating simulated attacks, providing effective protection against cyber threats. The website's security is assessed based on its resistance to identified threats, highlighting the importance of adequate security practices, such as the use of encryption to ensure information confidentiality. The results indicate that the combination of these measures is highly effective in protecting web servers, strengthening the overall security of the system, and offering solid defense against the exploited vulnerabilities. It is concluded that the application of robust defenses is essential to ensure the integrity, reliability, and continuous protection of systems.

Keywords: Cyber Attacks; Web Servers; Man-in-the-Middle; SQL Injection; Brute Force; Cryptography.

LISTA DE FIGURAS

Figura 1: Transação HTTP 16
Figura 2: Transação HTTPS 17
Figura 3: Diferenças Técnicas 19
Figura 4: Pacote HTTP interceptado com Paros Proxy 21
Figura 5: Injeção de SQL baseada em Union 22
Figura 6: Injeção de SQL baseada em tempo 23
Figura 7: 5 tipos de ataques de força bruta 25
Figura 8: Ferramenta Ettercap 27
Figura 9: Ferramenta Wireshark 28
Figura 10: Tentativa de login 29
Figura 11: Execução do Ettercap 29
Figura 12: Análise de tráfego com o Wireshark 30
Figura 13: Ferramenta SQLMap 31
Figura 14: Site Acunetix Art 31
Figura 15: Vulnerabilidade do site Acunetix Art 32
Figura 16: Conexão e testes iniciais usando SQLMap 33
Figura 17: Banco de dados encontrados usando SQLMap 34
Figura 18: Acesso às tabelas do banco de dados 35
Figura 19: Listagem de tabelas do banco de dados acuart 35
Figura 20: Acesso às colunas da tabela users 36
Figura 21: Dados armazenados na tabela users 37
Figura 22: Acesso aos atributos na tabela users 38
Figura 23: Dados das colunas pass e uname na tabela users 38
Figura 24: Login no site Acunetix Art 39
Figura 25: Acesso às informações do usuário 39
Figura 26: Busca por dados de usuário 40
Figura 27: Busca por email do usuário 41
Figura 28: Busca por dados de usuário 42
Figura 29: Busca por cc do usuário 42
Figura 30: Varredura usando Nmap 43
Figura 31: Acesso a página de login 44
Figura 32: Campos Username e Password 44

Figura 33: Wordlist_user 45
Figura 34: Possíveis nomes de usuário 45
Figura 35: Wordlist_pass 45
Figura 36: Possíveis senhas 46
Figura 37: Execução do comando hydra 46
Figura 38: Início da execução 47
Figura 39: Tentativas de login 47
Figura 40: Tentativas de login 48
Figura 41: Conclusão da execução 48
Figura 42: Página de login 54
Figura 43: Lógica para estabelecer conexão com o banco de dados 55
Figura 44: Estrutura dos dados armazenados 55
Figura 45: Uso do HTTPS no site de login 56
Figura 46: Função para sanitização de entradas 57
Figura 47: Sanitização de entradas 57
Figura 48: Consultas preparadas 58
Figura 49: Verificação de senha 58
Figura 50: Entrada do ataque no site vulnerável 59
Figura 51: Resultado de login no site vulnerável 60
Figura 52: Entrada do ataque no site seguro 60
Figura 53: Resultado de login no site seguro 61
Figura 54: Limite de tentativas de login 61
Figura 55: Sessão de tentativas de login 62
Figura 56: Tentativas de login e tempo de bloqueio 63
Figura 57: Validação de usuário e senha 63
Figura 58: Tentativas de login excedidas 64
Figura 59: Sucesso ao fazer login 64

LISTA DE SIGLAS E ABREVIATURAS

ASCII	*American Standard Code for Information Interchange*, Código Padrão Americano para Intercâmbio de Informação
CA	*Certificate Authority*, Autoridade Certificadora
CAPTCHA	*Completely Automated Public Turing Test to Tell Computers and Humans Apart*, Teste de Turing Público e Automático para Diferenciar Computadores de Humanos
CSRF	*Cross-Site Request Forgery*, Falsificação de Requisições entre Sites
HSTS	*HTTP Strict Transport Security*, Segurança Rígida de Transporte HTTP
HTML	*HyperText Markup Language*, Linguagem de Marcação de Hipertexto
HTTP	*Hypertext Transfer Protocol*, Protocolo de Transferência de Hipertexto
HTTPS	*Hypertext Transfer Protocol Secure*, Protocolo de Transferência de Hipertexto Seguro
MFA	*Multifactor Authentication*, Autenticação Multifator
MitB	*Man-in-the-Browser*, Ataque Homem no Navegador
MitM	*Man-in-the-Middle*, Ataque Homem no Meio
NMAP	*Network Mapper*, Mapeador de Redes
OAST	*Open Source Application Security Testing*, Teste de Segurança de Aplcações de Código Aberto
OWASP	*Open Web Application Security Project*, Projeto Aberto de Segurança em Aplicações Web
PHP	*Hypertext Preprocessor*, Pré-processador de Hipertexto
SGBD	Sistema de Gerenciamento de Banco de Dados
SQL	*Structured Query Language*, Linguagem de Consulta Estruturada
SSL	*Secure Sockets Layer*, Camada de Sockets Seguros
TCP	*Transmission Control Protocol,* Protocolo de Controle de Transmissão
TLS	*Transport Layer Security*, Segurança da Camada de Transporte
URI	*Uniform Resource Identifier*, Identificador Uniforme de Recursos
URL	*Uniform Resource Locator*, Localizador Uniforme de Recursos
WAF	*Web Application Firewall*, Firewall de Aplicações Web
XSS	*Cross-Site Scripting*, Script Entre Sites
IDS	*Intrusion Detection System*, Sistema de Detecção de Intrusões

IPS *Intrusion Prevention System*, Sistema de Prevenção de Intrusões

SUMÁRIO

1 INTRODUÇÃO 11
1.1 Justificativa 12
1.2 Objetivos 13
1.2.1 Objetivo geral 13
1.2.2 Objetivos específicos 13
1.3 Procedimentos metodológicos 13
1.4 Estrutura da monografia 14
2 REFERENCIAL TEÓRICO 15
2.1 Protocolo HTTP 15
2.1.1 Funcionamento do HTTP 15
2.1.2 Métodos HTTP 16
2.1.3 Vulnerabilidades do HTTP 16
2.2 Protocolo HTTPS 17
2.2.1 Funcionamento do HTTPS 17
2.2.2 TLS/SSL: Transport Layer Security e Secure Sockets Layer 18
2.2.3 Vantagens do HTTPS 18
2.2.5 Implementação do HTTPS 19
2.2.6 Comparação entre HTTP e HTTPS 19
2.3 Man-in-the-Middle (MitM) 20
2.4 SQL injection 21
2.4.1 Injeção de SQL baseada em *Union* 22
2.4.2 Injeção de SQL baseada em Erro 22
2.4.3 Injeção de *SQL Blind* 23
2.5 Força bruta 24
2.5.1 Ataques de força bruta simples 25
2.5.2 Ataques de dicionário 26

2.5.3 Ataques de força bruta híbrida 26
2.5.4 Ataques de força bruta reversa 26
2.5.5 Preenchimento de credenciais 26
3 SIMULAÇÕES DOS ATAQUES CIBERNÉTICOS 27
3.1 Simulação *Man-in-the-Middle (MitM)* 27
3.1.1 Tentativa de *login* no servidor alvo 28
3.1.2 Configuração e execução do Ettercap 29
3.1.3 Análise de tráfego com o Wireshark 30
3.2 Simulação *SQL injection* 30
3.2.1 Indicação de vulnerabilidade 31
3.2.2 Exploração de vulnerabilidades utilizando SQLMap 33
3.2.3 Extração de dados 34
3.2.4 Fazendo *login* com usuário e senha obtidos 39
3.2.5 Busca por dados sensíveis de usuário fictício 40
3.3 Simulação de ataque de força bruta 43
3.3.1 Indicação de vulnerabilidade 43
3.3.2 Criação de *wordlists* 45
3.3.3 Uso do Hydra para força bruta 46
4 ESTRATÉGIAS DE MITIGAÇÃO 49
4.1 Mitigação de ataques *Man-in-the-Middle (MitM)* 49
4.2 Mitigação de ataques *SQL injection* 50
4.3 Mitigação de ataques de força bruta 52
5 IMPLEMENTAÇÃO DE UM SISTEMA DE LOGIN SEGURO 54
5.1 Ferramentas utilizadas 54
5.2 Conexão com banco de dados e gestão de usuários 54
5.3 Prevenção contra ataques *Man-in-the-Middle (MitM)* 56
5.4 Prevenção contra ataques *SQL injection* 56
5.4.1 Comparação entre o site vulnerável e sistema de login seguro 59

5.5 Prevenção contra ataques de força bruta.....61
6 CONSIDERAÇÕES FINAIS.....65
7 REFERÊNCIAS 67

1 INTRODUÇÃO

No cenário atual da tecnologia da informação, a segurança cibernética emergiu como uma prioridade crítica para organizações de todos os tamanhos. A proliferação de ataques cibernéticos sofisticados, direcionados a servidores web, expôs a vulnerabilidade dos sistemas digitais e a necessidade urgente de desenvolver e implementar medidas de segurança eficazes.

Uma vulnerabilidade é definida como uma condição que, quando explorada por um atacante, pode resultar em uma violação de segurança. Exemplos de vulnerabilidades incluem falhas no projeto, na implementação ou na configuração de programas, serviços ou equipamentos de rede. A exploração dessas vulnerabilidades por atacantes permite a execução de ações maliciosas, como invasão de sistemas, acesso a informações confidenciais, lançamento de ataques contra outros computadores ou até mesmo a interrupção de serviços (CERT.br, 2012).

As ameaças à segurança na Web podem ser classificadas em ataques passivos e ativos. Ataques passivos envolvem a interceptação de tráfego sem alterar os dados, enquanto ataques ativos podem incluir a modificação de informações em trânsito ou a simulação de usuários legítimos. Além disso, as ameaças podem ser segmentadas por alvo, como servidores Web, navegadores e o tráfego de rede entre eles (Stallings, 2008).

Um exemplo notável de ataque cibernético é o caso do site de comércio eletrônico Netshoes que, em 2018, sofreu um vazamento de dados que afetou quase dois milhões de clientes. O incidente expôs informações pessoais, como nome, CPF, e-mail, data de nascimento e histórico de compras dos clientes. Consequentemente, a Netshoes foi condenada a pagar R$ 500 mil em indenização por danos morais. Embora dados sensíveis, como informações de cartões de crédito ou senhas, não tenham sido comprometidos, o vazamento deixou os clientes suscetíveis a diversas formas de fraude, ressaltando a importância de implementar medidas eficazes de segurança cibernética (Stefanello, 2023).

Diante disso, um aspecto crucial na segurança de um site é o uso de protocolos de comunicação seguros. O *Hypertext Transfer Protocol (HTTP)* é um protocolo responsável por enviar informações variadas, como textos e mídias, entre clientes e servidores. Esse processo se baseia em um modelo de requisição e resposta: o cliente

inicia a comunicação, envia uma solicitação ao servidor e aguarda sua resposta (Berners-Lee et al., 1996). Contudo, o HTTP não oferece criptografia, tornando as comunicações vulneráveis a uma variedade de ataques cibernéticos. Entre os ataques mais comuns que exploram vulnerabilidades do HTTP estão *SQL injection*, *Man-in-the-Middle (MitM)* e ataques de força bruta. Esses tipos de ataques podem comprometer a integridade, confidencialidade e disponibilidade dos dados transmitidos, destacando a importância de adotar medidas de segurança robustas, como a transição para o protocolo HTTPS, que oferece uma camada adicional de proteção por meio da criptografia (CERT.br, 2012).

1.1 Justificativa

O estudo do tema em questão é relevante, pois compreender as vulnerabilidades associadas ao HTTP e implementar medidas de segurança eficazes são passos cruciais para proteger os servidores web e garantir a integridade, confidencialidade e disponibilidade das informações transmitidas (Stallings, 2008).

De acordo com a Cartilha de Segurança para Internet (CERT.br, 2012), usuários e sistemas estão expostos a diversos riscos, como furto de dados, invasão de privacidade e exploração de vulnerabilidades. A cartilha alerta que muitos acreditam estar a salvo de ataques, mas, na realidade, criminosos realizam varreduras para identificar falhas em sistemas conectados à Internet, incluindo servidores web. Essa falsa sensação de segurança é um dos maiores desafios na proteção contra ataques cibernéticos.

Com a crescente sofisticação dos ataques cibernéticos e a dependência cada vez maior de sistemas digitais, torna-se imprescindível o desenvolvimento contínuo de estratégias de segurança robustas e adaptáveis às novas ameaças.

Diante deste contexto, este trabalho visa responder a seguinte questão: Como identificar vulnerabilidades e mitigar os ataques cibernéticos em servidores web?

1.2 Objetivos

1.2.1 Objetivo geral

O objetivo geral deste trabalho é analisar detalhadamente as vulnerabilidades de segurança associadas aos servidores web, com ênfase nos protocolos de comunicação HTTP e HTTPS, e investigar estratégias de criptografia e outras medidas de segurança eficazes para mitigar os riscos de ataques cibernéticos, como *SQL injection*, *Man-in-the-Middle (MitM)* e ataques de força bruta.

1.2.2 Objetivos específicos

- Analisar o funcionamento dos protocolos HTTP e HTTPS, destacando suas diferenças e implicações para a segurança.
- Identificar e descrever as principais vulnerabilidades de segurança presentes em servidores web.
- Detalhar e simular os métodos de ataque cibernético *Man-in-the-Middle (MitM)*, *SQL injection* e ataques de força bruta.
- Fornecer diretrizes de segurança cibernética destinadas a reduzir os riscos inerentes aos tipos de ataques cibernéticos investigados, destacando medidas preventivas e estratégias de resposta eficazes.
- Desenvolver e implementar um site de *login* simples, aplicando as medidas de segurança propostas, com o objetivo de demonstrar, na prática, a eficácia das soluções sugeridas na proteção contra os ataques abordados.

1.3 Procedimentos metodológicos

Conforme sua natureza, esta pesquisa apresenta-se como uma síntese que visa explanar a área de conhecimento abordada no projeto e ressaltar sua evolução ao longo do tempo. Por meio da análise das informações coletadas, procura-se compreender as causas e fatores explicativos relacionados ao tema estudado (WAZLAWICK, 2014).

Quanto aos seus objetivos, trata-se de uma pesquisa exploratória e descritiva. A abordagem exploratória busca examinar fenômenos específicos com a finalidade

de identificar possíveis anomalias que possam servir de base para estudos mais detalhados e aprofundados. Já a pesquisa descritiva tem como propósito coletar dados mais sólidos sobre uma determinada realidade, frequentemente por meio de levantamentos de informações (WAZLAWICK, 2014).

Com relação aos procedimentos técnicos, esta pesquisa é caracterizada como bibliográfica e experimental. A pesquisa bibliográfica envolve a consulta e análise de diferentes fontes, incluindo artigos e dissertações acadêmicas. Por sua vez, a pesquisa experimental distingue-se pela manipulação de uma ou mais variáveis experimentais, permitindo ao pesquisador observar seus efeitos (WAZLAWICK, 2014).

1.4 Estrutura da monografia

O texto dessa monografia está organizado em 6 capítulos. No capítulo 1 é abordado a importância da segurança cibernética no contexto atual da tecnologia da informação, destacando a vulnerabilidade dos sistemas digitais diante de ataques sofisticados. O capítulo 2 abrange o embasamento teórico do trabalho, contendo conceitos e definições a respeito dos protocolos HTTP e HTTPS, e dos ataques cibernéticos *Man-in-the-Middle (MitM)*, *SQL injection* e força bruta. No capítulo 3 são apresentadas simulações dos ataques investigados, demonstrando o funcionamento e expondo vulnerabilidades. No capítulo 4 são mostradas estratégias de mitigação para cada um dos ataques simulados no capítulo anterior. No capítulo 5, é apresentada uma implementação de um sistema de *login* seguro, demonstrando a defesa contra os ataques discutidos ao longo do trabalho. Por fim, no capítulo 6, são apresentadas as considerações finais, destacando a relevância dos resultados obtidos.

2 REFERENCIAL TEÓRICO

Este capítulo visa fornecer uma base teórica sólida sobre os principais protocolos de comunicação utilizados na web, como o HTTP e o HTTPS, além de explorar vulnerabilidades comuns que afetam servidores web e sistemas de autenticação. A compreensão profunda desses protocolos e suas fraquezas é essencial para a implementação de medidas de segurança adequadas, especialmente para proteger contra ataques cibernéticos, como *SQL injection*, *Man-in-the-Middle (MitM)* e ataques de força bruta. A seguir, são apresentados os tópicos fundamentais para entender os conceitos de segurança em servidores web, que servirão de embasamento para as análises e soluções propostas neste trabalho.

