**Segurança da Informação - Vulnerabilidades de aplicações SQLi**

**Vander de Oliveira Rocha Junior**

**David Coelho do Nascimento Silva**

**Resumo**

Essa pesquisa visa a exploração de falhas de segurança, e extração de dados sensíveis das aplicações, utilizando o método de *SQL Injection* .

Nesta pesquisa, será abordado os diferentes tipos de *SQL Injection* explicando a teoria e com maior enfoque na prática. Na teoria serão abordados os diferentes métodos relacionados a *SQL Injection ,* na prática será executado a injeção de códigos maliciosos em aplicações vulneráveis, demonstrando quais as formas são utilizadas para explorar este tipo de vulnerabilidade. Os testes serão realizados em três aplicações diferentes que já apresentam este tipo de vulnerabilidade, há uma análise comparativa entre um código vulnerável e um seguro, finalizando com uma seção de mitigação e boas práticas, para manter as aplicações seguras a este tipo de vulnerabilidade. Durante todo o teste, será obtido as informações sigilosas das bases de dados das aplicações, como nomes de usuários, senhas, números de cartões de créditos e versões do ambiente.

This research aims to explore security vulnerabilities, specifically in the area of cybersecurity and pentesting, using the SQL Injection method.

In our research, we approach the different types of SQL Injection explaining the theory and with a focus on practice. In theory, the different methods related to SQL Injection are addressed, in practice, malicious code is injected into vulnerable applications, demonstrating which ways are used to exploit this type of vulnerability. The tests are carried out in three different applications that present this type of vulnerability, there is a comparative analysis between a vulnerable and a safe code, ending with a section on mitigation and best practices, to keep applications safe for this type of vulnerability.

**1-Introdução**

O intuito desta pesquisa é abordar os aspectos em que envolve a segurança e as vulnerabilidades de uma aplicação SQL, com o diferencial de apresentar uma abordagem prática com injeção de código SQL em três ambientes que apresentam vulnerabilidades, conhecido como “SQL Injection'' que tem como comportamento a injeção de comando em campos vulneráveis a fim de serem interpretados e respondidos pela aplicação. Será abordado três sub métodos contidos no tema “SQL Injection'' bem como suas características e modo de funcionamento.

Este projeto se baseia em técnicas que são utilizadas em alguns livros, como no livro de Weidman (2014), nos sites de estudos que podem ser encontradas informações e ambientes simulados para testes, como nos sites Hack the Box ¹, Desecurity ² e Udemy ³. Essa pesquisa visa explicar a abordagem dos testes de segurança, os passos para se iniciar e executar com sucesso a exploração de SQL Injection. Todo o conteúdo aqui descrito é um conjunto dos estudos realizados nas contribuições teóricas dos sites e livros citados, com o objetivo de tornar prática toda a teoria encontrada nas fontes citadas.

¹ Disponível em <<https://www.hackthebox.com/>> Acesso em: 24 agosto, 2022.

² Disponível em <<https://desecsecurity.com/>> Acesso em: 24 agosto, 2022.

³ Disponível em <[https://udemy.com/](https://desecsecurity.com/)> Acesso em: 24 agosto, 2022.

De forma geral, uma exploração *SQL injection*, como o próprio nome sugere, é um teste realizado em aplicações da internet que utilizam banco de dados. Uma exploraçãoconsiste em testes de segurança em aplicações, para identificar e explorar vulnerabilidades. O objetivo de uma exploração *SQL injection,* é injetar consultas SQL nas aplicações, de forma que altere a requisição realizada entre a aplicação e o banco de dados, podendo assim, extrair informações sigilosas e até mesmo alterar os dados ali contidos.

Uma exploração de *SQL injection* se torna uma forma de precaução, que avalia os riscos e suas consequências, tornando possível realizar medidas de proteção para evitar possíveis ataques e vazamento de informações, identificando vulnerabilidades que, muitas vezes, passam despercebidas durante o desenvolvimento de aplicações.

Com a crescente demanda por tecnologia, e constantes ataques cibernéticos, buscamos abordar este tema, com o intuito de fornecer uma introdução prática, executando as etapas necessárias para se realizar uma exploração *SQL injection,* utilizando os métodos adequados e essenciais para exploração.

**Aviso Legal**

Os testes aqui mostrados foram realizados em um ambiente seguro, criados pelos próprios autores deste trabalho sem violar nenhuma lei. Ao realizar os testes aqui registrados sem a devida permissão, há a possibilidade de responder judicialmente, violando as leis (Lei Carolina Dieckmann - 12737, Lei Marco Civil da Internet - 12965 e Lei Geral de Proteção de Dados - 13709). Foram exploradas três aplicações que tem como finalidade o estudo e práticas relacionadas à cibersegurança. Essas aplicações podem ser baixadas na internet gratuitamente para utilizá-las como base de estudos, mais informações sobre as aplicações utilizadas podem ser encontradas no tópico 3.2.

Não recomendamos a execução destes testes em sites da internet sem a prévia permissão de seus responsáveis, pois como citado, há leis específicas relacionadas a invasão e roubo de informações, podendo assim, o executante responder judicialmente por invasão, violação e roubo de dados pessoais.

**1.1 - Objetivos**

**1.1.1- Objetivo Geral**

O intuito da realização do projeto é descrever os passos e métodos utilizados para a realização de uma exploração de falhas de segurança. Apresentar uma visão geral de como funciona uma exploração de *SQL injection*, sua elaboração e execução dos testes. Mostrar de forma clara, como se inicia a exploração*,* explicando a teoria e aplicando as técnicas.

Utilizaremos ambientes virtuais, a fim de validar, testar e divulgar neste artigo os resultados gerados na pesquisa. Complementando com uma visão geral da exploração de vulnerabilidades, abordando a teoria envolvida e apresentando a prática aplicada no nosso projeto.

**1.1.2- Objetivos Específicos**

Este projeto tem dois objetivos específicos, explicar os três diferentes tipos de injeção sql e exemplificar as técnicas. A ideia por trás desta exploração de vulnerabilidades, é buscar falhas de segurança nas aplicações e injetar códigos SQL maliciosos, que invalidam verificações da aplicação e manipulam as requisições entre a aplicação e a base de dados.

**1.2- Justificativa**

Com a grande demanda por tecnologia, um grande volume de dados e informações sensíveis são armazenadas em servidores, a integridade e segurança desses dados se tornam cada vez mais essenciais para que não ocorra acessos indevidos e vazamentos de informações.

Como a exploração desta vulnerabilidade se encontra no top três da lista OWASP Top Ten, será mostrado de que maneira os atacantes extraem informações e como se prevenir.

