

Universidade Unicesumar

Leonardo de Lima Póss

Desafio Profissional:
Vulnerabilidades em
Aplicações WEB

Curitiba,

2023

Leonardo de Lima Póss

Desafio Profissional:
Vulnerabilidades em
Aplicações WEB

Trabalho apresentado ao
curso de engenharia de
software como requisito para
obtenção de nota.

Professora Ana Paula

Curitiba,

2023

Sumário:

Sumário:	3
Introdução:	4
Contexto e justificativa do trabalho prático:	4
Objetivos:	4
Metodologia:	4
Ambiente Virtual:	4
Instalação e configuração do ambiente virtual:	4
Instalação e Configuração do Linux na Máquina Virtual:	5
Instalação e Configuração do WebGoat:	5
Visão Geral do WebGoat:	5
Descrição e Funcionalidades do WebGoat:	5
Como acessar e navegar no WebGoat:	6
Práticas Comuns de Segurança em Aplicações Web:	6
Conceitos Básicos de Aplicações de Segurança:	6
Identificação de Vulnerabilidades Comuns em Aplicações Web:	7
Boas Práticas para Mitigação de Vulnerabilidades em Aplicações Web:	8
SQL Injection:	8
Tarefas dentro do WebGoat:	9
Introduction:	9
Básico HTTP:	10
Ferramentas de Desenvolvedor:	11
Tríade Cia:	12
Conclusão:	12
Síntese de Resultados e Conclusões do Trabalho Prático:	12
Limitações do Trabalho e Sugestões Para Trabalhos Futuros:	13
Referencias Bibliográficas:	13

Introdução:

Contexto e justificativa do trabalho prático:

Com o tema de cibersegurança como foco principal da disciplina, fora escolhida a atividade de criar um ambiente virtual com SO (Sistema Operacional), Linux para que assim através das ferramentas e funcionalidades disponibilizadas pela ‘WebGoat’ fossem utilizadas de forma a não acarretar riscos aos computadores pessoais dos alunos ao realizarem a tarefa, sua principal justificativa é ter o mínimo de experiência prática no âmbito de vulnerabilidades em cibersegurança, com ênfase em: SQL injection. As atividades realizadas serão “Introduction” e “General” exceto as atividades contidas em “General” HTTP Proxies e Writing New Lesson, também será realizada a atividade (A3) Injection com referência ao SQL Injection (Intro).

Objetivos:

Criar, configurar e utilizar uma máquina virtual que execute o sistema operacional Kali Linux, afim de experienciarmos a exploração de vulnerabilidades em ambiente controlado.

Metodologia:

Para criação do ambiente virtual fora utilizado o software VirtualBox-64bits além de usar o SO indicado nas instruções de confecção da atividade, Kali Linux e a ferramenta qBitTorrent para a realização do download do SO, para o fim de estudar as vulnerabilidades fora escolhido o SQL Injection e como ferramenta de estudo fora utilizado a WebGoat dentro do ambiente virtual.

Ambiente Virtual:

Instalação e configuração do ambiente virtual:

A instalação e configuração do ambiente virtual seguiu sem dificuldades, somente com o tutorial fornecido nas instruções de confecção da atividade.

Instalação e Configuração do Linux na Máquina Virtual:

Na instalação do Linux houveram algumas dificuldades, em instalar corretamente o sistema em uma primeira tentativa fora instalado uma versão sem interface gráfica, o que levou a criar um novo ambiente virtual para a instalação com interface gráfica, outra dificuldade foi no momento de escolher a “partição de disco” o que também levou a mais tarde criar um novo ambiente virtual e mudar as configurações de instalação até que finalmente houve sucesso, já na instalação do JRE, não teve problemas e seguir as instruções fornecidas já foi o bastante.

Instalação e Configuração do WebGoat:

A instalação do WebGoat foi o que mais houve dificuldades, visto que o link fornecido não obteve sucesso na instalação e teve a procura de um que estivesse funcional até que fora encontrado no endereço virtual acessível pelo link seguinte: “<https://github.com/WebGoat/WebGoat/releases/download/v2023.4/webgoat-2023.4.jar>“, que obteve sucesso em sua instalação, o comando “Java -jar webgoat-container-8.1.0-war-exec.jar” foi alterado para “Java -jar webgoat-2023.4.jar”, desta forma a instalação e configuração foi bem sucedida.