2.1 Protocolo HTTP

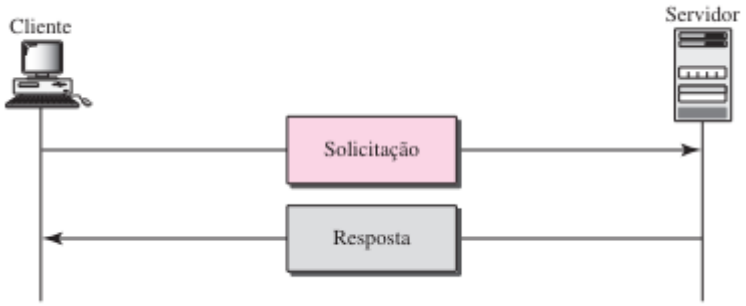
O HTTP, ou *Hypertext Transfer Protocol*, é um protocolo de comunicação que permite a transferência de dados na web. É um protocolo baseado em texto e sem estado, o que significa que cada solicitação do cliente ao servidor é independente das anteriores (Kurose; Ross, 2014).

2.1.1 Funcionamento do HTTP

O funcionamento do HTTP pode ser descrito em termos de requisições e respostas (Figura 1):

- Requisições HTTP: Enviadas pelo cliente (geralmente um navegador web) ao servidor. Uma requisição HTTP típica inclui um método (como *GET* ou *POST*), um URI (*Uniform Resource Identifier*), a versão do protocolo e cabeçalhos adicionais (Kurose; Ross, 2014).
- Respostas HTTP: Enviadas pelo servidor em resposta à requisição do cliente. Uma resposta HTTP inclui um código de status (indicando sucesso, redirecionamento ou erro), a versão do protocolo, cabeçalhos e o corpo da mensagem contendo os dados solicitados (Kurose; Ross, 2014).

Figura 1: Transação HTTP



Fonte: Forouzan (2010)

2.1.2 Métodos HTTP

Os métodos HTTP definem a ação que o cliente deseja que o servidor execute. De acordo com Forouzan (2010), os métodos são:

- *GET*: Solicita um documento ao servidor.
- *HEAD*: Solicita informação sobre um documento, mas não o documento em si.
- *POST*: Envia informações do cliente para o servidor.
- *PUT*: Envia um documento do servidor para o cliente.
- *TRACE*: Ecoa uma solicitação que chega.
- *CONNECT*: Reservado.
- *OPTION*: Solicita detalhamento sobre opções disponíveis.

2.1.3 Vulnerabilidades do HTTP

O HTTP, por ser um protocolo de texto simples, é suscetível a várias vulnerabilidades de segurança. De acordo com Quora (2023), inclui:

- Intercepção de dados: Dados trafegados em HTTP são transmitidos em texto claro, permitindo que atacantes interceptem e leiam as informações.
- Ataques *Man-in-the-Middle (MitM)*: Atacantes podem interceptar e modificar as comunicações entre cliente e servidor.
- Falsificação de Requisição entre Sites (CSRF): Atacantes podem induzir um usuário autenticado a executar ações não desejadas em um site.

- Ataques de *Cross-Site Scripting (XSS):* Neste tipo de ataque, um invasor injeta código malicioso em um site, que pode roubar informações confidenciais do navegador do usuário ou realizar ações não autorizadas em nome do usuário.

2.2 Protocolo HTTPS

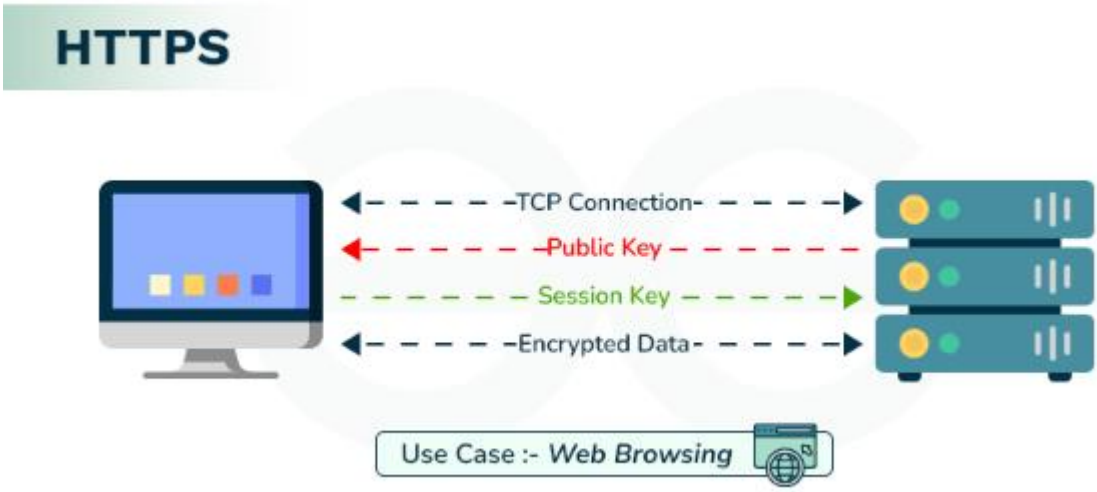
O HTTPS, ou *Hypertext Transfer Protocol Secure*, é uma extensão do HTTP que adiciona uma camada de segurança utilizando o protocolo *TLS (Transport Layer Security)* (Kurosee; Ross, 2014).

2.2.1 Funcionamento do HTTPS

O HTTPS funciona da seguinte maneira (Figura 2):

- Estabelecimento de Conexão Segura: Antes da troca de dados, cliente e servidor realizam um *handshake* TLS para estabelecer uma conexão segura. Durante esse processo, são utilizados certificados digitais para autenticação e a negociação de chaves de criptografia (Stallings, 2008).
- Transferência de Dados Criptografados: Após o *handshake*, todos os dados transmitidos são criptografados, garantindo confidencialidade e integridade (Stallings, 2008).

Figura 2: Transação HTTPS



Fonte: GeeksforGeeks (2024)

2.2.2 TLS/SSL: Transport Layer Security e Secure Sockets Layer

De acordo com Stallings (2008), o SSL e seu sucessor, o TLS, são protocolos que proporcionam segurança nas comunicações sobre redes de computadores. Eles funcionam na camada de transporte e utilizam criptografia assimétrica para garantir que apenas o destinatário pretendido possa decifrar a mensagem. O processo envolve:

Handshake SSL/TLS: Um processo inicial onde o cliente e o servidor concordam sobre os parâmetros de criptografia a serem usados. Inclui a autenticação do servidor e, opcionalmente, a autenticação do cliente (Stallings, 2008).

Criptografia de Sessão: Após o *handshake*, uma chave de sessão simétrica é criada e usada para criptografar os dados trocados durante a sessão. A criptografia simétrica é usada por ser mais eficiente em termos de desempenho (Stallings, 2008).

2.2.3 Vantagens do HTTPS

As principais vantagens do HTTPS incluem:

- Confidencialidade: Dados são criptografados, protegendo-os contra interceptação (Stallings, 2008).
- Integridade: Garante que os dados não sejam alterados durante a transmissão (Stallings, 2008).
- Autenticação: Certificados digitais verificam a identidade do servidor, evitando ataques de phishing e MitM (Stallings, 2008).

2.2.4 Certificados digitais

Certificados digitais são fundamentais para o funcionamento do HTTPS. Eles são emitidos por autoridades certificadoras (CAs) e contêm a chave pública do servidor, informações sobre a CA emissora e a assinatura digital da CA (Stallings, 2008).

2.2.5 Implementação do HTTPS

A implementação de HTTPS envolve:

- Obtenção de um Certificado Digital: Adquirido de uma CA confiável (Stallings, 2008).
- Configuração do Servidor: O servidor web deve ser configurado para utilizar o certificado digital e suportar conexões TLS (Stallings, 2008).
- Redirecionamento de Tráfego: Configurar redirecionamento de todo o tráfego HTTP para HTTPS (Stallings, 2008).

2.2.6 Comparação entre HTTP e HTTPS

Figura 3: Diferenças Técnicas

Característica	HTTP	HTTPS
Segurança	Não seguro	Seguro
Criptografia	Não	TLS/SSL
Porta Padrão	80	443
Certificados	Não utilizados	Utiliza certificados digitais
Confidencialidade	Dados em texto claro	Dados criptografados
Integridade	Sem garantias	Proteção contra alterações

Fonte: Autoria própria (2024)

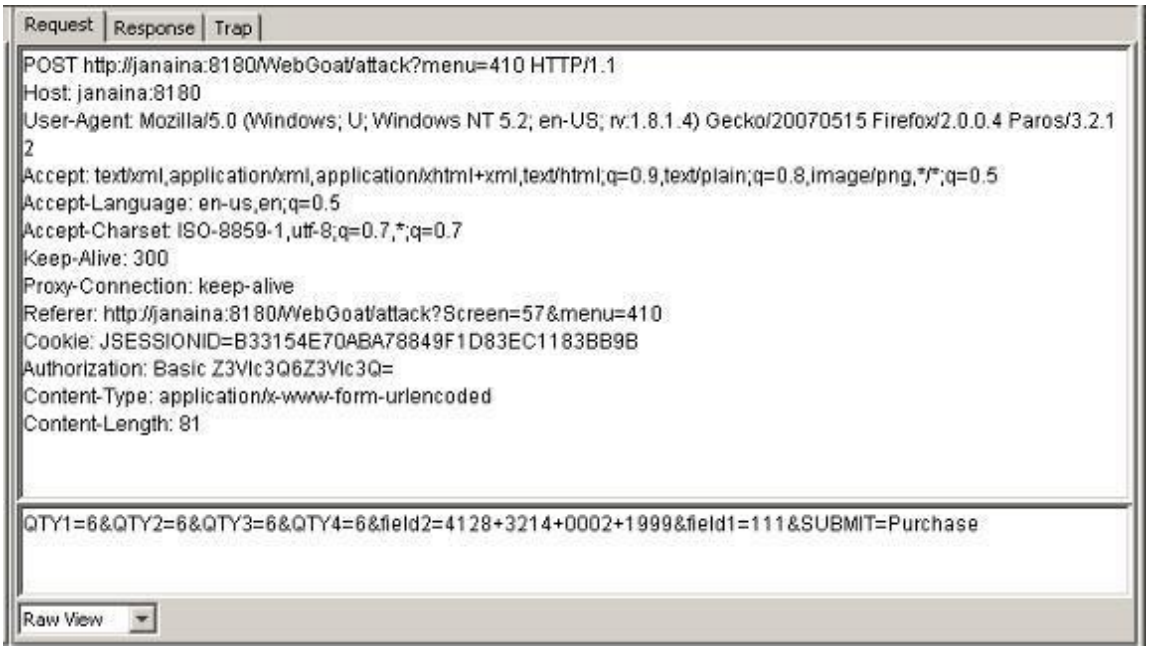
2.3 Man-in-the-Middle (MitM)

O ataque *Man-in-the-Middle (MitM)* é um tipo de ataque cibernético em que um terceiro malicioso assume o controle de uma comunicação confidencial entre duas partes, permitindo que o atacante leia, modifique ou substitua o tráfego de comunicação entre as vítimas. Ao usar o protocolo MitM, o atacante pode permanecer invisível e sem deixar rastros de sua interceptação. (Ahsan et al., 2019).

Segundo Ahsan et al., (2019), o ataque MitM ocorre em duas etapas: interceptação e decodificação. Na interceptação, o atacante pode se aproveitar de redes Wi-Fi desprotegidas ou comprometer o roteador para inserir suas ferramentas entre o dispositivo da vítima e os sites visitados. Uma variação desse ataque é o *Man-in-the-Browser (MitB)*, que envolve a instalação silenciosa de *malware* no navegador da vítima para capturar dados sensíveis. Na etapa de decodificação, o atacante pode usar técnicas como falsificação de certificados HTTPS, exploração de vulnerabilidades SSL/TLS ou sequestro de chaves de autenticação para ler o tráfego SSL sem ser detectado. Esses ataques comprometem a segurança da comunicação online, permitindo ao atacante capturar informações confidenciais, como credenciais de *login* e transações financeiras.

Por exemplo, em um ataque, a comunicação entre sistemas é interceptada durante transações HTTP, manipulando a conexão TCP entre cliente e servidor. O atacante cria duas novas conexões: uma entre cliente e atacante e outra entre atacante e servidor, mostrado na Figura 4. Agindo como um *proxy*, o atacante pode ler, inserir e modificar dados na comunicação interceptada. Isso é possível devido à natureza ASCII do protocolo HTTP, permitindo a visualização e interferência nos dados transferidos, como capturar *cookies* de sessão ou alterar transações monetárias. (OWASP, 2023).

Figura 4: Pacote HTTP interceptado com Paros Proxy



Fonte: Owasp (2023)

Embora o ataque MITM também possa ocorrer em conexões HTTPS, ele requer o estabelecimento de duas sessões SSL independentes. O navegador estabelece uma conexão SSL com o invasor e o invasor estabelece outra conexão SSL com o servidor web. O navegador pode alertar o usuário sobre a validade do certificado digital, mas esse aviso pode ser ignorado. Em alguns casos, o invasor pode comprometer o certificado do servidor ou usar um certificado assinado por uma CA confiável (OWASP, 2023).

2.4 SQL injection

Um ataque de injeção SQL envolve a inserção ou "injeção" de consultas SQL através dos dados de entrada fornecidos pelo cliente para a aplicação. Se bem-sucedida, essa exploração pode permitir a leitura de dados confidenciais do banco de dados, a modificação dos dados (inserção/atualização/exclusão), a execução de operações de administração no banco de dados (como desligar o SGBD), a recuperação de conteúdo de arquivos específicos no sistema SGBD e, em alguns casos, a emissão de comandos para o sistema operacional (OWASP, 2023).

Na ausência de medidas de proteção adequadas, a injeção de SQL pode expor o aplicativo a um alto risco de comprometimento, afetando a confidencialidade e a integridade dos dados, além dos aspectos de autenticação e autorização. Isso significa que um atacante pode acessar informações sensíveis armazenadas em bancos de dados de programas ou aplicativos vulneráveis, como credenciais de usuário, segredos comerciais ou registros de transações (Packetlabs, 2023).

Existem diferentes maneiras de explorar a injeção de SQL. Os três tipos mais comuns de injeção de SQL são: baseadas em Union, baseadas em Erro e injeção de *SQL Blind*.

2.4.1 Injeção de SQL baseada em *Union*

Este método, ilustrado na Figura 5, utiliza a palavra-chave "*union*" para combinar os resultados da consulta original com os resultados de consultas maliciosas injetadas em uma única resposta HTTP. (Daityari, 2023).

Figura 5: Injeção de SQL baseada em Union

```
select title, link from post_table
where id < 10
union
select username, password
from user_table; --;
```

Fonte: Kinsta (2023)

2.4.2 Injeção de SQL baseada em Erro

Esta técnica é geralmente fácil de explorar, pois fornece mensagens de erro detalhadas quando uma consulta SQL falha, como por exemplo, devido a erros de sintaxe. Um invasor pode usar essas informações para descobrir a consulta exata executada pelo servidor SQL e criar solicitações maliciosas. Dependendo das configurações do banco de dados, isso pode ser usado para inserir ou modificar dados, além de recuperar informações (Packetlabs, 2023). Por exemplo, um invasor pode inserir 'OR '1'='1 em um campo de formulário. Se o aplicativo for vulnerável, ele poderá

retornar uma mensagem de erro que revela informações sobre o banco de dados (Daityari, 2023).

2.4.3 Injeção de *SQL Blind*

Esse tipo de vulnerabilidade requer que o invasor utilize técnicas indiretas, como consultas baseadas em booleanos ou baseadas em tempo, para extrair informações do banco de dados através de uma série de declarações lógicas complexas (Packetlabs, 2023).

No ataque booleano de injeção SQL, um invasor manipula as entradas do usuário para tentar duas versões diferentes de uma cláusula booleana na consulta SQL:

"… e 1=1"
"… e 1=2"

Essas consultas são projetadas para gerar uma página carregada normalmente se a condição for verdadeira e uma página diferente ou com erro se for falsa. Ao analisar como a página é carregada, o invasor pode determinar o sucesso da manipulação, extraindo lentamente informações do banco de dados sem ver a consulta real ou a resposta do banco de dados (Daityari, 2023).