**1.3- OWASP - Open Web Application Security Project**

A OWASP é uma fundação sem fins lucrativos que trabalha para melhorar a segurança do software. Com diversos projetos de código aberto liderados pela comunidade e membros espalhados por todo o mundo, com diversas conferências e treinamentos. A Owasp é a fonte para profissionais da área de TI em geral para proteger a web, um dos documentos mais importantes desenvolvidos pela fundação, é o OWASP Top Ten. O OWASP Top Ten é um documento que busca conscientizar o desenvolvimento de softwares de forma segura, ele representa um consenso sobre as 10 vulnerabilidades mais críticas e comuns em aplicações web. A vulnerabilidade *SQL Injection* é a terceira das principais listadas no OWASP Top Ten **2- Referencial Teórico**

De acordo com Giavaroto e Santos (2013), Não é novidade que houve um grande aumento no acesso à tecnologia da informação e a inclusão digital. Praticamente tudo pode ser feito pela internet, como: compras, pesquisas, estudos, reuniões e entre outros. Com o rápido e constante avanço da tecnologia, muitas vulnerabilidades surgem, e há diversas formas que auxiliam na busca de vulnerabilidades e falhas de segurança. Evidenciando como se faz necessário a utilização de ferramentas para identificar e corrigir problemas relacionados à segurança em sistemas de informação.

No momento atual, em que a internet abrange diversos setores, e se torna necessária em diversos negócios, o papel da segurança da informação é crucial. Proteger informações privilegiadas é um critério primário, e os métodos mais usados de autenticação de usuários utilizam sistemas que validam o usuário, através de login e senha. No entanto, a exigência de senhas seguras pode trazer alguns problemas, como o usuário esquecer a senha. É comum um usuário esquecer suas credenciais de acessos, e ter que solicitar redefinição de acesso por conta própria. Desta forma, se faz necessário manter um ambiente seguro e simples, para que os usuários possam interagir facilmente com os sistemas. A interação de dois ambientes tão diferentes que precisam interagir: o ambiente tecnológico e o ambiente humano, nos traz uma validação sobre a reflexão do comportamento humano sobre a segurança da informação, para questionar a validade e eficácia dos responsáveis por lidar com essas questões.(SILVA, 2007).

Em seu artigo Bertoglio e Zorzo (2016) abordam uma constante em empresas. Organizações e entidades que têm contido dados sensíveis, a preocupação com a segurança dessas informações uma vez que a perda deles causem perdas significativas as instituições. Visando essa necessidade de assegurar a integridades destes dados foi desenvolvido Testes de segurança que, por sua vez, efetuam técnicas para averiguar o nível de ameaça em que os dados estão dispostos. Dentre essas técnicas, a de Penetração (Pentest) se destaca, uma vez que este simula literalmente um ataque ao um sistema, mas com os devidos cuidados e sigilos dos dados obtidos, o processo como um todo deve ser registrado a fim de mensurar todas as brechas de segurança mensurar seus níveis de verificações entre outros.

Testes de segurança para empresas e organizações se demonstra interessante uma vez que este simula uma situação muito próxima da realidade, juntando esse fato com o acúmulo de informação obtidos com os testes e constatando a equipe de profissionais responsáveis por esta área, as medidas de soluções se tornam muito eficazes para cada tipo de contexto. A metodologia Tramonto é um framework para gerenciamento de pentest que auxilia outras metodologias de pentest, uma vez que flexibiliza os testes que normalmente são específicos para cada contexto. (BERTOGLIO e ZORZO, 2016).

Em seu artigo, Maciel (2019) tratou sobre a atual situação contemporânea do contexto dos serviços, que em sua totalidade se utilizam de danos sensíveis que oferecem serviço ágeis e eficientes oferecidos pelas empresas. Com esses serviços, vieram também os cibercriminosos que aproveitam de falhas nos sistemas para prejudicar as empresas. Com a ideia que não existe sistema cem por cento seguro, surgiram técnicas para diminuir o risco de empresas terem seus dados sensíveis vazados, prejudicando o negócio. Uma dessas técnicas é o Pentest tem como objetivo identificar as vulnerabilidades dos sistemas, através de invasões com a finalidade de identificar brechas de segurança. O objetivo foi a identificação de vulnerabilidades no servidor OJS (*Open Journal System*), por meio de técnicas de Pentest como: scanners Acunetix, e teste de penetração manual. O scanners Acunetix é uma ferramenta que replica as técnicas utilizadas por invasores, localizando vulnerabilidades. Após os testes o sistema se mostrou seguro, ressaltando uma de suas características a de ser código aberto, com fóruns de discussões feitos pelo PKP (*Public Knowledge Project*), que apontam falhas e expõe possíveis soluções pelos usuários, essa característica acompanhada pelo *feedback* tornou o servidor seguro, após análises e implementação de sugestões.

**3- Metodologia**

Os testes serão realizados em um ambiente seguro, sem violar nenhuma lei. Através de uma máquina cliente, serão realizados os testes de *SQL injection* em um servidor que hospeda três diferentes aplicações web vulneráveis. Essas aplicações foram projetadas para profissionais e estudantes da área de cibersegurança, para aplicarem técnicas e encontrar vulnerabilidades. A máquina cliente que será utilizada para realizar os testes possui o sistema operacional Kali Linux, e o servidor com as aplicações e base de dados possui o sistema operacional Ubuntu. O método utilizado será a identificação de campos vulneráveis, manipulação de consultas e extração de informações.

**3.1 -**

Existem três tipos de *SQL Injection*, a diferença entre eles é a forma como extraímos informações ou manipulamos banco de dados. Iremos abordar os três tipos, porém, o foco para gerar resultados será em explorar apenas o primeiro tipo, pois é o mais comum e pode ser realizado sem a necessidade de ferramentas automatizadas.

**3.1.1 SQL In-Band/Classic -** É um dos métodos mais comuns de explorar. Este método é utilizado quando utilizamos o mesmo canal para realizar o ataque e obter os resultados. Os resultados são exibidos diretamente na aplicação. O SQL *Classic* possui duas técnicas mais conhecidas:

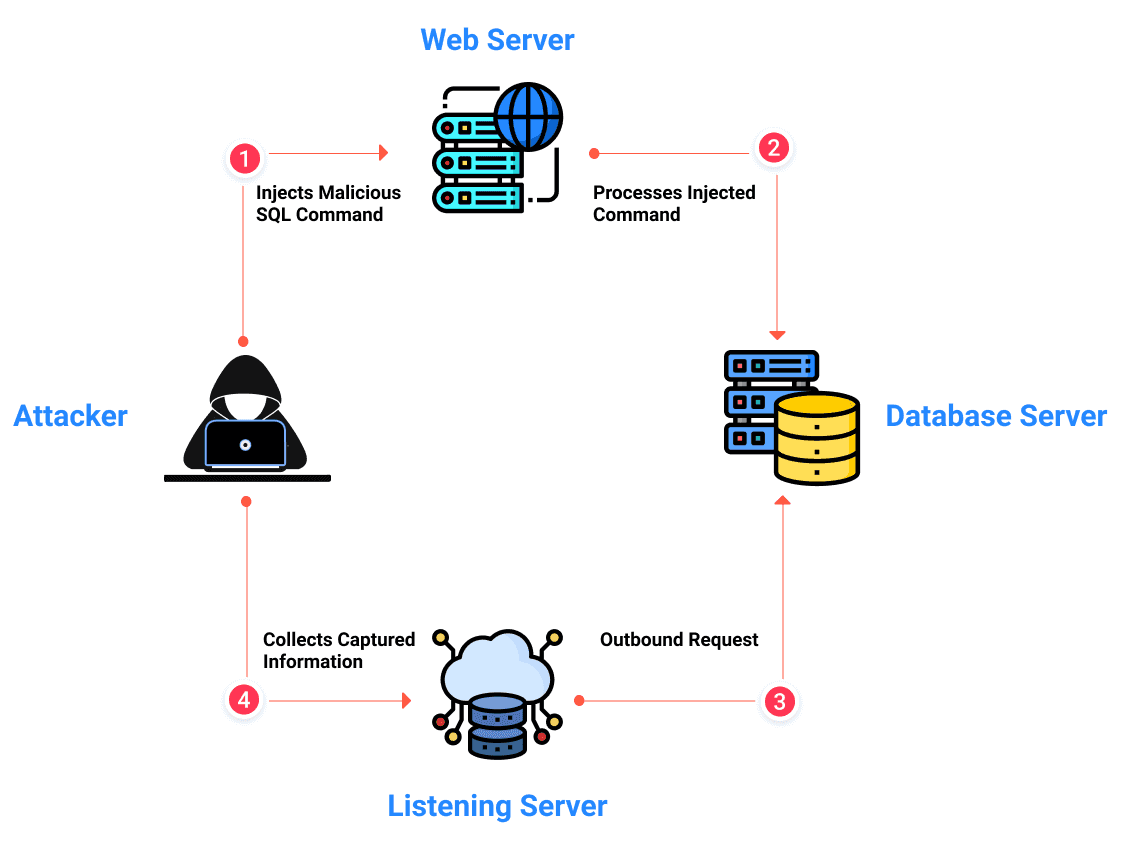
* Error Based SQLi - É uma forma de enumerar a base de dados, buscando informações sobre sua estrutura baseando-se nas mensagens de erro enviadas pelo servidor do banco de dados.
* Union Based SQLi - Consiste em utilizar o operador UNION da linguagem SQL para retornar consultas adicionais dentro da mesma requisição.

**3.1.2 SQL Inferential/Blind -** É um método mais complexo de se explorar. Os resultados não são exibidos diretamente na aplicação web. Neste método, após injetar o sql, avaliamos o comportamento da aplicação e através de seu comportamento podemos extrair informações. As 2 técnicas mais conhecidas deste método são:

* Blind-Boolean-Based - Consiste em enviar uma consulta SQL do tipo booleana para o banco de dados que força a aplicação a retornar dados diferentes, dependendo do resultado da query ser verdadeiro ou falso. Ao validar o retorno da consulta sendo verdadeiro ou falso, o conteúdo da resposta HTTP vai ser diferente ou permanecer o mesmo, é um método demorado porque é necessário o mapeamento do nome de uma coluna ou registro do banco de dados, caractere a caractere.
* Blind-Time-Based - Seguindo o mesmo conceito da técnica anterior, o comportamento da aplicação é analisado, porém, é enviado um *delay* no momento da injeção de sql que força o banco de dados esperar um tempo específico antes de retornar a resposta. Dependendo da resposta ser verdadeira ou falsa, o conteúdo da resposta HTTP é retornada instantaneamente ou com o tempo de *delay* que foi injetado na consulta.

**3.1.3 SQLi Out Of Band -** Este método não é muito comum pois necessita de funcionalidades ativas no banco de dados da aplicação alvo. Se faz necessário que o servidor do banco de dados possua as configurações de requisição DNS e HTTP para que suas informações possam ser extraídas e enviadas a um outro servidor. São os pacotes UTL\_HTTP do banco de dados ORACLE, utilizado para fazer requisições HTTP e o comando XP\_DIRTREE do Microsoft SQL Server, que permite o servidor de banco de dados fazer requisições DNS a outro servidor. Este método consiste em injetar um código malicioso em um dos inputs de usuário da aplicação, para fazer uma requisição DNS para o servidor controlado pelo *hacker*, ou seja, o servidor da aplicação processa a requisição e realiza uma comunicação com o servidor que é controlado pelo *hacker*, como é exibido na imagem 1.

Figura 1: Esquema SQL out of band.



Fonte:<sibermetin.com/sql-injection-nedir> Acesso em: 16 setembro, 2022.

**3.2 - Aplicações vulneráveis**

As aplicações abaixo foram selecionadas porque simulam um ambiente real. Possuem suas próprias bases de dados e foram projetadas para servir como alvo de ataques.

**Mutillidae** é uma aplicação open-source que foi desenvolvida pela própria OWASP, para estudantes e profissionais interessados na área de cibersegurança, é uma aplicação que contém diversas vulnerabilidades listadas na *OWASP Top Ten*. Muito utilizada para treinamentos em cursos e por profissionais de cibersegurança. Pode ser baixada diretamente no site: https://github.com/webpwnized/mutillidae

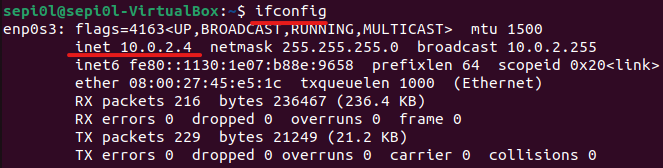
**BWAPP** é uma aplicação de código aberto que possui mais de 100 vulnerabilidades, abrangendo todas as vulnerabilidades listadas pelo *OWASP Top Ten.* Essa aplicação web é destinada a todos que buscam aplicar seus conhecimentos em busca de vulnerabilidades e como preveni-las. Pode ser baixada diretamente no site: <https://github.com/lmoroz/bWAPP>

**DVWA** é uma aplicação web que possui diversas vulnerabilidades, destinada a profissionais de cibersegurança que buscam testar suas habilidades e ferramentas em um ambiente legalizado. Possui três níveis de segurança para dificultar a extração de informações da aplicação. Pode ser baixada no site: https://github.com/digininja/DVWA

**4 - Resultado**

Iniciando os testes, primeiramente se faz necessário ter o endereço da aplicação web. Ao ter acesso ao endereço do servidor, podemos acessar suas aplicações, como é mostrado na imagem 2.