Visão Geral do WebGoat:

Descrição e Funcionalidades do WebGoat:

O WebGoat é um projeto de aplicativo web intencionalmente inseguro usado para fins educacionais e de treinamento em segurança da web. Ele foi desenvolvido para ajudar os desenvolvedores e profissionais de segurança a entenderem e explorarem as vulnerabilidades comuns encontradas em aplicações web.

Vulnerabilidades intencionais: O WebGoat possui uma série de lições e exercícios que demonstram várias vulnerabilidades de segurança da web, como injeção de SQL, Cross-site scripting (XSS), ataques de força bruta, entre outros.

Aprendizado prático: Os usuários podem interagir com o WebGoat através de um navegador da web e realizar exercícios que envolvem a exploração e a exploração das vulnerabilidades presentes na aplicação.

Feedback e orientação: O WebGoat fornece feedback detalhado sobre as ações do usuário, orientando-os sobre como as vulnerabilidades podem ser exploradas e como se proteger contra elas.

Ampliação de conhecimento: O projeto WebGoat inclui uma ampla variedade de lições e cenários, cobrindo várias vulnerabilidades e técnicas de ataque comuns, ajudando os usuários a expandirem seu conhecimento sobre segurança da web.

Como acessar e navegar no WebGoat:

Após baixar e instalar o WebGoat em um servidor local, você pode acessá-lo por meio de um navegador da web, digitando o endereço local onde o aplicativo está sendo executado. Ao criar uma conta de usuário (se necessário), você poderá explorar as lições disponíveis, que abrangem diversas vulnerabilidades de segurança. O WebGoat fornece instruções e orientações para realizar exercícios práticos, permitindo que os usuários experimentem diferentes técnicas de ataque e aprendam sobre as consequências de suas ações.

Práticas Comuns de Segurança em Aplicações Web:

Conceitos Básicos de Aplicações de Segurança:

A segurança em aplicações web é um aspecto fundamental para proteger dados, usuários e sistemas contra ameaças e vulnerabilidades. Alguns conceitos básicos são essenciais para garantir a segurança em aplicações web. A autenticação, por exemplo, é o

processo de verificar a identidade do usuário antes de conceder acesso a recursos protegidos. Já a autorização controla as permissões e os privilégios dos usuários autenticados. É importante implementar mecanismos seguros de autenticação e autorização, como senhas fortes, autenticação em duas etapas e gerenciamento adequado de sessões.

Outro conceito crucial é a proteção contra ataques de injeção, como a injeção de SQL e de código. É recomendado utilizar consultas parametrizadas ou instruções preparadas e evitar a concatenação direta de dados de entrada em consultas ou comandos. Além disso, é necessário proteger contra ataques de Cross-Site Scripting (XSS) e Cross-Site Request Forgery (CSRF), por meio da implementação de filtros de entrada, escape de saída e tokens CSRF.

O gerenciamento de erros e exceções também desempenha um papel importante na segurança. É crucial evitar o fornecimento de informações detalhadas sobre erros ao usuário final, implementando um tratamento adequado de erros e registrando-os de forma segura. Manter o software atualizado, aplicar patches de segurança e utilizar criptografia para proteger a comunicação e dados sensíveis são práticas indispensáveis.

Além disso, é recomendado realizar testes de segurança, como testes de penetração e varreduras de vulnerabilidades, para identificar possíveis brechas e corrigi-las antes que sejam exploradas.