No ataque de injeção SQL baseado em tempo, um invasor tenta determinar se há uma vulnerabilidade em um aplicativo web usando uma função de espera do sistema de gerenciamento de banco de dados, como o *MySQL's sleep()*, como visto na Figura 6. Se essa consulta resultar em atraso, o invasor sabe que o aplicativo é vulnerável. (Daityari, 2023).

Figura 6: Injeção de SQL baseada em tempo

```
select * from comments
WHERE post_id=1-SLEEP(15);
```

Fonte: Kinsta (2023)

De acordo com Eco Trust, (2023), é possível identificar vulnerabilidades de *SQL Injection* através de scanners de vulnerabilidades, porém pode-se detectar manualmente por meio de testes em cada ponto de entrada da aplicação, como:

- Enviar um caractere de aspas simples (') e procurar por erros ou comportamentos anômalos.
- Inserir sintaxes específicas de *SQL injection* e comparar as respostas da aplicação para valores diferentes.
- Utilizar condições booleanas, como OU 1=1 e OU 1=2, e observar diferenças nas respostas.
- Enviar cargas projetadas para causar atrasos de tempo na execução de uma consulta SQL e medir a variação no tempo de resposta.
- Utilizar cargas OAST para provocar interações de rede fora de banda e monitorar quaisquer respostas resultantes.

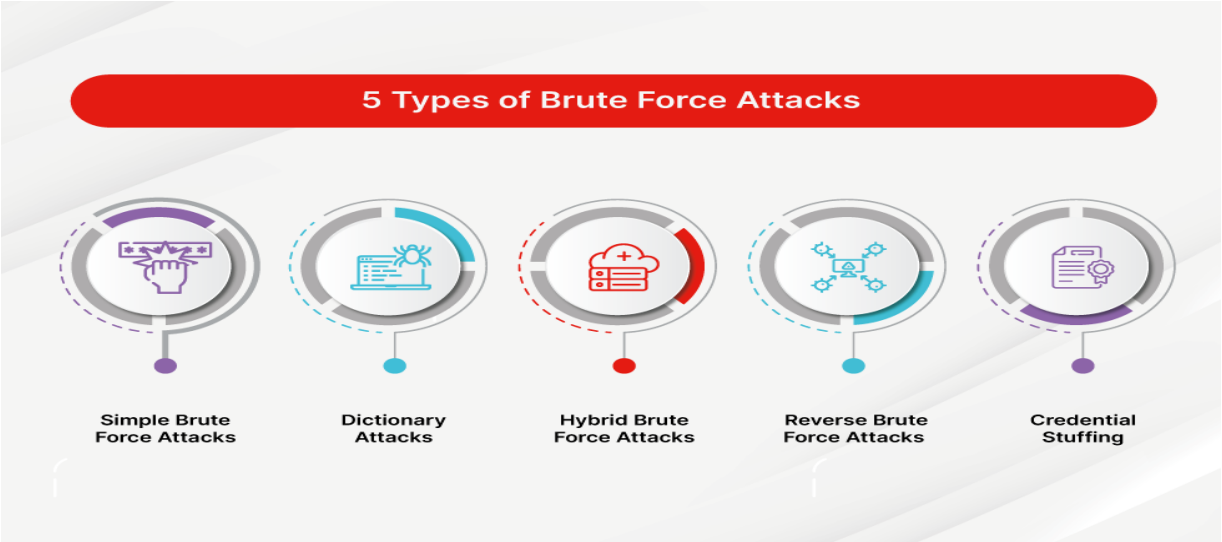
2.5 Força bruta

Um ataque de força bruta, envolve tentar adivinhar um nome de usuário e uma senha por meio de repetidas tentativas e erros. Esse método permite que o atacante consiga acessar sites, computadores e serviços, executando processos com as mesmas permissões do usuário legítimo (CERT.br, 2012).

Qualquer dispositivo, rede ou serviço acessível via Internet que utilize autenticação por nome de usuário e senha pode ser vulnerável a ataques de força bruta. Dispositivos móveis, mesmo protegidos por senha, podem ser comprometidos tanto por ataques remotos quanto por acesso físico direto, caso o invasor tenha acesso ao dispositivo (CERT.br, 2012).

Há uma variedade de métodos de ataque de força bruta que capacitam invasores a acessar sistemas de forma não autorizada e roubar dados de usuários, vistos na Figura 7.

Figura 7: 5 tipos de ataques de força bruta



Fonte: Fortinet (2023)

De acordo com CERT.br (2012), essas são ações maliciosas que podem ser efetuadas quando o invasor tem acesso ao usuário e a senha da vítima:

- Alterar sua senha, dificultando seu acesso futuro ao site ou computador comprometido;
- Invadir seu serviço de e-mail, permitindo o acesso ao conteúdo de suas mensagens e à sua lista de contatos, além de enviar mensagens em seu nome;
- Acessar suas redes sociais e enviar mensagens para seus seguidores contendo códigos maliciosos ou alterar suas configurações de privacidade;
- Invadir seu computador e, conforme as permissões do seu usuário, executar ações como apagar arquivos, obter informações confidenciais e instalar códigos maliciosos.

2.5.1 Ataques de força bruta simples

O invasor tenta manualmente adivinhar credenciais de *login*, explorando senhas fracas ou comuns, como "senha123" ou "1234". Esses ataques são facilitados pelo uso generalizado de senhas simples e previsíveis (FORTINET, 2023).

2.5.2 Ataques de dicionário

Utilizam listas de palavras comuns e variam caracteres e números, testando essas combinações contra nomes de usuário para encontrar a senha correta. O nome "ataques de dicionário" vem do uso de listas de palavras comuns, que são testadas sistematicamente (FORTINET, 2023).

2.5.3 Ataques de força bruta híbrida

Combinam ataques de dicionário com força bruta simples, tentando variações de palavras comuns e números para descobrir senhas como "SanDiego123". Essa abordagem aumenta a eficácia ao misturar palavras comuns com números e caracteres (FORTINET, 2023).

2.5.4 Ataques de força bruta reversa

Começam com uma senha conhecida, frequentemente obtida de uma violação de dados, e tentam encontrar o nome de usuário correspondente a essa senha em um grande banco de dados. Essa técnica pode ser usada para explorar senhas comuns, como "Senha123", contra muitos nomes de usuário diferentes (FORTINET, 2023).

2.5.5 Preenchimento de credenciais

Aproveitam-se de credenciais roubadas testando-as em múltiplos sites, explorando o hábito dos usuários de reutilizarem senhas. Se uma combinação de nome de usuário e senha for reutilizada em vários sites, um ataque bem-sucedido pode comprometer múltiplas contas (FORTINET, 2023).

3 SIMULAÇÕES DOS ATAQUES CIBERNÉTICOS

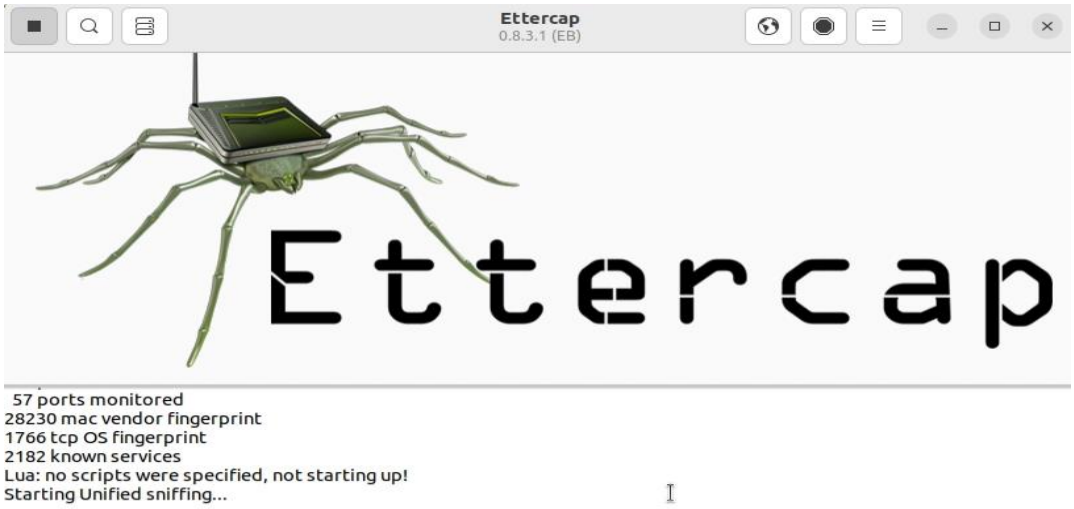
Este capítulo tem como objetivo realizar simulações práticas de três ataques cibernéticos amplamente conhecidos: *Man-in-the-Middle (MitM)*, *SQL injection* e ataques de força bruta. Através dessas simulações, será possível observar o funcionamento de cada ataque em um ambiente controlado, identificando suas consequências e as falhas de segurança que podem ser exploradas. A análise prática desses ataques contribui para a compreensão de suas dinâmicas e permite a proposição de estratégias de mitigação mais eficazes, evidenciando a importância de medidas preventivas para proteger sistemas contra tais ameaças.

3.1 Simulação *Man-in-the-Middle (MitM)*

A seguir, será realizada uma simulação de ataque *Man-in-the-Middle (MitM)* utilizando as ferramentas Ettercap, Wireshark, e o site Acunetix Art.

Ettercap: É uma ferramenta de análise de protocolos e interceptação de rede de código aberto que suporta ativos de interceptação e ataques *Man-in-the-Middle*. O Ettercap (Figura 8), é compatível com sistemas operacionais baseados em Linux como o Debian/Ubuntu, Fedora, FreeBSD e macOS (Controle Net, 2023).

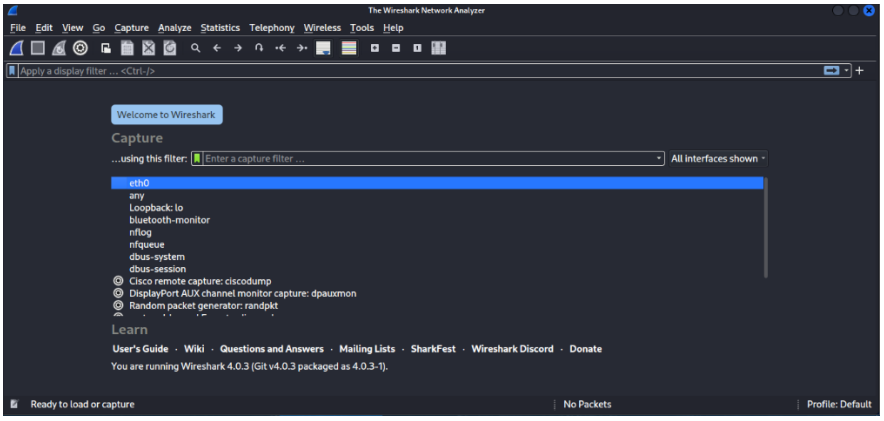
Figura 8: Ferramenta Ettercap



Fonte: Ettercap (2024)

Wireshark: é uma ferramenta de análise de pacotes e *sniffer* de rede. O Wireshark (Figura 9), captura o tráfego de rede na rede local e armazena esses dados para análise off-line (Buckbee, 2022).

Figura 9: Ferramenta Wireshark



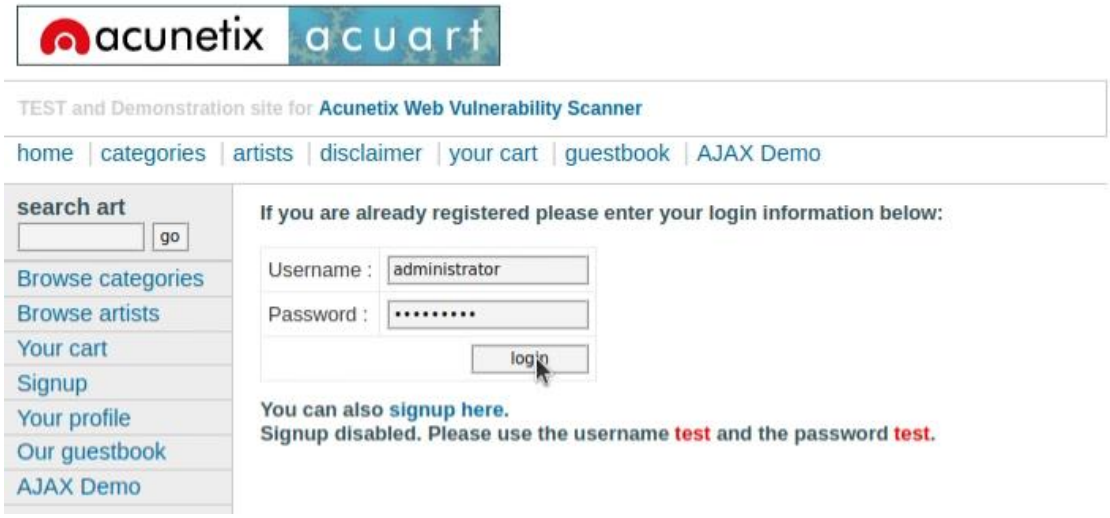
Fonte: Bladimir Peláez (2024)

Acunetix Art: É uma aplicação web de exemplo desenvolvida em PHP intencionalmente vulnerável, utilizada para fins educacionais e de simulação de ataques.

3.1.1 Tentativa de *login* no servidor alvo

Na Figura 10, um usuário tenta fazer *login* na página “testphp.vulnweb.com/login.php” usando o nome de usuário “administrator” e a senha “senha123$”. Esta tentativa de *login* é parte do processo de captura de credenciais.

Figura 10: Tentativa de *login*



Fonte: Acunetix Art (2024)

3.1.2 Configuração e execução do Ettercap

Na Figura 11, a interface gráfica do Ettercap é usada para iniciar o ataque MitM na rede. O Ettercap captura o tráfego de rede entre a vítima e o servidor. O log do Ettercap mostra que ele interceptou a tentativa de *login*, exibindo o nome de usuário “administrator” e a senha “senha123$”.

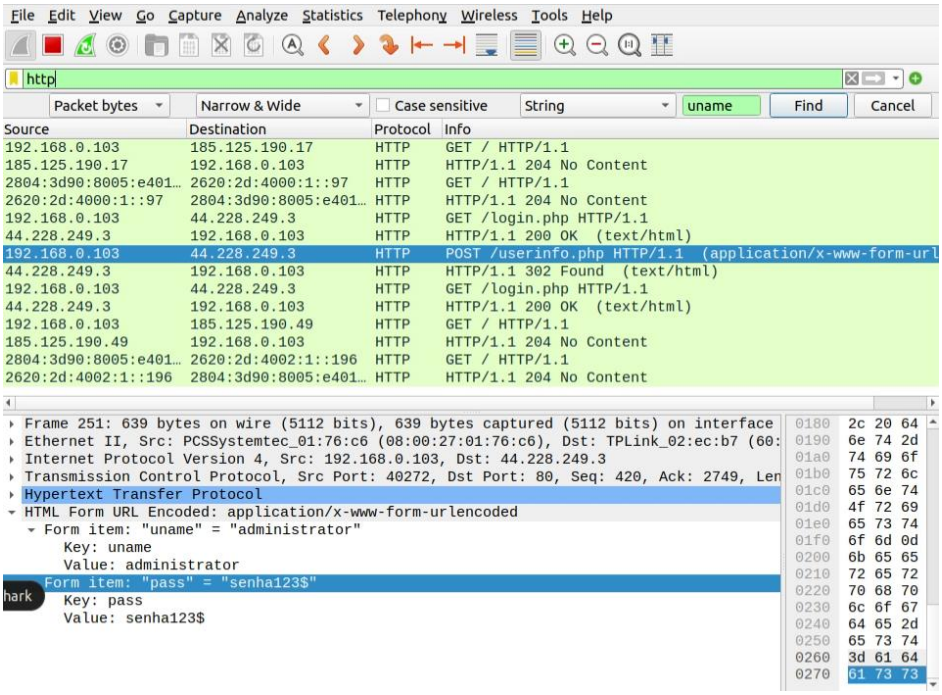
Figura 11: Execução do Ettercap

Fonte: Ettercap (2024)

3.1.3 Análise de tráfego com o Wireshark

Na Figura 12, o Wireshark é usado para analisar o tráfego de rede capturado pelo Ettercap. A imagem mostra o pacote HTTP POST contendo as credenciais de *login* enviadas para “userinfo.php”. Os campos “uname” e “pass” contêm o nome de usuário “administrator” e a senha “senha123$”, respectivamente.