Figura 2: Endereço IP

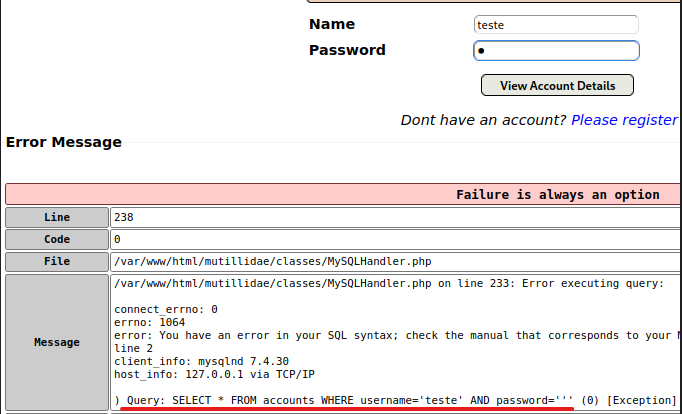


**Fonte**: Produzido pelos autores do artigo.

**4.1 - Mutillidae**

Ao acessar a aplicação mutillidae do servidor, vamos a tela que traz informações do usuário. Nesta tela, os usuários inserem login e senha para visualizar suas informações. buscamos uma vulnerabilidade do tipo In-band usando o método error-based, que consiste em inserir um caractere que possa quebrar a query e retornar algum erro por parte do banco de dados. Exemplo na imagem 3

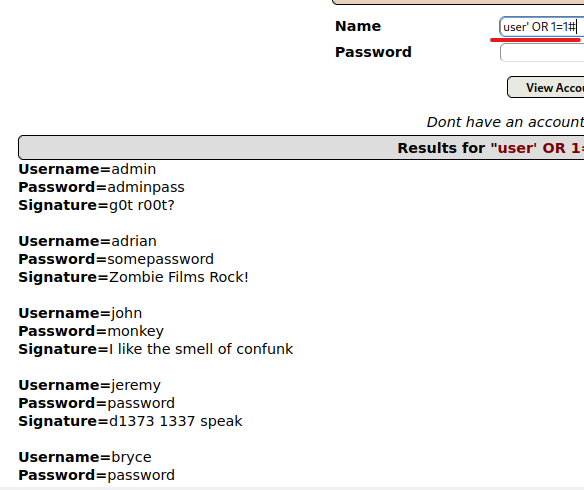
Figura 3: Página de usuários



**Fonte**: Produzido pelos autores do artigo.

Ao inserir no campo Name o valor ‘teste’ e no campo Password uma aspas simples, ocorre um erro na consulta, e pode-se constatar que foi retornado um erro da base de dados, como é mostrado na figura 3. O erro exibe informações até mesmo da consulta que é montada pelo backend. Estes campos não possuem tratamento e suas variáveis são inseridas diretamente na consulta que é feita pelo banco de dados, ou seja, podemos injetar consultas nos inputs vulneráveis. Após confirmar a vulnerabilidade, será inserido códigos sql para manipular a consulta, no campo “name” é inserido o valor: user’ *OR 1=1#* portanto, a query executada pelo backend será: *SELECT \* FROM accounts WHERE username = user’ OR 1=1#'.* Ao inserir as aspas simples, a consulta é encerrada e é adicionado o operador OR a consulta, que faz a validação username = user ou 1=1. Como 1 é igual a 1, isso torna a consulta verdadeira, descartando a validação que é feita pelo backend e retornando os resultados. O símbolo cardinal é utilizado para encerrar a query e ignorar qualquer código ou validação que venha posteriormente, como é mostrado na imagem 4.

Figura 4: Listagem de usuários



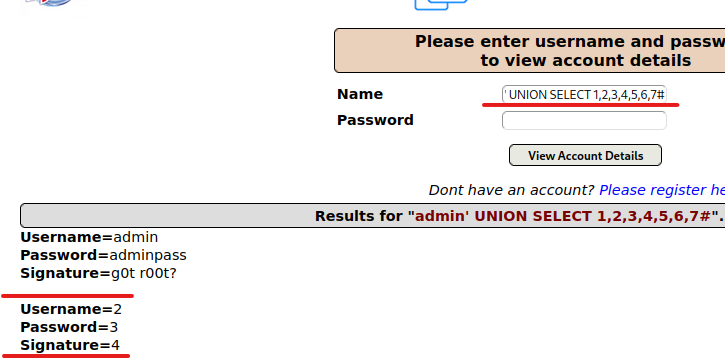
**Fonte**: Produzido pelos autores do artigo.

Foi obtido informações de todos os usuários da tabela, com uma simples alteração no campo input vulnerável.

A imagem 4 exibe três colunas da tabela (username, password e signature), o próximo passo é identificar a quantidade de colunas que possui essa tabela e quais as colunas são renderizadas pela aplicação. Com essas informações, será utilizado o operador UNION do sql para manipular a consulta. A query a seguir substitui a renderização da aplicação por números, identificando quais os campos a aplicação renderiza. Foi inserido a seguinte query no input vulnerável :

* admin' UNION SELECT 1,2,3,4,5,6,7#

Figura 5: Colunas renderizadas

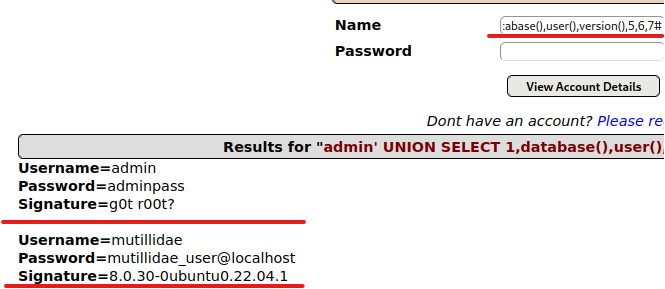


**Fonte**: Produzido pelos autores do artigo.

Como mostrado na imagem 5, essa consulta foi feita apenas para identificar as colunas que são renderizadas. Como vemos, as colunas renderizadas pela aplicação são as colunas 2, 3 e 4. Utilizamos funções sql para nos trazer informações do ambiente, substituindo as colunas 2, 3 e 4 de nossa consulta, como é mostrado na imagem 6:

* admin' UNION SELECT 1,database(),user(),version(),5,6,7#

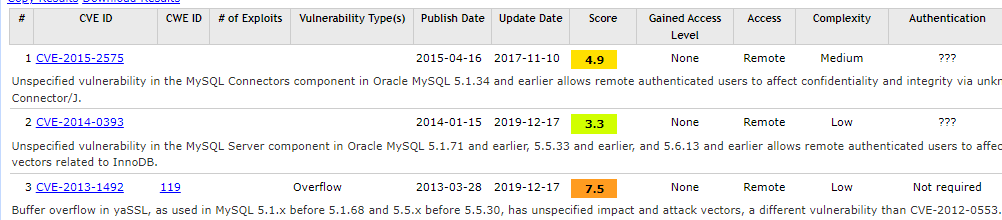
Figura 6: Informações da base de dados



**Fonte**: Produzido pelos autores do artigo.

Esse é um passo essencial em uma auditoria de testes de segurança, pois nesse ponto, é realizada a coleta de informações que podem servir para explorar falhas mais graves. Foi identificamos o nome do banco de dados, nome de usuário da conexão utilizada e qual a versão do banco de dados. Com essas informações, um atacante pode buscar falhas específicas para a versão do banco de dados, diversos sites na internet documentam falhas encontradas em versões de banco de dados. Exemplo na imagem 7.