Identificação de Vulnerabilidades Comuns em Aplicações Web:

A identificação de vulnerabilidades comuns em aplicações web é de extrema importância para garantir a segurança dos sistemas. Dentre as vulnerabilidades mais frequentes, destacam-se: injeção de SQL, onde dados não confiáveis são inseridos incorretamente em consultas SQL; Cross-Site Scripting (XSS), que permite a inserção de scripts maliciosos em páginas web; Cross-Site Request Forgery (CSRF), um ataque que engana o navegador do usuário para realizar ações indesejadas em um site; exposição de dados sensíveis, que ocorre quando informações confidenciais são armazenadas ou transmitidas de forma insegura; autenticação fraca, envolvendo senhas fracas ou

autenticação não segura; gerenciamento inadequado de sessões, que pode levar a ataques de sequestro de sessão; falta de validação de entrada, permitindo a execução de códigos maliciosos; configuração inadequada do servidor, expondo informações sensíveis; inclusão de arquivos não confiáveis, possibilitando a inserção de códigos maliciosos; e falta de controle de acesso, permitindo acesso não autorizado a informações ou funcionalidades restritas.

Boas Práticas para Mitigação de Vulnerabilidades em Aplicações Web:

Para mitigar vulnerabilidades em aplicações web, é crucial adotar boas práticas de segurança. Dentre as recomendações mais importantes estão: manter o software atualizado, incluindo sistema operacional, servidores web, frameworks e bibliotecas utilizados; utilizar o princípio do "menor privilégio" para limitar o acesso a recursos sensíveis; validar e filtrar rigorosamente a entrada de dados para evitar injeção de código malicioso; empregar criptografia para proteger informações sensíveis e garantir a segurança da comunicação; implementar autenticação segura, com senhas fortes e autenticação em duas etapas; aplicar controle de acesso granular para que cada usuário tenha acesso apenas ao necessário; gerenciar corretamente as sessões, utilizando tokens de sessão seguros e encerrando-as adequadamente; realizar testes de segurança regulares, como testes de penetração e varreduras de vulnerabilidades; proteger-se contra ataques de CSRF com mecanismos de proteção; e manter registros de atividades e monitorar a aplicação para detectar tentativas de intrusão e comportamentos suspeitos. Essas boas práticas ajudam a fortalecer a segurança das aplicações web e reduzir o risco de exploração de vulnerabilidades.

SQL Injection:

A SQL Injection é uma vulnerabilidade comum em aplicações web que ocorre quando dados não confiáveis são inseridos incorretamente em consultas SQL. Essa falha de segurança permite que um invasor execute comandos maliciosos no banco de dados, podendo manipular, modificar ou excluir dados, obter informações sensíveis e até mesmo assumir o controle total do sistema. O ataque de SQL Injection explora a falta de validação e sanitização adequadas dos dados de entrada, permitindo a inserção de trechos de código SQL não autorizados. Para evitar essa vulnerabilidade, é essencial utilizar práticas seguras de programação, como a utilização de consultas parametrizadas ou a adoção de ORM

(Object-Relational Mapping), que tratam os dados de forma automática e segura, prevenindo a injeção de código SQL malicioso. Além disso, a implementação de filtros e validações de entrada adequados é fundamental para garantir que somente dados confiáveis e válidos sejam processados pelas consultas SQL, protegendo assim o sistema contra ataques de SQL Injection.

Tarefas dentro do WebGoat:

Introduction:

Atividade 1:

Try it; type in your e-mail address below and check your inbox in WebWolf. Then type in the unique code from the e-mail in the field below.

✓


@ leonardo@gmail.com

Send e-mail

Type in your unique code

Go

✓



Congratulations. You have successfully completed the assignment.

Com o uso de uma caixa de Email próprio da WebGoat, foi inserido o nome de usuário para um Email fictício e foi enviado um código único, que nesta ocasião consistiu no nome de usuário escrito ao contrário.

Atividade 2:

Suppose we tricked a user into clicking on a link he/she received in an email. This link will open up our crafted password reset link page. The user does not notice any differences compared to the normal password reset page of the company. The user enters a new password and hits enter. The new password will be sent to your host. In this case, the new password will be sent to WebWolf. Try to locate the unique code.

Please be aware that the user will receive an error page after resetting the password. In a real attack scenario the user would probably see a normal success page (this is due to a limit on what we can control with WebWolf)


✓

[Click here to reset your password](#)

Type in your unique code

Go

✓



Congratulations. You have successfully completed the assignment.