Figura 12: Análise de tráfego com o Wireshark



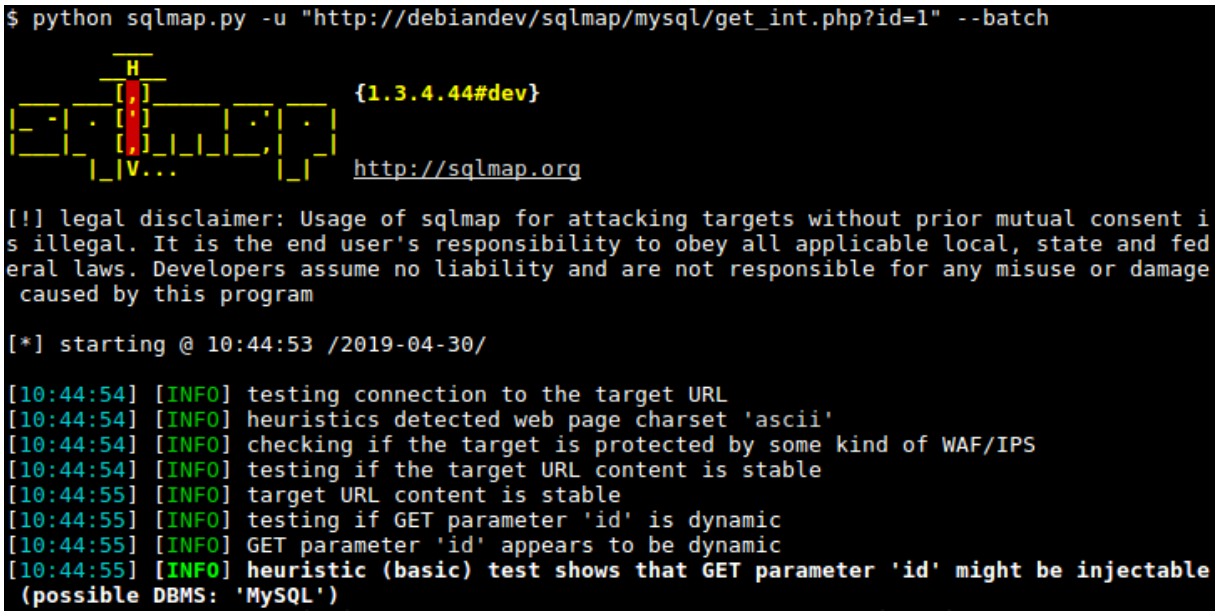
Fonte: Wireshark (2024)

3.2 Simulação *SQL injection*

Nesta simulação, será realizado um processo de *SQL injection* usando duas ferramentas principais: SQLMap e o site Acunetix Art.

SQLMap: É uma ferramenta de teste de invasão (de código aberto) que automatiza a detecção e exploração de falhas de injeção SQL em sistemas de banco de dados (Figura 13). Ela pode ajudar a enumerar alvos, realizar a identificação de bancos de dados, ler e escrever em sistemas de arquivos remotos e quebrar senhas. (IBSEC, 2023).

Figura 13: Ferramenta SQLMap



Fonte: sqlmap.org (2024)

3.2.1 Indicação de vulnerabilidade

A Figura 14 mostra uma página do site Acunetix Art, que contém “informações” sobre um dos artistas fictícios do site, cuja URL é “http://testphp.vulnweb.com/artists.php?artist=1”.

Figura 14: Site Acunetix Art

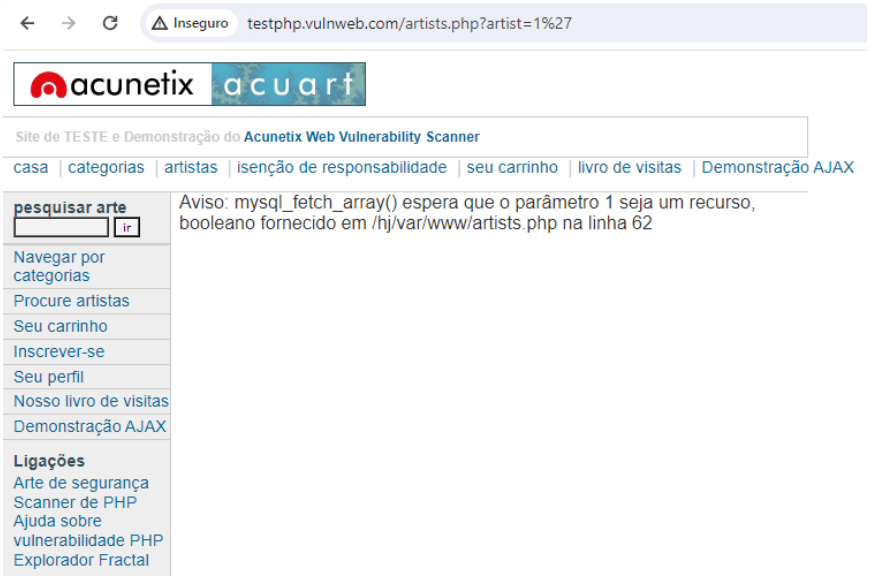
Fonte: Acunetix Art (2024)

Na Figura 15, com o objetivo de explorar vulnerabilidades, a URL é alterada para “http://testphp.vulnweb.com/artists.php?artist=1%27”, contendo o parâmetro “*artist*” definido como ‘1%27’. O ‘%27’ representa o *encoding* para o caractere (') aspas simples, comumente utilizado em tentativas de *SQL injection*.

Essa modificação resulta na exibição da seguinte mensagem de erro: "Aviso: mysql_fetch_array() espera que o parâmetro 1 seja um recurso, booleano fornecido em /hj/var/www/artists.php na linha 62". Esse erro indica que a função “mysql_fetch_array()” está recebendo um valor booleano em vez de um recurso válido, o que sugere falha na execução da consulta SQL, provavelmente devido à injeção de um caractere (') não esperado.

A presença desse erro visível evidencia um potencial para *SQL injection*, indicando que a aplicação não está tratando adequadamente as entradas fornecidas pelos usuários. Essa vulnerabilidade pode ser explorada por um atacante para injetar código SQL malicioso. O problema ocorre porque a aplicação passa diretamente o valor do parâmetro à consulta SQL, sem sanitização ou uso de *prepared statements*.

Figura 15: Vulnerabilidade do site Acunetix Art



Fonte: Acunetix Art (2024)

3.2.2 Exploração de vulnerabilidades utilizando SQLMap

A Figura 16 apresenta a execução inicial do SQLMap, onde o comando é executado com a URL alvo “http://testphp.vulnweb.com/artists.php?artist=1” e a opção --dbs, que instrui o SQLMap a enumerar os bancos de dados disponíveis no servidor. Durante o processo, o SQLMap realiza a conexão inicial com o banco de dados MySQL indicado pela URL fornecida. A mensagem "*[INFO] resuming back-end DBMS 'mysql'*" confirma que o SQLMap identificou o MySQL como o sistema de gerenciamento de banco de dados (DBMS) do *back-end*.

Figura 16: Conexão e testes iniciais usando SQLMap

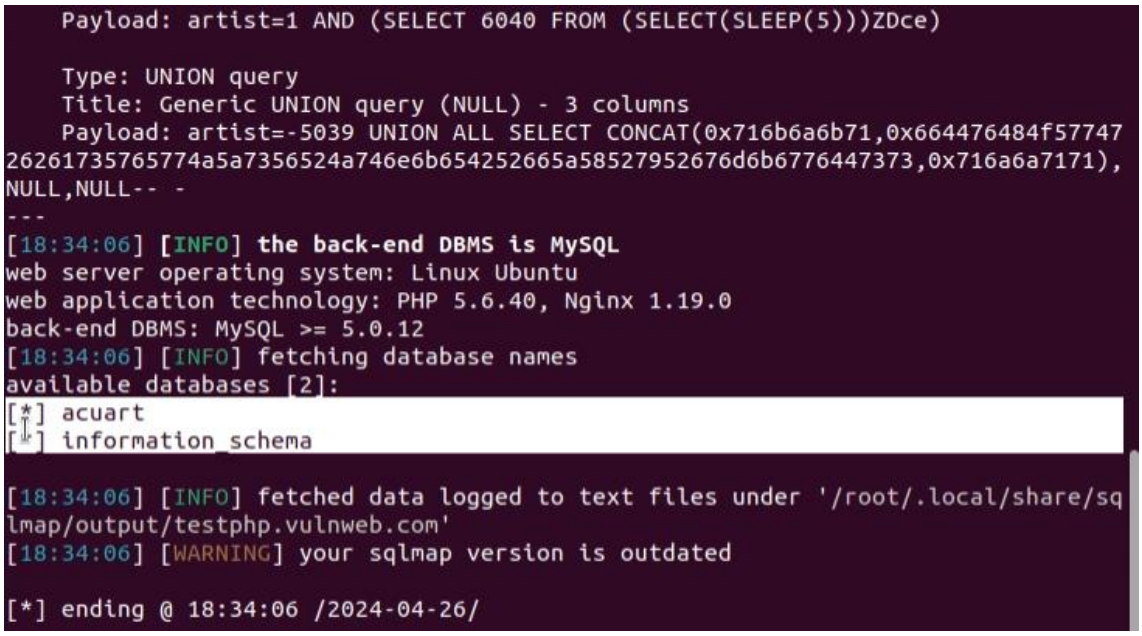
Fonte: Autoria própria (2024)

A detecção e exploração da vulnerabilidade pode ser observada na Figura 17. O SQLMap utiliza vários *payloads* de injeção SQL para explorar a URL alvo e determinar a existência de vulnerabilidades, culminando na identificação do banco de dados:

- Tipo de Consulta: *UNION query*
- Informações Coletadas: O SQLMap detecta o sistema de gerenciamento de banco de dados (SGBD) como MySQL, e o sistema operacional do servidor web como Linux Ubuntu, com PHP 5.6.40 e Nginx 1.19.0.
- Mensagem: “*[INFO] fetching database names*” indica que o SQLMap está listando os nomes dos bancos de dados disponíveis.

A exploração também resulta na listagem dos bancos de dados encontrados: “acuart” e “information_schema”.

Figura 17: Banco de dados encontrados usando SQLMap



Fonte: Autoria própria (2024)

3.2.3 Extração de dados

Na Figura 18, é mostrada a execução do comando “sqlmap -u http://testphp.vulnweb.com/artists.php?artist=1 -D acuart –tables”, utilizado para acessar as tabelas do banco de dados acuart:

- A opção -u especifica a URL alvo, que neste caso é http://testphp.vulnweb.com/artists.php?artist=1.
- A opção -D seguida por acuart indica que queremos listar as tabelas do banco de dados acuart.
- A opção --tables instrui o SQLMap a listar todas as tabelas disponíveis no banco de dados especificado.

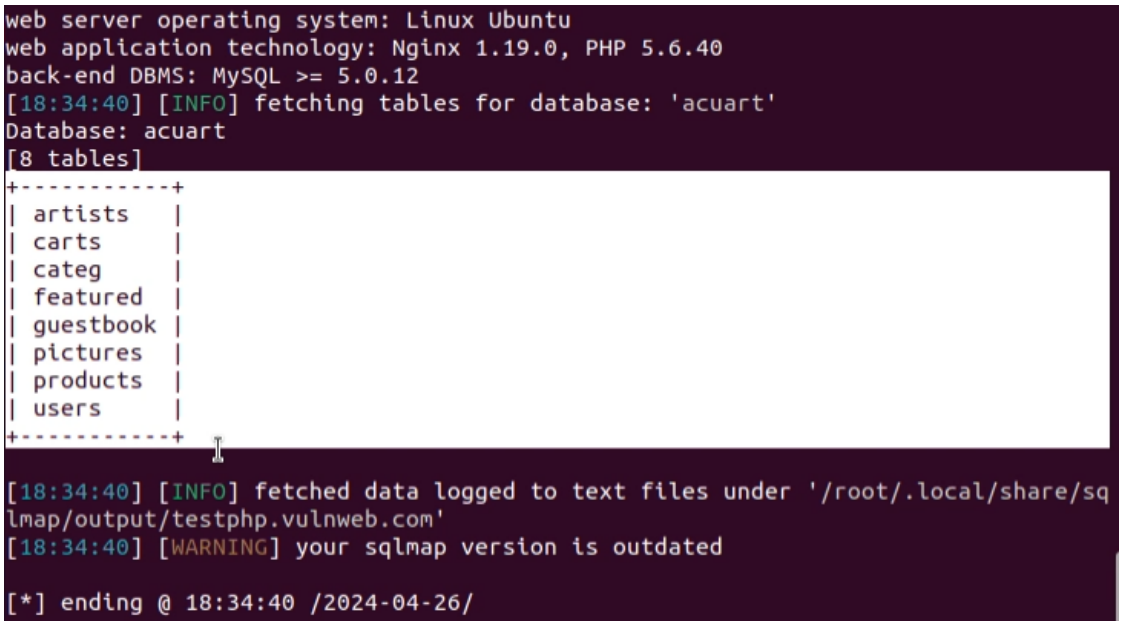
Figura 18: Acesso às tabelas do banco de dados

Fonte: Autoria própria (2024)

Na Figura 19, é mostrada a execução do SQLMap durante o processo de busca pelas tabelas do banco de dados acuart, ilustrando a interação da ferramenta com o servidor e os resultados obtidos:

- A mensagem "*[INFO] fetching tables for database: 'acuart'*" indica que o SQL-Map está buscando as tabelas do banco de dados acuart.
- O SQLMap lista as tabelas encontradas no banco de dados acuart, que são: *artists, carts, categ, featured, guestbook, pictures, products* e *users*.

Figura 19: Listagem de tabelas do banco de dados acuart



Fonte: Autoria própria (2024)

Na Figura 20, o comando “sqlmap -u http://testphp.vulnweb.com/artists.php?artist=1 -D acuart -T users –columns” está sendo executado para acessar as colunas da tabela users:

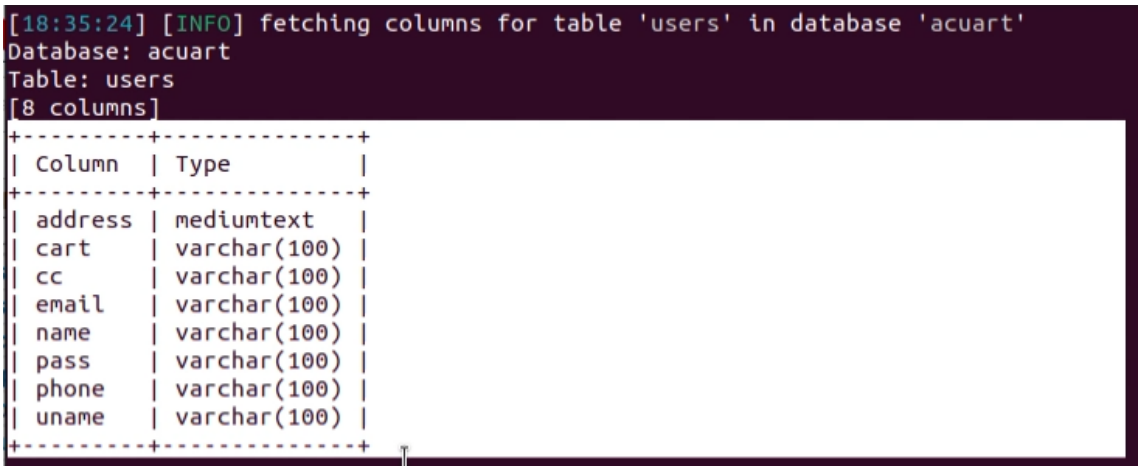
- A opção -u especifica a URL alvo, que é http://testphp.vulnweb.com/artists.php?artist=1.
- A opção -D seguida por acuart indica que estamos interessados no banco de dados acuart.
- A opção -T seguida por users especifica que queremos informações sobre a tabela users.
- A opção --columns instrui o SQLMap a listar todas as colunas da tabela especificada.

Figura 20: Acesso às colunas da tabela *users*

Fonte: Autoria própria (2024)

Na Figura 21, é apresentada a listagem de colunas. A mensagem "*[INFO] fetching columns for table 'users' in database 'acuart*'" indica que o SQLMap está buscando as colunas da tabela *users* no banco de dados acuart. As colunas listadas fornecem uma visão detalhada da estrutura da tabela *users*, incluindo os tipos de dados armazenados em cada coluna.

Figura 21: Dados armazenados na tabela *users*



Fonte: Autoria própria (2024)

Na Figura 22, o comando “sqlmap -u http://testphp.vulnweb.com/artists.php?artist=1 -D acuart -T users -C uname,pass –dump” está sendo executado para buscar os valores de “*uname*” e “*pass*” na tabela *users*:

- A opção -u especifica a URL alvo, que é “http://testphp.vulnweb.com/artists.php?artist=1”.
- A opção -D seguida por acuart indica que estamos interessados no banco de dados acuart.
- A opção -T seguida por *users* especifica que queremos informações sobre a tabela *users*.
- A opção -C seguida por *uname, pass* instrui o SQLMap a buscar especificamente as colunas *uname* (nome de usuário) e *pass* (senha).
- A opção --dump instrui o SQLMap a despejar o conteúdo dessas colunas.