Figura 7: Falhas CVE



**Fonte**: Produzido pelos autores do artigo.

Exemplo retirado do site CVE Details⁴:

4 Disponível em

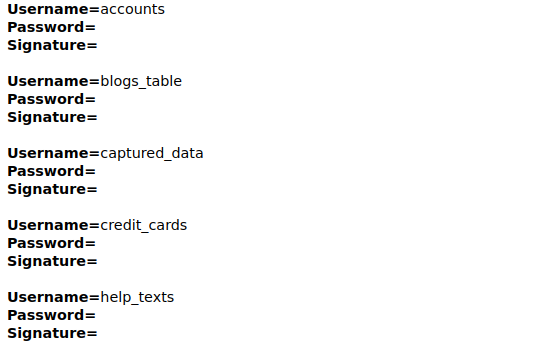
<[www.cvedetails.com/vulnerability-list/vendor\_id-185/product\_id-316/Mysql-Mysql.html](http://www.cvedetails.com/vulnerability-list/vendor_id-185/product_id-316/Mysql-Mysql.html)> Acesso 16 novembro, 2022.

Na consulta anterior, foi identificado que o nome do banco de dados é ‘mutillidae’, com essa informação é possível listar todas as tabelas contidas neste banco de dados utilizando a consulta:

* admin' UNION SELECT 1,table\_name,null,null,5,6,7 FROM information\_schema.tables WHERE table\_schema='mutillidae'#

Essa consulta lista o nome das tabelas onde o nome do banco de dados é ‘mutillidae’, como é mostrado na imagem 8.

Figura 8:Tabelas do banco mutillidae



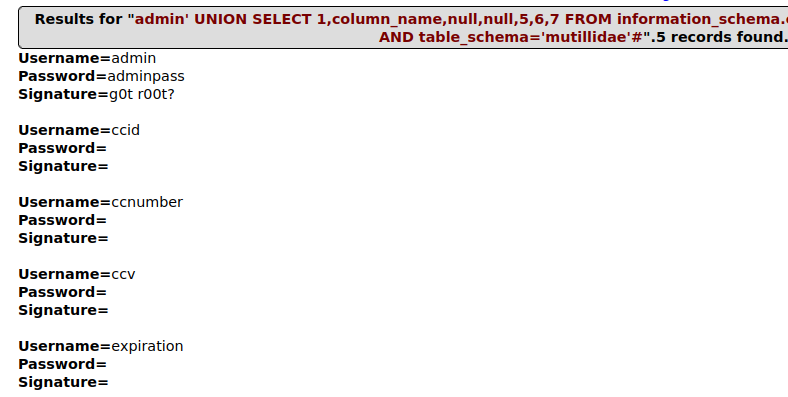
**Fonte**: Produzido pelos autores do artigo.

Há diversas tabelas no banco mutillidae, será extraído informações da tabela "credit cards". primeiramente é necessário listar os nomes de todas as colunas da tabela, através da query:

* admin'UNION SELECT 1,column\_name,null,null,5,6,7 FROM information\_schema.columns WHERE table\_name='credit\_cards' AND table\_schema='mutillidae'#

Essa consulta traz o nome de todas as colunas onde o nome da tabela é “credit\_cards” e o nome do banco “mutillidae”, como é mostrado na imagem 9.

Figura 9:Colunas da tabela credit cards

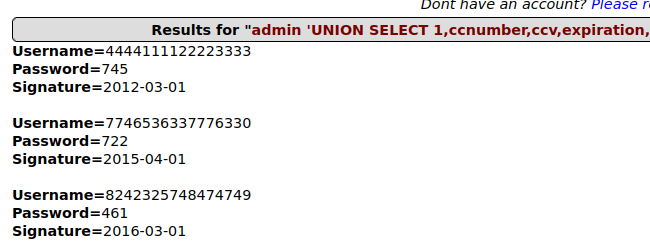


**Fonte**: Produzido pelos autores do artigo.

Essa tabela possui 4 colunas. Sabendo o nome das colunas, é possível extrair as informações ali contidas. Será utilizado a seguinte consulta, como é mostrado na imagem 10

* admin 'UNION SELECT 1,ccnumber,ccv,expiration,5,6,7 FROM credit\_cards#

Figura 10: Dados da tabela credit cards



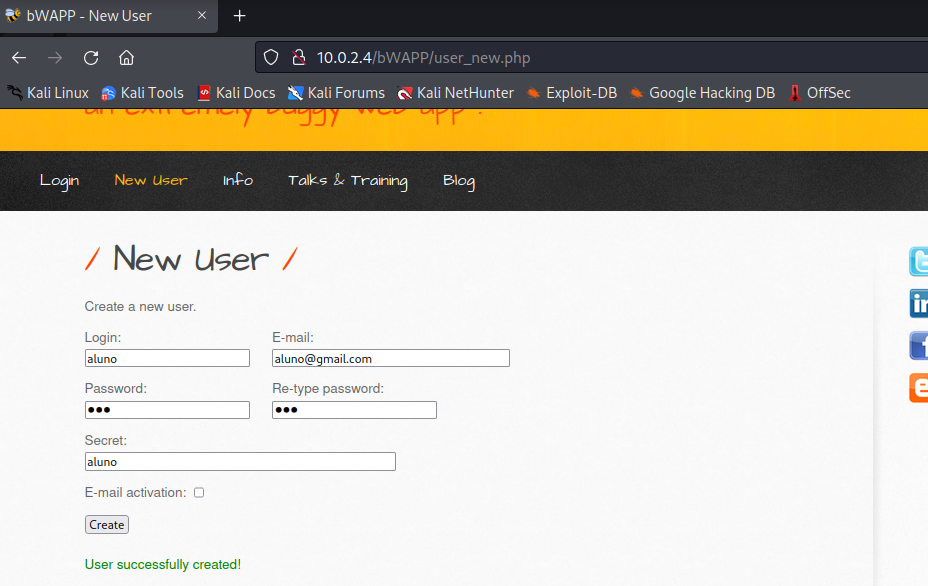
**Fonte**: Produzido pelos autores do artigo.