A atividade tinha como foco, demonstrar como links falsos de mudança de senha funcionam, o hyperlink presente na atividade redirecionava a uma página de alteração

9

de senha, que quando efetuada não resultava em nada além de deixar explícita a senha na barra de pesquisa.

Básico HTTP:

Atividade 1:

1 2 3

Enter your name in the input field below and press "Go!" to submit. The server will accept the request, reverse the input and display it back to the user, illustrating the basics of handling an HTTP request.

Try It!

Enter your name in the input field below and press "Go!" to submit. The server will accept the request, reverse the input and display it back to the user, illustrating the basics of handling an HTTP request.

✓

Enter Your Name: Go!

The server has reversed your name: odranoel

Com intuito de demonstrar o manejo básico de HTTP, foi inserido o nome “Leonardo” e como resposta a atividade retornou o nome escrito ao contrário.

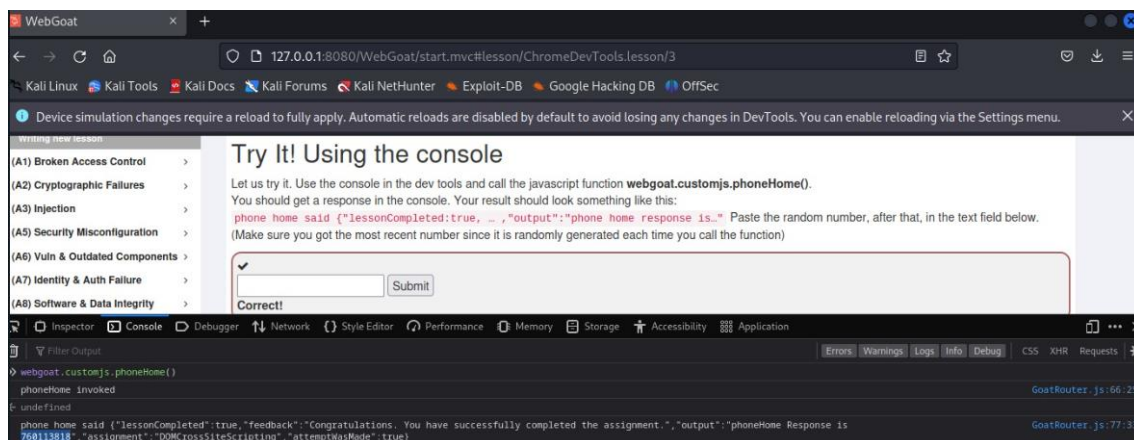
Atividade 2:

The screenshot shows a web browser window with the URL `127.0.0.1:8080/WebGoat/start.mvc#lesson/HttpBasics.lesson/2`. The page title is "The Quiz". The quiz question is: "What type of HTTP command did WebGoat use for this lesson. A POST or a GET." The user has selected "POST". The next question is: "What is the magic number:" with the input field containing "72". The "Go!" button is visible. Below the quiz, a message says: "Congratulations. You have successfully completed the assignment." The bottom of the screenshot shows the DevTools Network tab, displaying a list of requests. The first request is a GET request to `lessonoverview.mvc` with a status of 200. The second request is a GET request to `lessonmenu.mvc` with a status of 200. The third request is a GET request to `lessonoverview.mvc` with a status of 200. The fourth request is a GET request to `lessonmenu.mvc` with a status of 200. The fifth request is a GET request to `lessonoverview.mvc` with a status of 200. The sixth request is a GET request to `lessonmenu.mvc` with a status of 200. The seventh request is a GET request to `lessonoverview.mvc` with a status of 200. The eighth request is a GET request to `lessonmenu.mvc` with a status of 200. The ninth request is a GET request to `lessonoverview.mvc` with a status of 200. The tenth request is a GET request to `lessonmenu.mvc` with a status of 200. The eleventh request is a GET request to `lessonoverview.mvc` with a status of 200. The twelfth request is a GET request to `lessonmenu.mvc` with a status of 200. The thirteenth request is a GET request to `lessonoverview.mvc` with a status of 200. The fourteenth request is a GET request to `lessonmenu.mvc` with a status of 200. The fifteenth request is a GET request to `lessonoverview.mvc` with a status of 200. The sixteenth request is a GET request to `lessonmenu.mvc` with a status of 200. The seventeenth request is a GET request to `lessonoverview.mvc` with a status of 200. The eighteenth request is a GET request to `lessonmenu.mvc` with a status of 200. The nineteenth request is a GET request to `lessonoverview.mvc` with a status of 200. The twentieth request is a GET request to `lessonmenu.mvc` with a status of 200. The twenty-first request is a GET request to `lessonoverview.mvc` with a status of 200. The twenty-second request is a GET request to `lessonmenu.mvc` with a status of 200. The twenty-third request is a GET request to `lessonoverview.mvc` with a status of 200. The twenty-fourth request is a GET request to `lessonmenu.mvc` with a status of 200. The twenty-fifth request is a GET request to `lessonoverview.mvc` with a status of 200. The twenty-sixth request is a GET request to `lessonmenu.mvc` with a status of 200. The twenty-seventh request is a GET request to `lessonoverview.mvc` with a status of 200. The twenty-eighth request is a GET request to `lessonmenu.mvc` with a status of 200. The twenty-ninth request is a GET request to `lessonoverview.mvc` with a status of 200. The thirtieth request is a GET request to `lessonmenu.mvc` with a status of 200. The thirty-first request is a GET request to `lessonoverview.mvc` with a status of 200. The thirty-second request is a GET request to `lessonmenu.mvc` with a status of 200. The thirty-third request is a GET request to `lessonoverview.mvc` with a status of 200. The thirty-fourth request is a GET request to `lessonmenu.mvc` with a status of 200. The thirty-fifth request is a GET request to `lessonoverview.mvc` with a status of 200. The thirty-sixth request is a GET request to `lessonmenu.mvc` with a status of 200. The thirty-seventh request is a GET request to `lessonoverview.mvc` with a status of 200. The thirty-eighth request is a GET request to `lessonmenu.mvc` with a status of 200. The thirty-ninth request is a GET request to `lessonoverview.mvc` with a status of 200. The fortieth request is a GET request to `lessonmenu.mvc` with a status of 200. The forty-first request is a GET request to `lessonoverview.mvc` with a status of 200. The forty-second request is a GET request to `lessonmenu.mvc` with a status of 200. The forty-third request is a GET request to `lessonoverview.mvc` with a status of 200. The forty-fourth request is a GET request to `lessonmenu.mvc` with a status of 200. The forty-fifth request is a GET request to `lessonoverview.mvc` with a status of 200. The forty-sixth request is a GET request to `lessonmenu.mvc` with a status of 200. The forty-seventh request is a GET request to `lessonoverview.mvc` with a status of 200. The