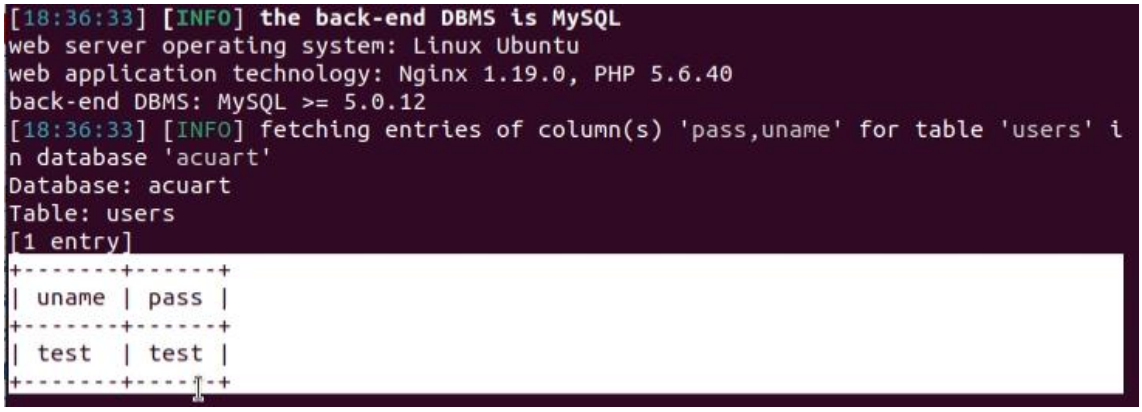
Figura 22: Acesso aos atributos na tabela *users*

Fonte: Autoria própria (2024)

Na Figura 23, é feita a busca e despejo de dados:

- A mensagem "*[INFO] fetching entries of column(s) 'pass,uname' for table 'users' in database 'acuart'*" indica que o SQLMap está buscando os dados das colunas *pass* e *uname* da tabela *users* no banco de dados acuart.
- O SQLMap despeja o conteúdo encontrado nas colunas especificadas.
- No exemplo fornecido, há uma entrada com *uname* (nome de usuário) como "test" e *pass* (senha) como "test".

Figura 23: Dados das colunas *pass* e *uname* na tabela *users*

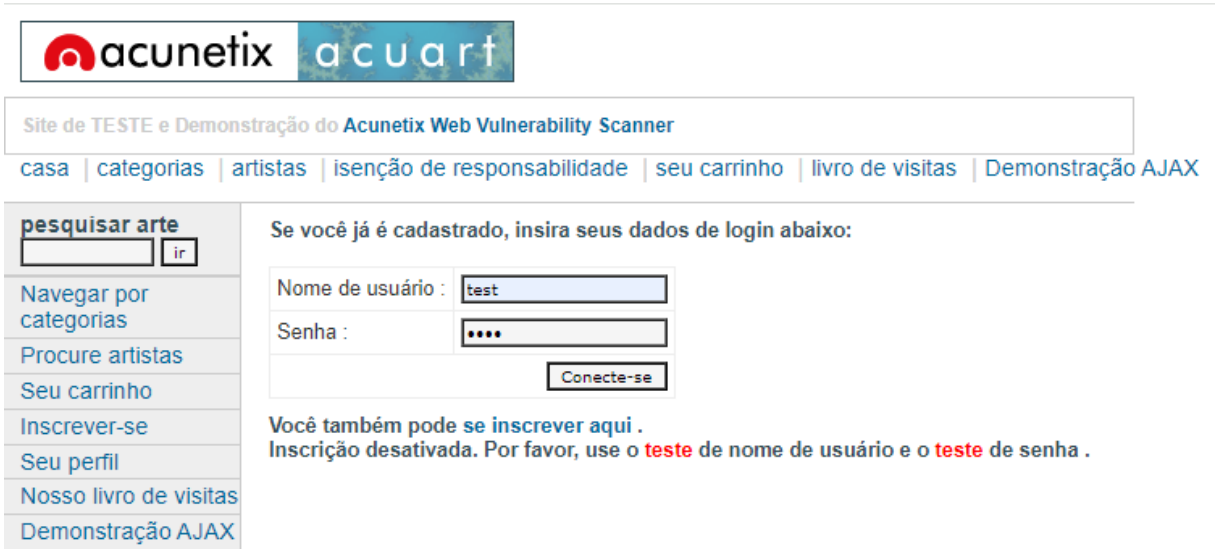


Fonte: Autoria própria (2024)

3.2.4 Fazendo *login* com usuário e senha obtidos

A Figura 24 exibe a interface de *login*, enquanto a Figura 25 confirma o acesso bem-sucedido ao sistema, evidenciando a exploração eficaz das credenciais comprometidas.

Figura 24: *Login* no site Acunetix Art



Fonte: Acunetix Art (2024)

Figura 25: Acesso às informações do usuário

Fonte: Acunetix Art (2024)

3.2.5 Busca por dados sensíveis de usuário fictício

Na Figura 26, o comando “sqlmap -u http://testphp.vulnweb.com/artists.php?artist=1 -D acuart -T users -C email –dump” está sendo executado:

- A opção -u especifica a URL alvo, que é “http://testphp.vulnweb.com/artists.php?artist=1”.
- A opção -D seguida por acuart indica que estamos interessados no banco de dados acuart.
- A opção -T seguida por users especifica que queremos informações sobre a tabela users.
- A opção -C seguida por email instrui o SQLMap a buscar especificamente a coluna email.
- A opção --dump instrui o SQLMap a despejar o conteúdo dessa coluna.

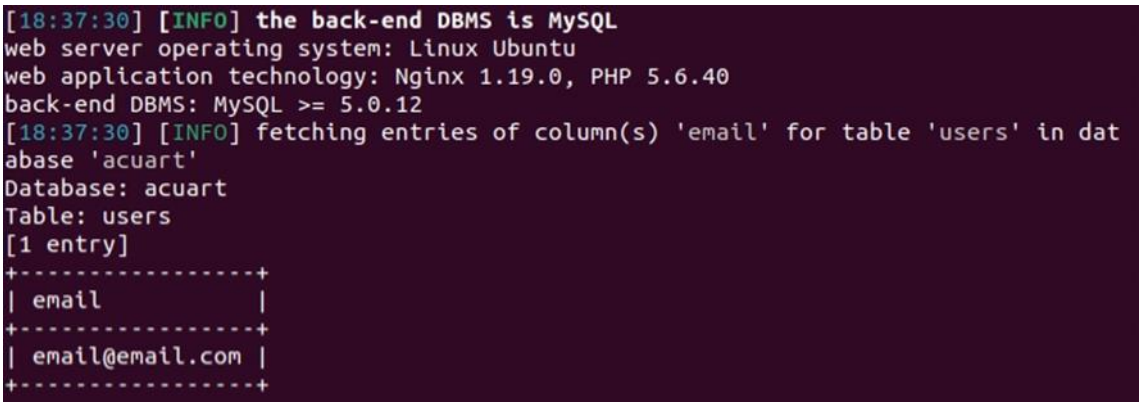
Figura 26: Busca por dados de usuário

Fonte: Autoria própria (2024)

A Figura 27 mostra o dado encontrado referente ao email do usuário.

- A mensagem “*[INFO] fetching entries of column(s) 'email' for table 'users' in database 'acuart'*” indica que o SQLMap está buscando os dados da coluna email da tabela users no banco de dados acuart.
- O SQLMap despeja o conteúdo encontrado na coluna especificada.
- No exemplo fornecido, há uma entrada com o email email@email.com.

Figura 27: Busca por email do usuário



Fonte: Autoria própria (2024)

Na Figura 28, o comando “sqlmap –u http://testphp.vulnweb.com/artists.php?artist=1 -D acuart -T users -C cc –dump” está sendo executado.

- A opção -u especifica a URL alvo, que é “http://testphp.vulnweb.com/artists.php?artist=1”.
- A opção -D seguida por acuart indica que estamos interessados no banco de dados acuart.
- A opção -T seguida por users especifica que queremos informações sobre a tabela users.
- A opção -C seguida por cc instrui o SQLMap a buscar especificamente a coluna cc (cartão de crédito).
- A opção --dump instrui o SQLMap a despejar o conteúdo dessa coluna.

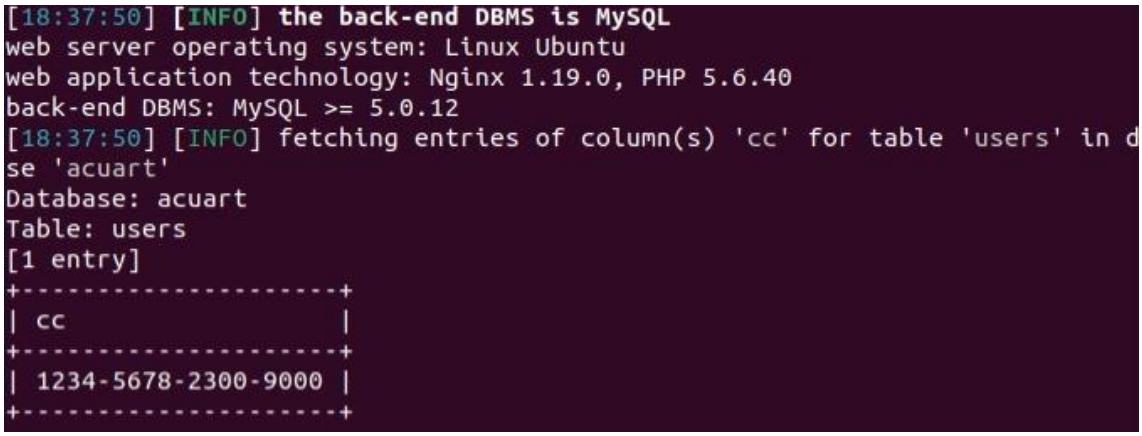
Figura 28: Busca por dados de usuário

Fonte: Autoria própria (2024)

A Figura 29 mostra o dado encontrado referente ao cartão de crédito do usuário:

- A mensagem "*[INFO] fetching entries of column(s) 'cc' for table 'users' in database 'acuart*'" indica que o SQLMap está buscando os dados da coluna cc da tabela users no banco de dados acuart.
- O SQLMap despeja o conteúdo encontrado na coluna especificada.
- No exemplo fornecido, há uma entrada com o número de cartão de crédito 1234-5678-2300-9000.

Figura 29: Busca por cc do usuário



Fonte: Autoria própria (2024)

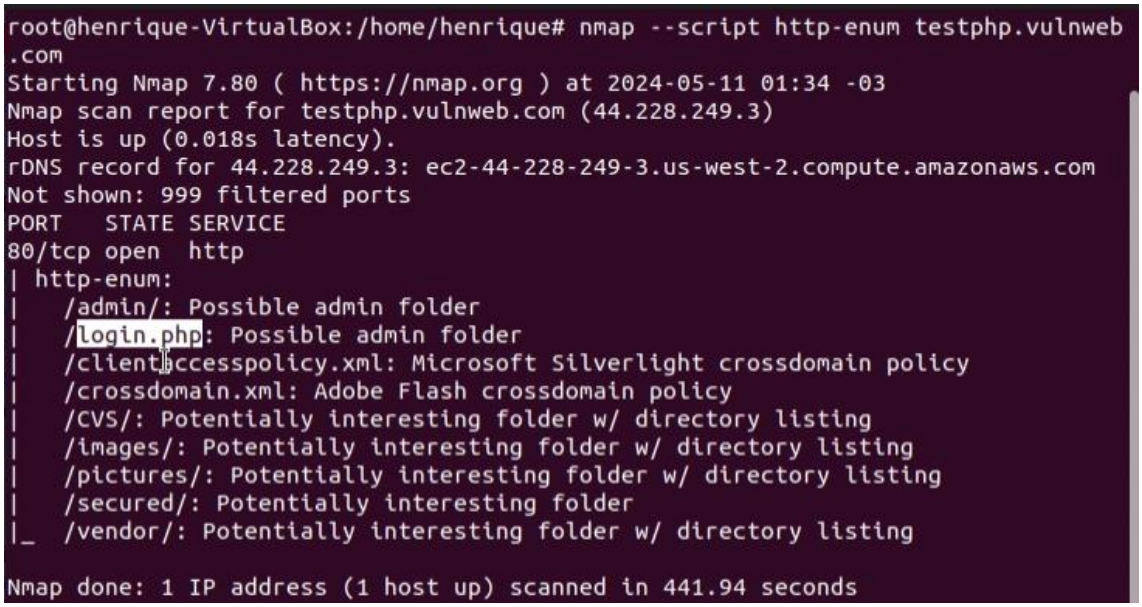
3.3 Simulação de ataque de força bruta

Nesta simulação, será realizado um processo de ataque de força bruta usando três ferramentas principais: Nmap, Hydra e o site Acunetix Art. O Nmap é uma ferramenta de linha de comando do Linux, de código aberto, que é usada para efetuar a varredura (*scan*) de endereços IP e portas em uma rede e detectar aplicações instaladas (Shivanandhan, 2023). O Hydra é uma poderosa ferramenta de força bruta desenvolvida em Python que permite aos usuários realizar ataques de *login* com base em dicionários de senhas e listas de usuários (ACADI-TI, 2023).

3.3.1 Indicação de vulnerabilidade

Na Figura 30, é apresentado o comando “nmap --script http-enum testphp.vulnweb.com”, que utiliza o script http-enum para enumerar diretórios e arquivos interessantes em um servidor web. A varredura identifica que o host testphp.vulnweb.com está ativo, com a porta 80/TCP aberta e utilizando o serviço HTTP. Entre os resultados, foram encontrados diretórios e arquivos potenciais, como /admin/ (possível diretório de administração), “login.php” (página de *login*), e outros diretórios como /*images*/, /*secured*/ e /*vendor*/. A página “login.php” apresenta um potencial formulário de *login*, que pode estar vulnerável a ataques de força bruta.

Figura 30: Varredura usando Nmap



Fonte: Autoria própria (2024)

A Figura 31 mostra o acesso à página de *login*:

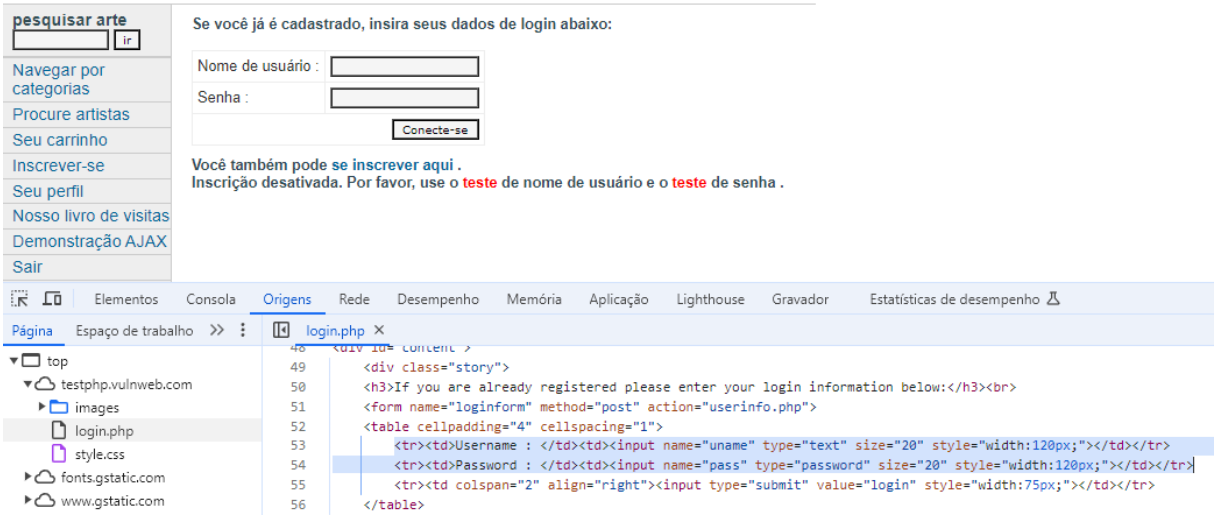
- A URL “testphp.vulnweb.com/login.php” é acessada diretamente no navegador.
- A página de *login* exibe campos para "*Username*" e "*Password*".

Figura 31: Acesso a página de *login*

Fonte: Acunetix Art (2024)

Na Figura 32, ao inspecionar o código da página, é possível notar que o campo de *login* trata-se de um formulário chamado “*loginform*”, que utiliza o método “*post*” e tem como ação “userinfo.php”. Foi identificado o nome do campo “*Username”* como “*uname*”, e do campo “*Password”* como “*pass*”.