Com essa consulta, foi extraído o número, ccv e data de expiração dos cartões na tabela credit\_cards.

**4.2 - bWAPP**

Será criado o usuário ‘aluno’ na aplicação bWAPP para acessar suas funcionalidades em busca de vulnerabilidades, como é mostrado na imagem 11.

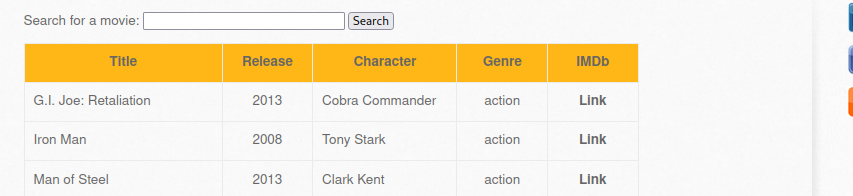
Figura 11: Tela de criação de usuário BWAPP



**Fonte**: Produzido pelos autores do artigo.

Após logar na aplicação, será feito uma busca por vulnerabilidades em um campo de buscas, na tela que faz buscas por nomes de filmes, como é mostrado na imagem 12.

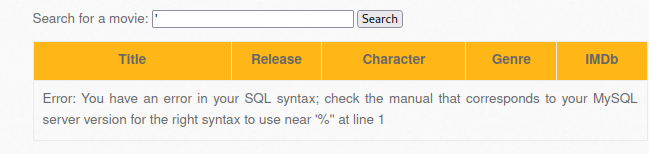
Figura 12: Filtragem de filmes



**Fonte**: Produzido pelos autores do artigo.

Após inserir uma aspas simples no campo de busca, é retornado um erro de consulta, como é mostrado na imagem 13.

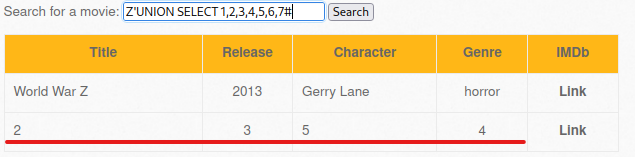
Figura 13: Erro de consulta



**Fonte**: Produzido pelos autores do artigo.

Ao identificar a vulnerabilidade, será realizado os mesmos passos da seção anterior para preparar a injeção de código malicioso. Primeiramente é necessário encontrar o número de colunas da tabela, utilizando o ORDER BY da linguagem sql, após obter o número de colunas da tabela, será utilizado o operador UNION para unir uma consulta extra a requisição e identificar as colunas que são renderizadas na aplicação, como é mostrado na imagem 14.

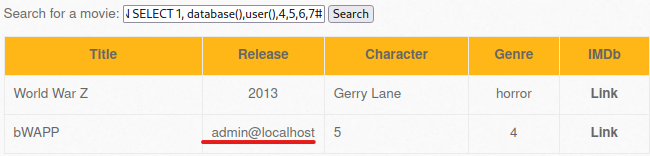
Figura 14: Colunas renderizadas na tela d filmes



**Fonte**: Produzido pelos autores do artigo.

As colunas 2, 3, 5 e 4 são renderizadas, será utilizado essas colunas para extrair informações da base de dados, como o nome do banco de dados e qual a conexão a aplicação utiliza, como mostra na imagem 15.

Figura 15: Nome da base e conexão

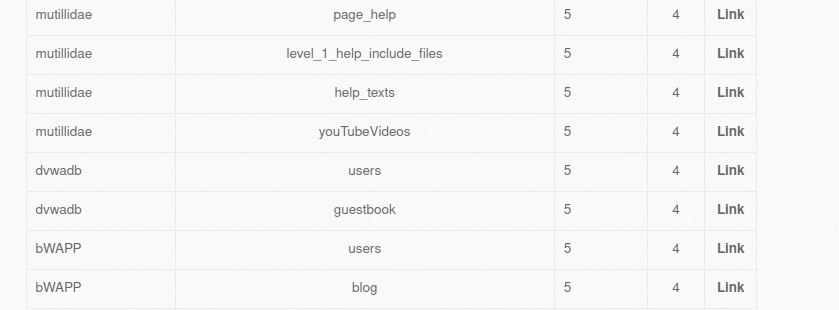


**Fonte**: Produzido pelos autores do artigo.

Foi encontrado um erro extremo, a aplicação utiliza uma conta de administrador para se comunicar com o banco de dados. Se a conta utilizada pela aplicação possui direitos de administrador, um invasor pode ter acesso a outros bancos de dados, ou seja, extrair informações de todas as aplicações do servidor de banco de dados. Será injetado a seguinte consulta, para obter informações de todos os bancos de dados e suas respectivas tabelas, como é mostrado na imagem 16.

* 'UNION SELECT 1, table\_schema,table\_name,4,5,6,7 FROM information\_schema.tables#

Figura 16: Tabelas das bases de dados

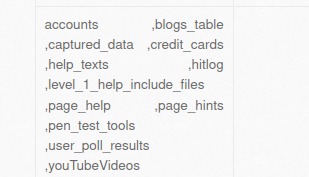


**Fonte**: Produzido pelos autores do artigo.

É possível identificar o nome de todos os bancos de dados das aplicações do servidor, com isso será extraído informações de outras aplicações através da aplicação vulnerável. Isso demonstra que uma única aplicação vulnerável pode expor dados de outras aplicações do servidor. Através da seguinte consulta, será listado todas as tabelas contidas no banco de dados da aplicação mutillidae, como é mostrado na imagem 17.

* 'UNION SELECT 1,GROUP\_CONCAT(table\_name, ' \n'),null,null,5,6,7 FROM information\_schema.tables WHERE table\_schema='mutillidae'#

Figura 17: Tabelas da base mutillidae



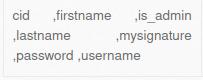
**Fonte**: Produzido pelos autores do artigo.

Para realizar um teste, será extraído informações da tabela accounts com a intenção de realizar login na aplicação Mutillidae com um usuário válido, como é mostrado na imagem 18.