forty-eighth request is a GET request to `lessonmenu.mvc` with a status of 200. The forty-ninth request is a GET request to `lessonoverview.mvc` with a status of 200. The fiftieth request is a GET request to `lessonmenu.mvc` with a status of 200. The fifty-first request is a GET request to `lessonoverview.mvc` with a status of 200. The fifty-second request is a GET request to `lessonmenu.mvc` with a status of 200. The fifty-third request is a GET request to `lessonoverview.mvc` with a status of 200. The fifty-fourth request is a GET request to `lessonmenu.mvc` with a status of 200. The fifty-fifth request is a GET request to `lessonoverview.mvc` with a status of 200. The fifty-sixth request is a GET request to `lessonmenu.mvc` with a status of 200. The fifty-seventh request is a GET request to `lessonoverview.mvc` with a status of 200. The fifty-eighth request is a GET request to `lessonmenu.mvc` with a status of 200. The fifty-ninth request is a GET request to `lessonoverview.mvc` with a status of 200. The sixtieth request is a GET request to `lessonmenu.mvc` with a status of 200. The sixty-first request is a GET request to `lessonoverview.mvc` with a status of 200. The sixty-second request is a GET request to `lessonmenu.mvc` with a status of 200. The sixty-third request is a GET request to `lessonoverview.mvc` with a status of 200. The sixty-fourth request is a GET request to `lessonmenu.mvc` with a status of 200. The sixty-fifth request is a GET request to `lessonoverview.mvc` with a status of 200. The sixty-sixth request is a GET request to `lessonmenu.mvc` with a status of 200. The sixty-seventh request is a GET request to `lessonoverview.mvc` with a status of 200. The sixty-eighth request is a GET request to `lessonmenu.mvc` with a status of 200. The sixty-ninth request is a GET request to `lessonoverview.mvc` with a status of 200. The seventieth request is a GET request to `lessonmenu.mvc` with a status of 200. The seventy-first request is a GET request to `lessonoverview.mvc` with a status of 200. The seventy-second request is a GET request to `lessonmenu.mvc` with a status of 200. The seventy-third request is a GET request to `lessonoverview.mvc` with a status of 200. The seventy-fourth request is a GET request to `lessonmenu.mvc` with a status of 200. The seventy-fifth request is a GET request to `lessonoverview.mvc` with a status of 200. The seventy-sixth request is a GET request to `lessonmenu.mvc` with a status of 200. The seventy-seventh request is a GET request to `lessonoverview.mvc` with a status of 200. The seventy-eighth request is a GET request to `lessonmenu.mvc` with a status of 200. The seventy-ninth request is a GET request to `lessonoverview.mvc` with a status of 200. The eightieth request is a GET request to `lessonmenu.mvc` with a status of 200. The eighty-first request is a GET request to `lessonoverview.mvc` with a status of 200. The eighty-second request is a GET request to `lessonmenu.mvc` with a status of 200. The eighty-third request is a GET request to `lessonoverview.mvc` with a status of 200. The eighty-fourth request is a GET request to `lessonmenu.mvc` with a status of 200. The eighty-fifth request is a GET request to `lessonoverview.mvc` with a status of 200. The eighty-sixth request is a GET request to `lessonmenu.mvc` with a status of 200. The eighty-seventh request is a GET request to `lessonoverview.mvc` with a status of 200. The eighty-eighth request is a GET request to `lessonmenu.mvc` with a status of 200. The eighty-ninth request is a GET request to `lessonoverview.mvc` with a status of 200. The ninetieth request is a GET request to `lessonmenu.mvc` with a status of 200. The ninety-first request is a GET request to `lessonoverview.mvc` with a status of 200. The ninety-second request is a GET request to `lessonmenu.mvc` with a status of 200. The ninety-third request is a GET request to `lessonoverview.mvc` with a status of 200. The ninety-fourth request is a GET request to `lessonmenu.mvc` with a status of 200. The ninety-fifth request is a GET request to `lessonoverview.mvc` with a status of 200. The ninety-sixth request is a GET request to `lessonmenu.mvc` with a status of 200. The ninety-seventh request is a GET request to `lessonoverview.mvc` with a status of 200. The ninety-eighth request is a GET request to `lessonmenu.mvc` with a status of 200. The ninety-ninth request is a GET request to `lessonoverview.mvc` with a status of 200. The hundredth request is a GET request to `lessonmenu.mvc` with a status of 200.