Figura 32: Campos *Username* e *Password*



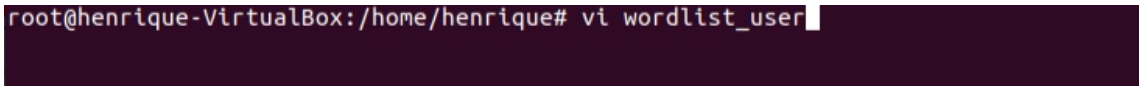
Fonte: Acunetix Art (2024)

3.3.2 Criação de *wordlists*

Para realizar um ataque de força bruta utilizando a ferramenta Hydra, foi necessário criar *wordlists* contendo possíveis nomes de usuários e senhas. A seguir, cada imagem ilustra uma etapa desse processo de criação de *wordlists*. Usando o editor de texto vi, duas *wordlists* são criadas, uma para possíveis usuários e outra para possíveis senhas.

Na Figura 33, é exibido o comando utilizado para abrir e editar a *wordlist* "*wordlist_user"* no terminal, usando o editor de texto vi.

Figura 33: *Wordlist_user*



Fonte: Autoria própria (2024)

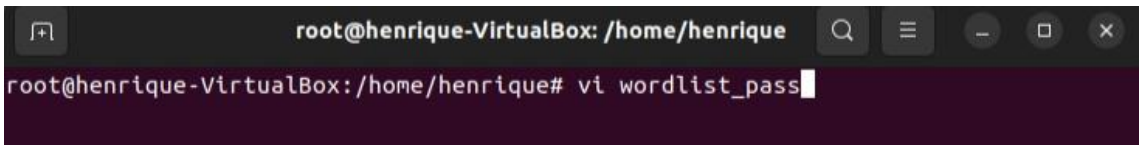
Em seguida, a Figura 34 mostra o conteúdo da *wordlist "wordlist_user"*, incluindo exemplos de possíveis nomes de usuários como "admin", "teste", "user", "administrador" e "test".

Figura 34: Possíveis nomes de usuário

Fonte: Autoria própria (2024)

Posteriormente, na Figura 35, é demonstrado o comando para editar a *wordlist "wordlist_pass"*, que contém as possíveis senhas a serem testadas.

Figura 35: *Wordlist_pass*



Fonte: Autoria própria (2024)

O conteúdo dessa *wordlist* é apresentado na Figura 36, incluindo senhas como “teste”, “admin”, “123456” e “test”.

Figura 36: Possíveis senhas

```
teste
administrador
admin
123456
test
```

Fonte: Autoria própria (2024)

3.3.3 Uso do Hydra para força bruta

Na Figura 37, é mostrado a execução do comando “hydra testphp.vulnweb.com http-form-post "/userinfo.php:uname=^USER^&pass=^PASS^:login page" -L wordlist_user -P wordlist_pass –V”.

- http-form-post define o tipo de formulário de *login*.
- /userinfo.php:uname=^USER^&pass=^PASS^:*login page* especifica a ação do formulário, os campos de login e senha, e o indicador de falha de *login*.
- -L *wordlist_user* especifica a *wordlist* de nomes de usuário.
- -P *wordlist_pass* especifica a *wordlist* de senhas.
- -V habilita o modo verbose para exibir mais detalhes durante a execução.

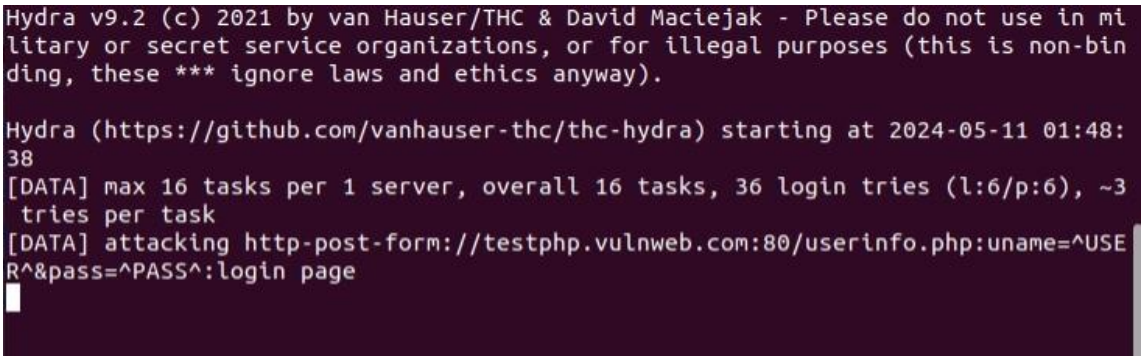
Figura 37: Execução do comando hydra

```
root@henrique-VirtualBox:/home/henrique# hydra testphp.vulnweb.com http-form-pos
t "/userinfo.php:uname=^USER^&pass=^PASS^:login page" -L wordlist_user -P wordli
st_pass -V
```

Fonte: Autoria própria (2024)

Na Figura 38, é apresentado o início da execução do Hydra, conforme indicado pelo *timestamp* e pela versão da ferramenta. Durante o processo, o Hydra configura as tarefas necessárias para testar combinações de *login*. A mensagem *"[DATA] attacking http-post-form*" confirma que o ataque ao formulário de *login* foi iniciado.

Figura 38: Início da execução



Fonte: Autoria própria (2024)

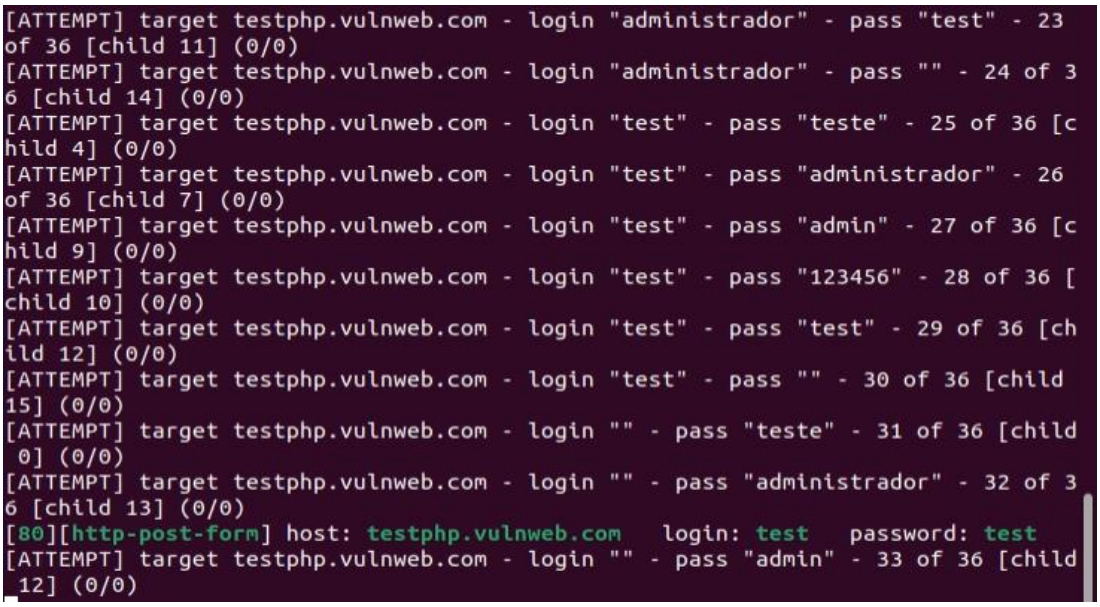
Na Figura 39, é apresentado o processo de tentativas de *login* realizadas pelo Hydra, que testa várias combinações de nomes de usuário e senhas especificadas nas *wordlists*. Cada tentativa é registrada, exibindo a combinação de *login* e senha utilizada.

Figura 39: Tentativas de *login*

Fonte: Autoria própria (2024)

Na Figura 40, é demonstrado o processo de identificação de credenciais válidas pelo Hydra, que continua testando combinações até encontrar uma correspondência. A combinação de *username* "test" e *password* "test" é identificada como válida.

Figura 40: Tentativas de *login*



Fonte: Autoria própria (2024)

Na Figura 41, é apresentada a conclusão da execução do Hydra, que finaliza o processo após identificar uma combinação válida. A mensagem "*1 of 1 target successfully completed, 1 valid password found*" confirma que uma senha válida foi encontrada. O *timestamp* registrado indica o horário exato de conclusão do ataque.

Figura 41: Conclusão da execução

Fonte: Autoria própria (2024)

4 ESTRATÉGIAS DE MITIGAÇÃO

A mitigação de ataques cibernéticos envolve a implementação de medidas para reduzir vulnerabilidades e impedir o acesso não autorizado. Uma abordagem eficaz inclui o uso de criptografia, autenticação robusta e validação rigorosa de entradas. A aplicação de protocolos seguros, como TLS/SSL, e a monitoração contínua dos sistemas são essenciais para prevenir ataques como *Man-in-the-Middle (MitM)*, *SQL injection* e ataques de força bruta, além de fortalecer a defesa contra ameaças em rede (Stallings, 2008).

4.1 Mitigação de ataques *Man-in-the-Middle (MitM)*

A mitigação de ataques *Man-in-the-Middle (MitM)* é essencial para garantir a segurança das comunicações em servidores web. Conforme destacado por Stallings (2008) em seu livro "Criptografia e Segurança de Redes: Princípios e práticas", uma das principais estratégias para proteger os dados em trânsito é a criptografia robusta, que impede a interceptação e leitura dos dados por terceiros mal-intencionados. A utilização do HTTPS, que emprega protocolos TLS/SSL, é crucial para garantir que os dados transmitidos entre cliente e servidor estejam criptografados, tornando muito difícil para atacantes interceptarem e acessarem informações sensíveis.

Além da criptografia, a implementação de certificados digitais emitidos por autoridades certificadoras confiáveis é fundamental. Stallings (2008) explica que esses certificados asseguram a autenticidade dos servidores, evitando ataques que envolvem a falsificação de identidade. Sem um certificado válido, um atacante pode facilmente se passar por um servidor legítimo, capturando dados confidenciais durante o processo.

Outra medida vital na mitigação de ataques MitM é a adoção de autenticação forte. Segundo Kurose; Ross (2010) em "Redes de Computadores e a Internet: uma abordagem *top-down*", a autenticação multifator (MFA) adiciona uma camada extra de segurança além da senha tradicional. Essa abordagem exige que o usuário forneça um segundo fator de autenticação, como um código enviado para o celular ou gerado por uma aplicação de autenticação. Isso dificulta significativamente o acesso não autorizado, mesmo que a senha seja comprometida.

O uso do *HTTP Strict Transport Security (HSTS)* é outra recomendação importante para prevenir ataques MitM. HSTS instrui os navegadores a usar exclusivamente conexões HTTPS com o servidor, evitando ataques de *downgrade* onde um atacante força a conexão para HTTP não seguro. Stallings (2008) enfatiza que essa prática garante que todas as comunicações sejam protegidas por criptografia robusta.

Ademais, deve-se monitorar e detectar atividades suspeitas. Ferramentas de monitoramento de rede, como sistemas de detecção de intrusões (IDS), são essenciais para identificar padrões anômalos de tráfego que possam indicar a presença de um MitM. Conforme discutido por Stallings (2008), essas ferramentas analisam o tráfego em tempo real, alertando os administradores sobre atividades potencialmente maliciosas.

Por fim, a análise e verificação da integridade dos certificados digitais recebidos ajudam a detectar certificados falsificados, um método comum utilizado em ataques MitM. Implementar mecanismos de verificação contínua dos certificados digitais assegura que apenas conexões legítimas e seguras sejam estabelecidas, protegendo assim os usuários de possíveis interceptações (Kurose; Ross, 2010).

Estas medidas, quando combinadas, formam uma defesa robusta contra ataques *Man-in-the-Middle*, protegendo as comunicações e garantindo a integridade e confidencialidade dos dados transmitidos.

4.2 Mitigação de ataques *SQL injection*

A mitigação de ataques *SQL injection* é essencial para proteger bancos de dados contra acessos não autorizados e manipulações maliciosas. Diversas estratégias são recomendadas pela para prevenir e mitigar esses ataques, assegurando a integridade e a segurança dos dados armazenados.

A primeira linha de defesa envolve a validação e sanitização rigorosa das entradas fornecidas pelos usuários. Implementar mecanismos de validação de dados é crucial para garantir que as informações recebidas sejam do tipo e formato esperados, reduzindo a possibilidade de injeções de código malicioso. Isso significa que os dados fornecidos pelos usuários devem ser verificados para assegurar que correspondem aos tipos e formatos esperados, como números inteiros, *strings* e datas, além da aplicação de expressões regulares para garantir conformidade com padrões específicos. Além disso, a sanitização de entradas, utilizando funções específicas para limpar os

dados e remover ou escapar caracteres especiais, é uma prática indispensável para eliminar potenciais vetores de ataque (OWASP, 2023).

Outra medida eficaz é o uso de consultas preparadas, também conhecidas como *prepared statements*. Essas consultas substituem as dinâmicas por parametrizadas, onde os dados de entrada são tratados como parâmetros ao invés de serem inseridos diretamente nas instruções SQL. Essa técnica evita a execução de código SQL arbitrário, tornando a aplicação mais segura contra tentativas de injeção de SQL (OWASP, 2023).

O princípio do menor privilégio é outra abordagem fundamental na mitigação de ataques *SQL injection*. As contas de banco de dados utilizadas pelas aplicações web devem ser configuradas com os privilégios mínimos necessários para realizar suas funções. Restringir as permissões ajuda a garantir que, mesmo se uma conta for comprometida, o atacante não possa executar operações destrutivas ou acessar dados sensíveis (Augustėnė, 2023).

Além dessas medidas preventivas, o monitoramento contínuo e a realização de testes de segurança são essenciais. Testes de penetração regulares permitem identificar e corrigir vulnerabilidades antes que sejam exploradas por atacantes. Esses testes simulam ataques reais, proporcionando uma visão prática das possíveis falhas de segurança e permitindo a implementação de correções antecipadas (OWASP, 2023).

Por fim, a utilização de firewalls de aplicação web (WAF) adiciona uma camada extra de proteção. WAFs são usados para fazer o monitoramento, a filtragem e o bloqueio de tráfego HTTP que entra e sai de um serviço web, sendo capazes de detectar e bloquear tentativas de injeção de SQL em tempo real, atuando como uma barreira entre os atacantes e os servidores web. Ferramentas de monitoramento de rede, como sistemas de detecção de intrusões (IDS), complementam esta proteção, ajudando a identificar padrões anômalos de tráfego e alertando os administradores sobre atividades suspeitas. (Augustėnė, 2023).

A implementação combinada dessas estratégias proporciona uma defesa robusta contra ataques *SQL injection*, assegurando a integridade e a segurança dos dados armazenados em bancos de dados das aplicações web.

4.3 Mitigação de ataques de força bruta

A mitigação de ataques de força bruta requer uma abordagem multifacetada que combina políticas de senhas fortes, limitação de tentativas de *login*, autenticação multifator e monitoramento contínuo. A implementação de políticas rigorosas de senhas é um passo inicial crucial. Estas políticas devem exigir que as senhas sejam fortes e complexas, incluindo uma combinação de letras maiúsculas e minúsculas, números e caracteres especiais. Isso torna mais difícil para atacantes adivinharem ou quebrarem as senhas através de tentativas repetidas. Além disso, a política de expiração periódica de senhas força os usuários a trocar suas senhas regularmente, limitando o tempo durante o qual uma senha comprometida pode ser utilizada (Šimonėlytė, 2023).

Para prevenir ataques de força bruta, é igualmente importante limitar o número de tentativas de *login* falhas. Implementar mecanismos de bloqueio de contas após um número definido de tentativas de *login* malsucedidas pode ser extremamente eficaz. Esse bloqueio pode ser temporário, desencorajando tentativas repetidas e automáticas, ou permanente, necessitando de intervenção administrativa para desbloquear a conta (Paulino, 2020). Além disso, o uso de CAPTCHAs nas páginas de *login* pode dificultar a automação das tentativas de *login*, tornando a tarefa de ataques automatizados mais complicada e menos eficiente (Šimonėlytė, 2023).

A adoção da autenticação multifator (MFA) é uma camada adicional de segurança que pode significativamente reduzir o risco de comprometimento das contas. Ao exigir mais de um fator de autenticação, como um código enviado para um dispositivo móvel ou gerado por um aplicativo de autenticação, a MFA adiciona uma barreira que os atacantes precisam superar, tornando os ataques de força bruta substancialmente menos eficazes (Paulino, 2020).