Listagem das colunas da tabela accounts:

* 'UNION SELECT 1,GROUP\_CONCAT(column\_name,'\n'),NULL,NULL,5,6,7 FROM information\_schema.columns WHERE table\_name='accounts' AND table\_schema='mutillidae'#

Figura 18: Colunas da tabela accounts

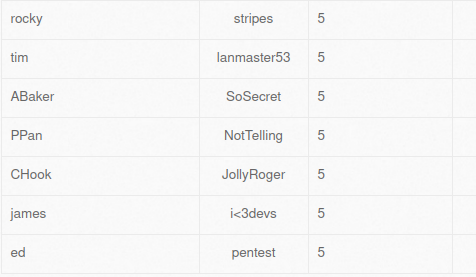


**Fonte**: Produzido pelos autores do artigo.

Na imagem 19, é mostrado os dados da tabela accounts que pertencem a aplicação mutillidae, username e password:

* 'UNION SELECT 1,username, password,4,5,6,7 FROM mutillidae.accounts#

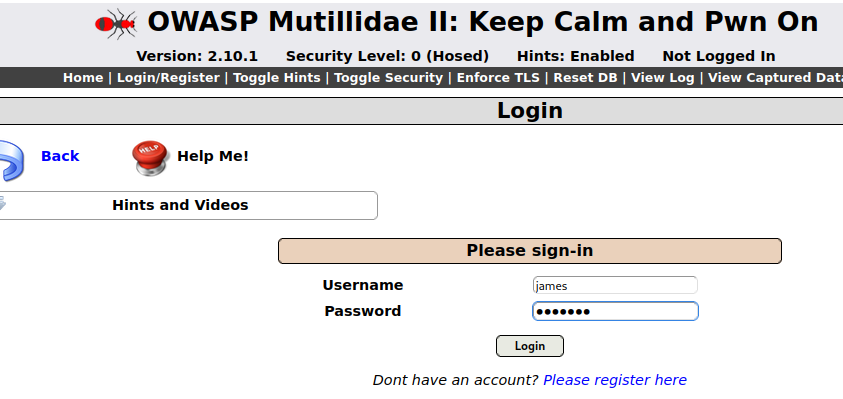
Figura 19: Dados da tabela accounts



**Fonte**: Produzido pelos autores do artigo.

Como é mostrado na imagem 20, será utilizado os dados do usuário ‘james’ para realizar o teste.

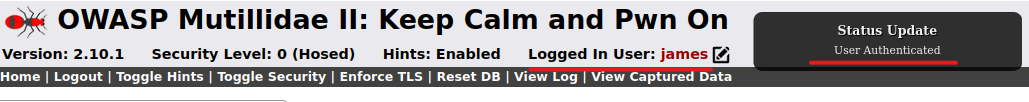
Figura 20: Teste de login



**Fonte**: Produzido pelos autores do artigo.

Foi possível realizar login em outra aplicação com um usuário válido, utilizando os dados extraídos na aplicação BWAPP, como é mostrado na imagem 21.

Figura 21: Login realizado



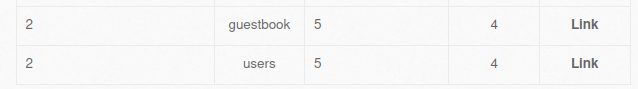
**Fonte**: Produzido pelos autores do artigo.

Será feito o mesmo teste, extraindo informações da base de dados da aplicação DVWA utilizando a seguinte consulta:

* 'UNION SELECT 1, 2,table\_name,4,5,6,7 FROM information\_schema.tables WHERE table\_schema = 'dvwadb'# .

Vemos o resultado na imagem 22:

Figura 22: Tabelas da base DVWA



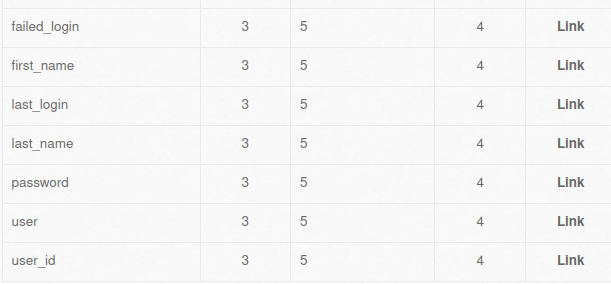
**Fonte**: Produzido pelos autores do artigo.

A aplicação DVWA possui apenas 2 tabelas. Será extraído o nome das colunas da tabela users, injetando o código:

* 'UNION SELECT 1, column\_name,3,4,5,6,7 FROM information\_schema.columns WHERE table\_name='users' AND table\_schema='dvwadb'#

O resultado é mostrado na imagem 23:

Figura 23:Colunas da tabela users



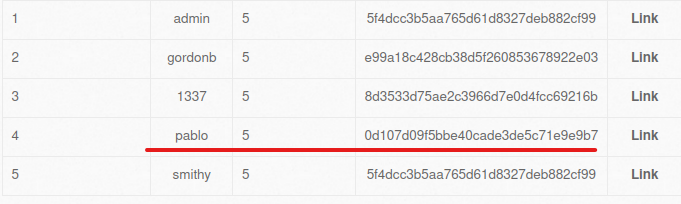
**Fonte**: Produzido pelos autores do artigo.

Sabendo o nome de suas colunas, é possível extrair as informações ao injetar a consulta:

* 'UNION SELECT 1, user\_id,user,password,5,6,7 FROM dvwadb.users#

Resultado é mostrado na imagem 24:

Figura 24: Dados da tabela users

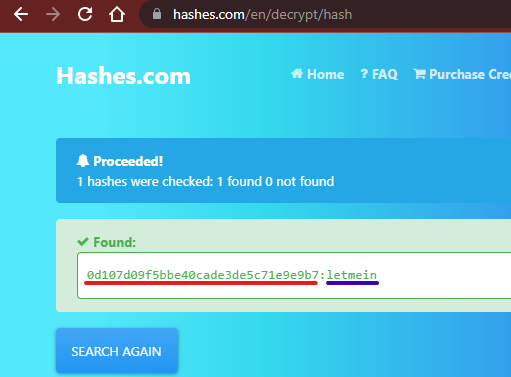


**Fonte**: Produzido pelos autores do artigo.

Essa aplicação utiliza uma boa prática, pois encripta as senhas dos usuários antes de salvar na base de dados. Porém, muitos usuários criam senhas simples, oque permite ao invasor decodificar os hashes por meio de força bruta. Existem muitas ferramentas disponíveis para decodificar hashes. Utilizamos o site Hashes 5 para tentar decodificar a senha do usuário ‘pablo’ e foi possível obter o resultado rapidamente.