Com o comando POST e o auxílio do console de inspeção, foi possível realizar a atividade, o objetivo era de conseguir o valor de um “número mágico”.

Ferramentas de Desenvolvedor:

Atividade 1:



Inserido o comando de Javascript ‘webgoat.customjs.phonehome()’ no console, foi possível obter o número de telefone fictício que era o objetivo da atividade (760113818).

Atividade 2:



Também com o auxílio do console de inspeção, com o objetivo de obter o numero aleatório, foi necessário clicar no botão ‘GO’ e procurar as requisições dentro da inspeção ‘NETWORK’, desta maneira foi obtido o número referido na imagem.

Tríade Cia:

Atividade de Questionário:

1. How could an intruder harm the security goal of confidentiality?

- ☐ Solution 1: By deleting all the databases.
- ☐ Solution 2: By stealing a database where general configuration information for the system is stored.
- ☒ Solution 3: By stealing a database where names and emails are stored and uploading it to a website.
- ☐ Solution 4: Confidentiality can't be harmed by an intruder.

2. How could an intruder harm the security goal of integrity?

- ☒ Solution 1: By changing the names and emails of one or more users stored in a database.
- ☐ Solution 2: By listening to incoming and outgoing network traffic.
- ☐ Solution 3: By bypassing authentication mechanisms that are in place to manage database access.
- ☐ Solution 4: Integrity can only be harmed when the intruder has physical access to the database storage.

3. How could an intruder harm the security goal of availability?

- ☐ Solution 1: By exploiting bugs in the systems software to bypass authentication mechanisms for databases.
- ☐ Solution 2: By redirecting emails with sensitive data to other individuals.
- ☐ Solution 3: Availability can only be harmed by unplugging the power supply of the storage devices.
- ☒ Solution 4: By launching a denial of service attack on the servers.

4. What happens if at least one of the CIA security goals is harmed?

- ☐ Solution 1: A system can be considered safe until all the goals are harmed. Harming one goal has no effect on the systems security.
- ☒ Solution 2: The systems security is compromised even if only one goal is harmed.
- ☐ Solution 3: It's not that bad when an attacker reads or changes data, at least some data is still available, hence only when the goal of availability is harmed the security of the system is compromised.
- ☐ Solution 4: It shouldn't be a problem if an attacker changes data or makes it unavailable, but reading sensitive data is not tolerable. There's only a problem when confidentiality is harmed.

Conclusão:

Síntese de Resultados e Conclusões do Trabalho Prático:

Após todo o processo de instalação e configuração, com foco na realização de algumas atividades direcionadas à vulnerabilidade SQL Injection, foi realizada 5 atividades de injeção dentro do ambiente virtual através do WebGoat, as atividades envolviam extrair informações das tabelas presentes, alterar dados, conceder e remover permissões a usuários desautorizados para o caso de conceder e usuários que antes possuíam as permissões para os casos de remover tais permissões.

Limitações do Trabalho e Sugestões Para Trabalhos Futuros:

Como já abordados durante o desenvolvimento do documento, as dificuldades se deram na instalação do SO, e na configuração do mesmo além de haver a dificuldade de baixar e acessar o WebGoat, tanto em relação a ter de procurar um novo link funcional quanto na execução dos comandos no terminal. A atividade HTTP Proxies referida na introdução apresentou diversas complicações não só na máquina virtual como na máquina física, esquentou mais que o habitual e travou as telas, tendo esses tipos de complicações, a atividade em questão não pode ser realizada, já a atividade Writting New Lesson, também referida na introdução possui um grau grande de complexidade, além das capacidades que são detidas atualmente pelo autor do trabalho, tendo isso em foco a atividade teve sua realização evitada.

Referencias Bibliográficas:

OWASP WebGoat Project. Disponível em:
https://www.owasp.org/index.php/Category:OWASP_WebGoat_Project. Acesso em: 19 de maio de 2023.

WebGoat. Disponível em: <https://github.com/WebGoat/WebGoat>. Acesso em: 19 de maio de 2023.

OWASP. SQL Injection. Disponível em: https://owasp.org/www-community/attacks/SQL_Injection. Acesso em: 19 de maio de 2023.

OWASP. Cross-Site Scripting (XSS). Disponível em: <https://owasp.org/www-community/attacks/xss/>. Acesso em: 19 de maio de 2023.

OWASP. Cross-Site Request Forgery (CSRF). Disponível em:
<https://owasp.org/www-community/attacks/csrf>. Acesso em: 19 de maio de 2023.

OWASP. Disponível em: <https://owasp.org/www-community/>. Acesso em: 19 de maio de 2023.

OWASP Top Ten Project. Disponível em: <https://owasp.org/www-project-top-ten/>. Acesso em: 19 de maio de 2023.

Microsoft Docs. SQL Injection. Disponível em: <https://docs.microsoft.com/en-us/sql/relational-databases/security/sql-injection?view=sql-server-ver15>. Acesso em: 19 de maio de 2023.

W3Schools. SQL Injection. Disponível em:
https://www.w3schools.com/sql/sql_injection.asp. Acesso em: 19 de maio de 2023.

PortSwigger. SQL Injection. Disponível em: <https://portswigger.net/web-security/sql-injection>. Acesso em: 19 de maio de 2023.