Monitorar tentativas de *login* e detectar padrões suspeitos de acesso também são práticas recomendadas para mitigar ataques de força bruta. A implementação de sistemas de monitoramento de acessos permite a identificação de atividades anômalas e o acionamento de alertas para notificar os administradores sobre possíveis tentativas de ataque. Além disso, a utilização de sistemas de detecção e prevenção de intrusões (IDS/IPS) pode identificar e bloquear ataques de força bruta em tempo real, fornecendo uma linha de defesa ativa contra tentativas de acesso não autorizadas (Fernandes, 2022).

A combinação dessas medidas cria uma defesa robusta contra ataques de força bruta, garantindo que as credenciais dos usuários sejam protegidas e que a segurança dos sistemas seja mantida. Implementar essas estratégias de forma coerente e integrada é essencial para criar um ambiente seguro e resistente contra as ameaças cibernéticas.

5 IMPLEMENTAÇÃO DE UM SISTEMA DE LOGIN SEGURO

Esta seção apresentará um site de *login* desenvolvido com foco na segurança, buscando prevenir ataques comuns como *SQL injection*, *Man-in-the-Middle (MitM)* e força bruta. O site utiliza boas práticas de segurança, incluindo validação de entradas, utilização do protocolo HTTPS e autenticação segura. A análise será feita com base em *prints* das funcionalidades implementadas, destacando como cada uma contribui para evitar esses ataques e garantir a integridade e segurança dos dados.

5.1 Ferramentas utilizadas

Para o desenvolvimento do site de *login* seguro, foram utilizadas diversas ferramentas. O XAMPP, uma plataforma que combina servidor Apache, MySQL e interpretadores de PHP, foi empregado para criar um ambiente local adequado para o desenvolvimento e teste de aplicações web. Além disso, o phpMyAdmin, uma ferramenta gráfica para gerenciamento de bancos de dados MySQL, foi utilizado para configurar e gerenciar as tabelas de usuários, bem como para executar consultas SQL de forma segura. O site foi desenvolvido utilizando HTML para estruturar o *front-end* e PHP no *back-end*, responsável por processar os dados inseridos pelos usuários. No *back-end*, o PHP implementou medidas de segurança essenciais para validar entradas e garantir uma comunicação segura com o banco de dados.

5.2 Conexão com banco de dados e gestão de usuários

Na Figura 42, é possível observar a interface principal do site de *login*, desenvolvida em HTML e PHP. A página de *login* é simples e objetiva, contendo dois campos principais: um para o nome de usuário e outro para a senha.

Figura 42: Página de *login*

Fonte: Autoria própria (2024)

As figuras abaixo mostram a implementação da conexão com o banco de dados e a gestão dos usuários. Na Figura 43, é mostrada a lógica para estabelecer a conexão com o banco de dados MySQL, utilizando a extensão mysqli. O código realiza a verificação da conexão e, em caso de falha, exibe uma mensagem de erro. Esse tipo de verificação é importante para garantir que a aplicação não funcione sem estar conectada ao banco de dados, prevenindo problemas futuros na execução de queries.

Já na Figura 44, é exibida a tabela de usuários no phpMyAdmin, mostrando a estrutura dos dados armazenados, como nome de usuário e senha. As senhas dos usuários estão armazenadas usando *hashing*, que transforma as senhas em uma *string* aleatória e irreversível, garantindo que mesmo se o banco de dados for comprometido, as senhas reais não possam ser facilmente descobertas. Isso ajuda a proteger contra ataques como força bruta e roubo de credenciais.

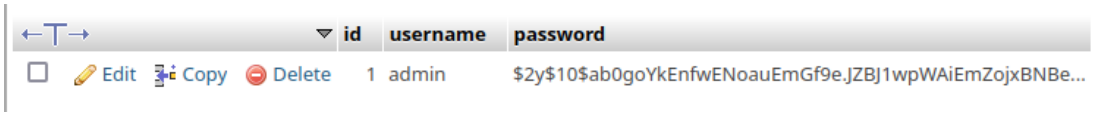
Figura 43: Lógica para estabelecer conexão com o banco de dados

```php
<?php
$servername = "localhost";
$username = "root";
$password = "";
$dbname = "secure_login";

try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch (PDOException $e) {
    echo "Conexão falhou: " . $e->getMessage();
}
?>
```

Fonte: Autoria própria (2024)

Figura 44: Estrutura dos dados armazenados

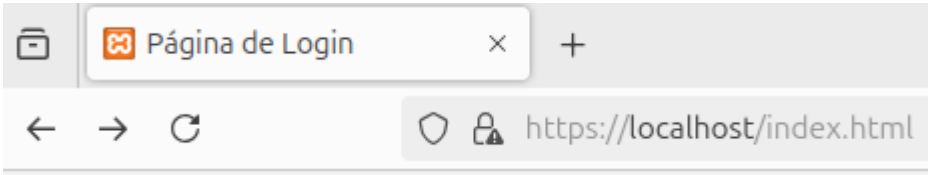


Fonte: Autoria própria (2024)

5.3 Prevenção contra ataques *Man-in-the-Middle (MitM)*

Na Figura 45, o uso do protocolo HTTPS é indicado pelo cadeado ao lado da URL "https://localhost/index.html", mostrando que os dados transmitidos entre cliente e servidor estão criptografados, protegendo contra ataques *Man-in-the-Middle (MitM)*. Conforme destacado por Stallings (2008), o uso de TLS/SSL é essencial para impedir a interceptação de dados por terceiros mal-intencionados, garantindo a integridade e confidencialidade das informações. Embora tenha sido utilizado um certificado SSL auto assinado no ambiente de desenvolvimento local, essa prática demonstra a importância de usar criptografia robusta. Em ambientes de produção, é crucial utilizar certificados emitidos por autoridades confiáveis para evitar ataques de falsificação de identidade.

Figura 45: Uso do HTTPS no site de *login*



Fonte: Autoria própria (2024)

5.4 Prevenção contra ataques *SQL injection*

Nestes trechos do código de *login*, são aplicadas várias medidas de segurança para prevenir ataques de *SQL injection*.

Na Figura 46**,** é apresentada a função “sanitizarEntrada()”, que desempenha um papel fundamental na mitigação de ataques como o *SQL injection*. A função utiliza “htmlspecialchars()” para escapar caracteres especiais que poderiam ser utilizados para injetar comandos maliciosos, alinhando-se com as práticas de segurança descritas pela OWASP (2023) para sanitização de entradas. Além disso, “trim()” remove espaços em branco extras, enquanto “stripslashes()” remove barras invertidas que poderiam ser usadas para burlar validações de entrada. Essas práticas garantem que os dados processados pela aplicação sejam limpos e seguros, prevenindo a inserção de código malicioso.

Figura 46: Função para sanitização de entradas

```php
// Função para sanitizar entradas
function sanitizarEntrada($dados) {
    return htmlspecialchars(stripslashes(trim($dados)));
}
```

Fonte: Autoria própria (2024)

Na Figura 47, o código mostra a aplicação da função de sanitização para processar os dados inseridos pelo usuário no formulário de *login*. Tanto o campo "usuario" quanto "senha" são filtrados usando “FILTER_SANITIZE_STRING” antes de passarem pela função “sanitizarEntrada()”. Isso garante que os dados submetidos sejam tratados conforme o esperado, atendendo à exigência de validação rigorosa mencionada no texto de OWASP (2023). Ao remover ou escapar caracteres perigosos, o código evita a entrada de dados maliciosos, reforçando a proteção contra *SQL injection* e outros tipos de ataques baseados em manipulação de entradas.

Figura 47: Sanitização de entradas

```php
// Verifica se o formulário foi enviado
if ($_SERVER['REQUEST_METHOD'] === 'POST') {

    // Sanitização de entradas do formulário
    $usuario = sanitizarEntrada(filter_input(INPUT_POST, 'username',
FILTER_SANITIZE_STRING));
    $senha = sanitizarEntrada(filter_input(INPUT_POST, 'password',
FILTER_SANITIZE_STRING));
```

Fonte: Autoria própria (2024)

Na Figura 48, o uso de consultas preparadas com o método “prepare()” e “bindParam()” é uma das formas mais eficazes de prevenir ataques de *SQL injection***,** como descrito pela OWASP (2023). Essa técnica assegura que os dados fornecidos pelo usuário, como o nome de usuário, sejam tratados como parâmetros e não como parte da consulta SQL em si. Assim, qualquer tentativa de injeção de código malicioso

é neutralizada, pois o dado inserido pelo usuário não pode modificar a estrutura da query SQL.

Figura 48: Consultas preparadas

```
    // Usar consultas preparadas para evitar SQL Injection
    $stmt = $conn->prepare("SELECT username, password FROM users WHERE
username = :username");
    $stmt->bindParam(':username', $usuario);
    $stmt->execute();

    $user = $stmt->fetch(PDO::FETCH_ASSOC);
```

Fonte: Autoria própria (2024)

Além disso, na Figura 49, o código utiliza a função “password_verify()” para verificar a correspondência da senha fornecida pelo usuário com o *hash* da senha armazenada, garantindo uma verificação segura. O controle de tentativas de *login* através de “$_SESSION['attempt']” também reforça a segurança ao limitar o número de tentativas falhas, mitigando possíveis ataques de força bruta. Essas práticas combinadas criam uma defesa robusta tanto contra *SQL injection* quanto contra tentativas de adivinhação de senha.

Figura 49: Verificação de senha

```
    if ($user) {
        // Verifica se a senha fornecida corresponde à senha armazenada
(hash)
        if (password_verify($senha, $user['password'])) {
            // Login bem-sucedido
            $_SESSION['attempt'] = 0; // Reseta tentativas
            header("Location: success.php");
            exit();
        } else {
            // Senha incorreta
            $_SESSION['attempt']++;
            $_SESSION['last_attempt_time'] = time();
            $remaining_attempts = $max_attempts - $_SESSION['attempt'];
            header("Location: fail.php?attempts=$remaining_attempts");
            exit();
        }
```

Fonte: Autoria própria (2024)

5.4.1 Comparação entre o site vulnerável e sistema de login seguro

Ao realizar os testes de *login*, é possível notar como a implementação de práticas de segurança impacta diretamente a proteção contra injeções de SQL baseadas em erro. A seguir, são apresentados os resultados obtidos ao comparar um site vulnerável, como o Acunetix Art, e o site seguro desenvolvido.

5.4.1.1 Site vulnerável Acunetix Art

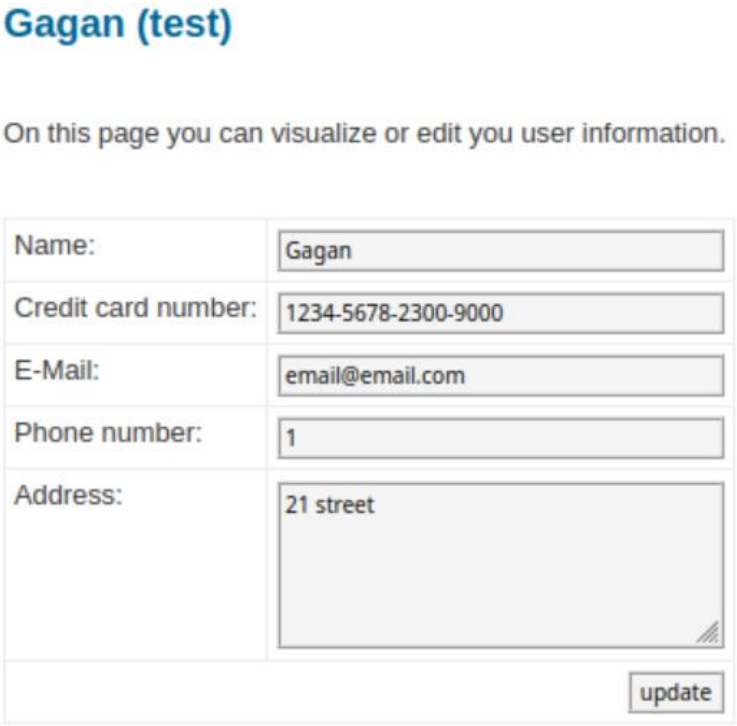
A Figura 50 mostra que o campo de *login* permite a entrada maliciosa 'OR '1'='1 no campo de usuário. Essa entrada manipula a consulta SQL para sempre retornar verdadeiro, permitindo ao invasor ignorar o processo de autenticação.

Figura 50: Entrada do ataque no site vulnerável

Fonte: Acunetix Art (2024)

Após o ataque, o invasor obtém acesso a informações confidenciais de um usuário registrado, como nome, número de cartão de crédito, e-mail, número de telefone e endereço, como mostrado na Figura 51. Essas falhas decorrem da ausência de práticas de segurança, como sanitização de entradas e uso de consultas preparadas. Como mencionado por Daityari (2023), a manipulação inadequada de entradas permite que o invasor explore a lógica da consulta SQL, causando resultados inesperados ou prejudiciais.

Figura 51: Resultado de *login* no site vulnerável

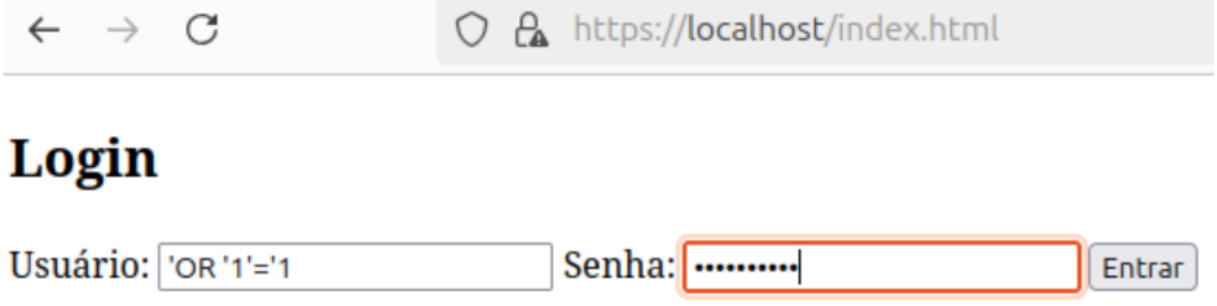


Fonte: Acunetix Art (2024)

5.4.1.2 Sistema de login seguro

Como mostrado na Figura 52, no site de *login* seguro desenvolvido, a mesma entrada maliciosa ('OR '1'='1) foi inserida no campo de usuário. Contudo, a aplicação utiliza práticas robustas de segurança, como sanitização de entradas e consultas preparadas com “prepare()” e “bindParam()”. Essas práticas neutralizam a tentativa de manipular a consulta SQL, tratando a entrada do usuário como simples texto, sem permitir alterações na lógica da consulta.

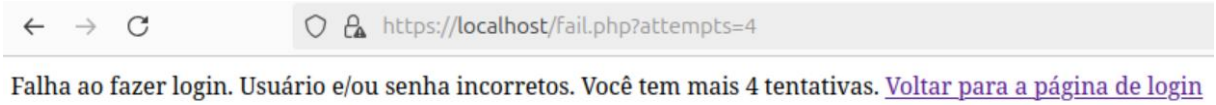
Figura 52: Entrada do ataque no site seguro



Fonte: Autoria própria (2024)

A aplicação exibe uma mensagem de erro genérica, informando que o *login* falhou, o que pode ser visto na Figura 53. Além disso, há um controle de tentativas, que permite apenas um número limitado de *login* antes de bloquear temporariamente o acesso. Essa abordagem garante que nenhuma informação adicional seja exposta ao invasor, mesmo diante de tentativas maliciosas.

Figura 53: Resultado de *login* no site seguro



Fonte: Autoria própria (2024)

Conforme explicado pela OWASP (2023), o uso de consultas preparadas é uma das práticas mais eficazes para impedir que invasores manipulem consultas SQL e obtenham informações sobre a estrutura do banco de dados, protegendo a aplicação contra injeções de SQL baseadas em erro.

5.5 Prevenção contra ataques de força bruta

Nestes trechos do código de *login*, são aplicadas medidas de segurança para prevenir ataques de força bruta, através da limitação das tentativas de *login*.

Na Figura 54, observa-se a implementação de um limite de tentativas de *login* por meio das variáveis “$max_attempts” e “$block_time”. Esses parâmetros estabelecem um número máximo de tentativas falhas (neste caso, 5) e um tempo de bloqueio (60 segundos) para evitar tentativas repetitivas de *login*, o que é uma prática eficaz na prevenção de ataques de força bruta. Conforme Paulino (2020) destaca, esse tipo de mecanismo temporário desestimula tentativas automáticas de adivinhar senhas.

Figura 54: Limite de tentativas de *login*

```
// Limite de tentativas de login
$max_attempts = 5;
$block_time = 60; // 1 minuto
```

Fonte: Autoria própria (2024)

Na Figura 55, o código verifica se a sessão de tentativas de *login* ($_SESSION['attempt']) já foi inicializada. Caso contrário, ela é definida como zero, garantindo que o sistema conte corretamente as tentativas de *login* a partir da primeira interação do usuário. Esse mecanismo assegura que o controle de tentativas falhas esteja ativo desde o início da sessão, protegendo a aplicação contra ataques baseados em várias tentativas de *login*.

Figura 55: Sessão de tentativas de *login*

```php
if (!isset($_SESSION['attempt'])) {
    $_SESSION['attempt'] = 0;
    $_SESSION['last_attempt_time'] = time();
}
```

Fonte: Autoria própria (2024)

Na Figura 56, o código implementa a lógica para verificar se o número de tentativas de *login* excedeu o limite definido em “$max_attempts”. Se o usuário exceder esse limite, o sistema calcula quanto tempo já se passou desde a última tentativa ($time_since_last_attempt) e compara com o tempo de bloqueio definido. Se o tempo de bloqueio ainda não tiver passado, o usuário é notificado e direcionado para tentar novamente após o período de espera. Caso o período de bloqueio tenha expirado, o contador de tentativas é reiniciado ($_SESSION['attempt'] = 0). Essa estratégia de bloqueio temporário, conforme Paulino (2020), é fundamental para mitigar ataques de força bruta ao limitar a frequência das tentativas de *login* automáticas, tornando esses ataques menos eficazes.

Figura 56: Tentativas de *login* e tempo de bloqueio

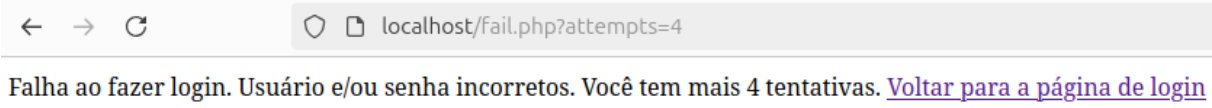
```
    // Verifica se o usuário excedeu as tentativas e aplica o bloqueio
    if ($_SESSION['attempt'] >= $max_attempts) {
        $time_since_last_attempt = time() - $_SESSION['last_attempt_time'];

        if ($time_since_last_attempt < $block_time) {
            // Se o tempo de bloqueio ainda não passou, exibe a mensagem e
redireciona
            echo "Você excedeu o número de tentativas de login. Tente
novamente em " . (int)($block_time - $time_since_last_attempt) . "
segundos.";
            header("refresh:10;url=index.html"); // Redireciona após 10
segundos
            exit();
        } else {
            // Reseta as tentativas após o período de bloqueio
            $_SESSION['attempt'] = 0;
        }
    }
```

Fonte: Autoria própria (2024)

Após o usuário falhar em suas tentativas de *login*, o sistema exibe uma mensagem ao usuário, como mostrado na Figura 57. A mensagem informa que o nome de usuário ou senha estão incorretos. O código também comunica ao usuário quantas tentativas restam, ajudando a manter uma comunicação clara.

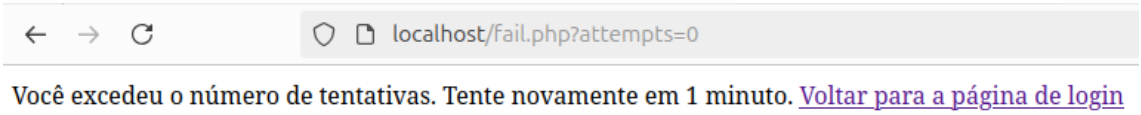
Figura 57: Validação de usuário e senha



Fonte: Autoria própria (2024)

Caso o usuário exceda o número de tentativas permitidas, conforme definido anteriormente, o sistema exibe a mensagem da Figura 58, informando que o número de tentativas foi excedido e que ele deve aguardar 1 minuto antes de tentar novamente.

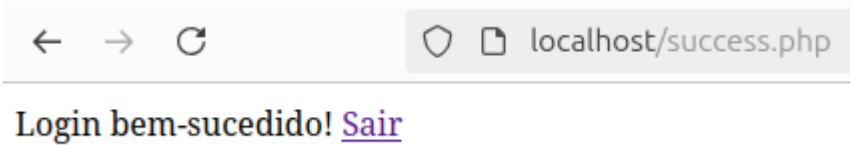
Figura 58: Tentativas de *login* excedidas



Fonte: Autoria própria (2024)

A Figura 59, mostra um caso em que após a autenticação, indicando que o usuário inseriu as credenciais corretas e passou pelas verificações de segurança implementadas, a página exibe uma mensagem de "Login bem-sucedido!”

Figura 59: Sucesso ao fazer *login*



Fonte: Autoria própria (2024)

6 CONSIDERAÇÕES FINAIS

O presente trabalho teve como objetivo explorar e analisar detalhadamente vulnerabilidades em servidores web, com ênfase em ataques cibernéticos como *Man-in-the-Middle (MitM), SQL injection* e ataques de força bruta, além de apresentar estratégias de mitigação para minimizar tais ameaças. Através de uma abordagem que combinou pesquisa teórica e simulações práticas, foi possível não apenas identificar as principais fragilidades em protocolos de comunicação, como HTTP e HTTPS, mas também propor soluções de segurança viáveis e aplicáveis em contextos reais.

A elaboração deste trabalho envolveu diversas etapas, cada uma com desafios específicos. Durante a fase de pesquisa e levantamento de informações, houve um aprofundamento no estudo dos protocolos e nas vulnerabilidades associadas, o que permitiu uma compreensão mais sólida dos fundamentos da segurança em servidores web, essencial para embasar as simulações subsequentes. O uso das ferramentas Ettercap, SQLMap e Hydra mostrou-se fundamental para ilustrar a aplicação prática dos conceitos estudados, permitindo uma análise mais tangível dos riscos e dos métodos de ataque.

Na etapa de simulação, foram realizados ataques controlados a ambientes de teste, o que possibilitou observar as consequências de vulnerabilidades exploradas de maneira prática. As simulações demonstraram como ferramentas amplamente acessíveis podem ser usadas por agentes mal-intencionados, destacando a necessidade de reforço constante das práticas de segurança. A partir dessas experiências, foram formuladas recomendações de mitigação que incluem o uso de criptografia, autenticação multifatorial, implementação de políticas de senha robustas, consultas parametrizadas e controle de acesso rigoroso.

As dificuldades mais significativas encontradas ao longo do desenvolvimento concentraram-se nas implementações práticas das simulações dos ataques cibernéticos e na criação de um site para demonstrar uma aplicação mais segura. A etapa de simulação exigiu um domínio técnico aprofundado sobre cada ferramenta utilizada, além de ajustes específicos nos ambientes de teste. Cada ferramenta apresentou suas particularidades, e configurar o Ettercap, SQLMap e Hydra para capturar tráfego e explorar vulnerabilidades de forma controlada envolveu muitos testes e configurações para garantir que as simulações ocorressem sem comprometer a segurança do ambiente.

A criação do site para demonstrar métodos de defesa também representou um desafio técnico. O site foi desenvolvido com foco em segurança e incluiu o uso do protocolo HTTPS com um certificado auto assinado, sanitização de entradas, consultas preparadas para evitar *SQL injection* e um limite de tentativas de *login* para proteger contra ataques de força bruta. Cada uma dessas medidas de segurança exigiu um planejamento cuidadoso e testes rigorosos para assegurar que funcionassem conforme esperado, tornando a implementação um processo complexo, mas enriquecedor, que complementou o embasamento teórico.

A pesquisa realizada abre caminho para futuros estudos e melhorias no campo da segurança cibernética. Entre os tópicos que podem ser explorados estão a adaptação de estratégias de mitigação para novos tipos de ataques que surgem constantemente no cenário digital, especialmente com o aumento da complexidade dos sistemas e da sofisticação dos ciberataques. Outras áreas de interesse incluem o desenvolvimento de métodos de detecção de intrusão mais eficientes, capazes de identificar comportamentos anômalos em tempo real e responder de forma proativa às ameaças.

Além disso, seria interessante a criação de uma plataforma educacional que permita aos usuários aprenderem e experimentarem práticas de segurança cibernética em um ambiente controlado. Essa plataforma poderia ser utilizada para simular ataques e defesas, aumentando a conscientização e o treinamento em segurança digital para novos profissionais e usuários comuns. Por fim, a integração de inteligência artificial para identificar padrões de ataques e automatizar respostas em servidores web também representa uma área promissora para pesquisas futuras, visando aumentar a eficiência e a precisão das defesas cibernéticas.

7 REFERÊNCIAS

ACADI-TI. **Hydra**: ferramenta de brute force para testes de segurança. 2023. Disponível em: https://acaditi.com.br/hydra-ferramenta-de-brute-force-para-testes-de-seguranca/. Acesso em: 29 mai. 2024.

ACUNETIX ART. Disponível em: http://testphp.vulnweb.com. Acesso em: 03 mai. 2024.

AHSAN, A.; MALLIKA, A.; MD., M.; TSOU, J. **Man-in-the-middle-attack**: understanding in simple words. 2019. Disponível em: https://www.researchgate.net/publication/330249434_Man-in-the-middle-attack_Understanding_in_simple_words. Acesso em: 23 mai 2024.

ALONSO, Andres. **A Falha de SQL Injection - Com Pedro Nogueira [Vídeo].** Disponível em: https://www.youtube.com/watch?v=PJ_ovUGFfY8&t=925s. Acesso em: 03 mai. 2024.

AUGUSTENE, Agnė. **O que é injeção SQL? Exemplos e como se proteger**. 2023. Disponível em: https://nordvpn.com/pt-br/blog/o-que-e-injecao-sql/. Acesso em: 07 jun. 2024.

BERNERS-LEE, T.; NIELSEN, H.; FIELDING, R. T. **Hypertext Transfer Protocol - HTTP/1.0.** RFC Editor, 1996. RFC 1945. (Request for Comments, 1945). Disponível em: https://rfc-editor.org/rfc/rfc1945.txt. Acesso em: 07 jun. 2024

BUCKBEE, Michael. **Como usar o Wireshark**: tutorial completo e dicas. 2022. Disponível em: https://www.varonis.com/#:~:text=O%20Wireshark%20é%20uma%20ferramenta%20de%20análise%20de%20pacotes%20e,Bluetooth%2C%20sem%20fio%20(IEEE). Acesso em: 31 mai. 2024.

CERT.BR (São Paulo). **Cartilha de Segurança para Internet.** 2. ed. São Paulo: Isbn, 2012.

CONTROLE NET. **Sniffer**: o que é um analisador de pacotes e protocolos de rede. 2023. Disponível em: https://www.controle.net/faq/sniffer-o-que-e-um-analisador-de-protocolos. Acesso em: 31 mai. 2024.

COVIL HACKER. **Tutorial de Brute Force em sites com Nmap e Hydra [Vídeo].** Disponível em: https://www.youtube.com/watch?v=IbDAHDSlgYg. Acesso em: 09 mai. 2024.

DAITYARI, Shaumik. **Injeção SQL:** Um Guia Detalhado para Usuários do WordPress. Kinsta 2023. Disponível em: https://kinsta.com/pt/blog/injecao-sql/#:~:text=Os%20ataques%20de%20Inje%C3%A7%C3%A3o%20SQL,resultados%20de%20consultas%20maliciosas%20injetadas.&text=Nessa%20consulta%2C%20o%20operador%20UNION,SELECT%20username%2C%20password%20FROM%20user_table. Acesso em: 21 mai. 2024.

DAMELE, B.; MIROSLAV, S. **SQLMAP**: Ferramenta automática de injeção SQL e controle de banco de dados. 2023. Disponível em: https://sqlmap.org/. Acesso em: 28 mai. 2024.

ECOTRUST. **SQL Injection:** o que é, como se proteger e exemplos. 2023. Disponível em: https://blog.ecotrust.io/sql-injection/. Acesso em: 16 mai. 2024.

FERNANDES, Mirian. **BRUTE FORCE:** tudo o que você precisa saber! 2022. Disponível em: https://blog.starti.com.br/brute-force/. Acesso em: 07 jun. 2024.

FOROUZAN, Behrouz A. **Comunicação de Dados e Redes de Computadores**. 4. ed. Porto Alegre: AMGH Editora Ltda, 2010. 1145 p.

FORTINET. **O que é ataque de força bruta?** 2023. Disponível em: https://www.fortinet.com/br/resources/cyberglossary/brute-force-attack#:~:text=Um%20ataque%20de%20força%20bruta,sistemas%20e%20redes%20de%20organizações. Acesso em: 16 mai. 2024.

GANGAN, Subodh. **A Review of Man-in-the-Middle Attacks**. 2015. Disponível em: https://arxiv.org/pdf/1504.02115. Acesso em: 23 mai. 2024.

GEEKSFORGEEKS. **Explique o funcionamento do HTTPS.** 2024. Disponível em: https://www.geeksforgeeks.org/explain-working-of-https/. Acesso em: 07 jun. 2024.

GEEKSFORGEEKS. **O que é mitigação de ataques?** 2022. Disponível em: https://www.geeksforgeeks.org/what-is-attack-mitigation/?ref=lbp. Acesso em: 07 jun. 2024.

GIGLER, T.; GLAS, B.; SMITHLINE, N.; STOCK, A. **OWASP Top Ten**. 2021. Disponível em: https://owasp.org/. Acesso em: 23 mar. 2024.

IBSEC, Blog. **SQLMAP:** Ferramenta para Detecção e Exploração de Vulnerabilidades SQL. Disponível em: https://ibsec.com.br/sqlmap-ferramenta-para-deteccao-e-exploracao-de-vulnerabilidades-sql/. Acesso em: 28 mai. 2024.

KUROSE, J.; ROSS, K. **Redes de computadores e a internet:** uma abordagem top-down. 6. ed. São Paulo: Pearson, 2014. 658 p.

OWASP. **Manipulator-in-the-middle attack**. 2023. Disponível em: https://owasp.org/www-community/attacks/Manipulator-in-the-middle_attack. Acesso em: 21 mai. 2024.

OWASP. **SQL Injection.** 2023. Disponível em: https://owasp.org/www-community/attacks/SQL_Injection. Acesso em: 16 mai. 2024.

PACKETLABS. **How Does SQL Injection Impact Clients?** 2023. Disponível em: https://www.packetlabs.net/posts/how-does-sql-injection-impact-clients/. Acesso em: 16 mai. 2024

PAULINO, Antônio. **Ataques de Força Bruta:** medidas de proteção e mitigação. 2020. Disponível em: https://www.sidechannel.blog/ataques-de-forca-bruta-medidas-de-protecao-e-mitigacao/. Acesso em: 07 jun. 2024.

PELÁEZ, Bladimir. **Wireshark**: a ferramenta indispensável para análise de rede. 2024. Disponível em: https://community.revelo.com.br/wireshark-a-ferramenta-indispensavel-para-analise-de-rede/. Acesso em: 31 maio 2024.

PENTESTTV. **Stealing Passwords Using Wireshark and Ettercap [Vídeo].** Disponível em: https://www.youtube.com/watch?v=VSPHsgPEPqc. Acesso em: 03 mai. 2024.

QUORA. **Quais são as vulnerabilidades de segurança no HTTP?** 2023. Disponível em: https://www.quora.com/What-are-the-security-vulnerabilities-in-HTTP. Acesso em: 03 jun. 2024.

SHIVANANDHAN, Manish. **O que é o Nmap e como usá-lo**: um tutorial para a melhor ferramenta de varredura de todos os tempos. 2023. Disponível em: https://www.freecodecamp.org/portuguese/news/o-que-e-o-nmap-e-como-usa-lo-um-tutorial-para-a-melhor-ferramenta-de-varredura-de-todos-os-tempos/. Acesso em: 29 mai. 2024.

SIMONELYTE, Miglė. **O que são ataques de força bruta e como evitá-los?** 2023. Disponível em: https://nordvpn.com/pt-br/blog/ataque-de-forca-bruta/. Acesso em: 07 jun. 2024.

STALLINGS, William. **Criptografia e segurança de redes**: princípios e práticas. 4. ed. São Paulo: Pearson Prentice Hall, 2008. 494 p.

STEFANELLO, Lucas Adiers. **4 Casos de Empresas que tiveram Falhas na Segurança do Site**. 2023. Disponível em: https://incuca.net/4-casos-de-empresas-que-tiveram-falhas-na-seguranca-do-site/. Acesso em: 16 mai. 2024.

WAZLAWICK, R. S. **Metodologia de Pesquisa para Ciência da Computação.** Rio de Janeiro: Elsevier, 2014.