O texto que foi sublinhado em vermelho é o hash do usuário que está salvo no banco de dados,e texto que foi sublinhado em azul é a senha que foi extraída do hash, como é mostrado na imagem 25:

Figura 25: Senha desencriptada

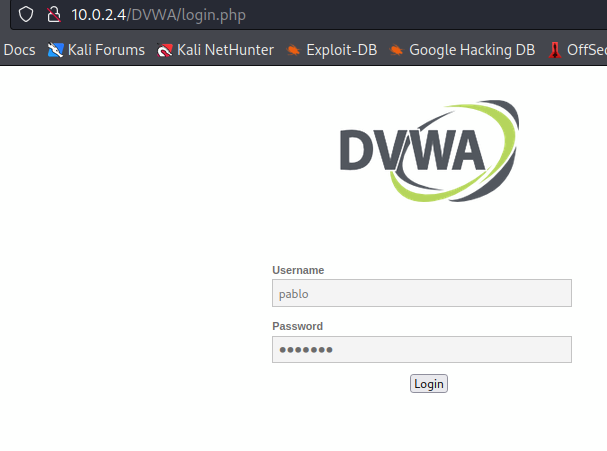


**Fonte**: Produzido pelos autores do artigo.

Ao decodificar o hash, será realizado um teste de login na aplicação DVWA, utilizando o usuário ‘pablo’ e senha ‘letmein’, como é mostrado na imagem 26.

5 Disponível <<https://hashes.com/en/decrypt/hash>> Acesso em: 26 novembro, 2022.

Figura 26: Login DVWA

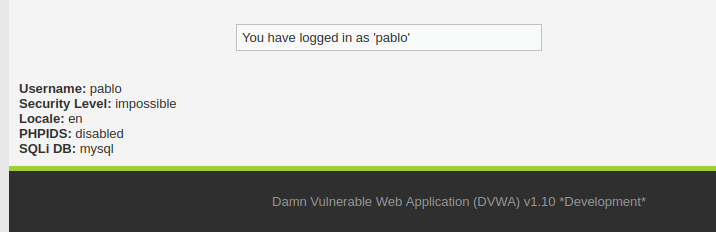


**Fonte**: Produzido pelos autores do artigo.

Como é mostrado na imagem 27, foi possível logar com o usuário e a senha obtida, no rodapé da aplicação é exibido o nome do usuário logado.

Figura 27: Informações de sessão do usuário



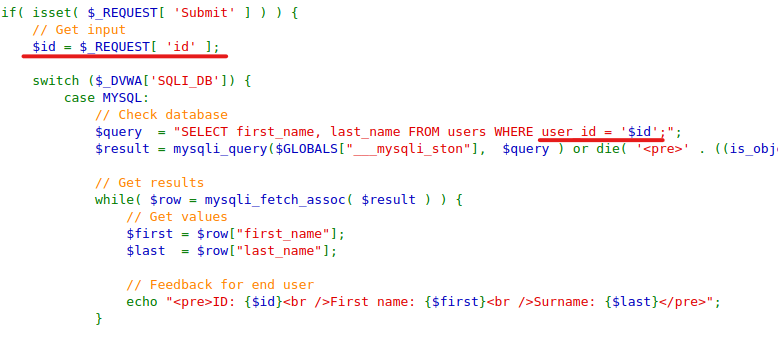


**Fonte**: Produzido pelos autores do artigo.

**4.3 - DVWA**

Não foram encontradas vulnerabilidades relacionadas a sql classic na aplicação DVWA. Como a aplicação se mostrou segura, será utilizada como exemplo para uma demonstração de um código back-end seguro e um vulnerável. O exemplo a seguir, descreve um código vulnerável, onde o input do usuário é inserido diretamente na consulta que será realizada ao banco de dados, como é mostrado na imagem 28.

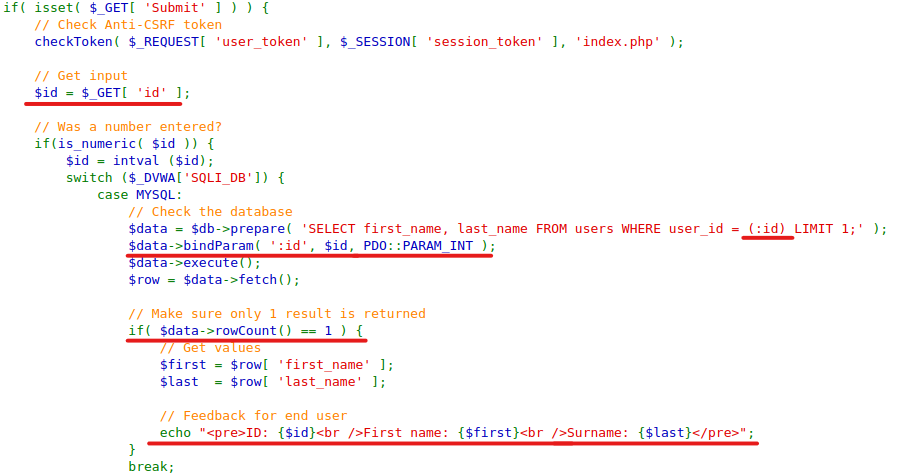
Figura 28: Código vulnerável



**Fonte**: Produzido pelos autores do artigo.

O próximo código, possui as boas práticas para manter o código seguro e evitar que as consultas possam ser manipuladas, como é mostrado na imagem 29.

Figura 29: Código seguro



**Fonte**: Produzido pelos autores do artigo.

O código acima faz uma validação do campo que o usuário digitou, caso ele seja numérico, o valor é armazenado em outra variável do tipo inteiro. A consulta é preparada antes de ser executada, em sua preparação ela recebe um parâmetro e esse parâmetro faz uma ligação com o valor da variável que recebeu o valor digitado pelo usuário. Esse parâmetro de ligação é conhecido como “Bind-param” e no momento que é realizado o bind-param, o back-end aplica uma condição para permitir essa ligação, essa condição é a constante “PDO::PARAM\_INT” que só permite a ligação entre parâmetro e variável se ambas corresponderem ao tipo definido, no caso, valor inteiro.

**5 - Mitigações e boas práticas**

Concluindo, indicamos fortemente que as seguintes mitigações e boas práticas sejam sempre consideradas. Os dados de uma organização são de grande valor e devem estar sempre seguros.

**5.1 - Validar inputs da aplicação**

Validar os campos da aplicação, tanto no front-end como no back-end. No front-end, essa validação pode ser feita para prevenir que o usuário insira dados incorretos ou códigos maliciosos nos campos, por exemplo, um formulário onde o usuário preenche o campo email, deve verificar se o que foi digitado pelo usuário é correspondente a um email. No back-end essa validação pode ser feita através de sanitização, ou seja, os campos da aplicação devem ser validados pelo back-end, e antes de ser feito uma requisição ao banco de dados, as informações digitadas pelo usuário devem ser preparadas de forma a não permitir que essas informações causem danos a aplicação. Os inputs do usuário devem ser tratados sempre, e não devem ser inseridos diretamente na consulta montada pelo back-end.

**5.2 - Tratar erros**

Todo erro que ocorrer no back-end deve ser tratado de forma que não seja renderizado para o usuário. Aplicações que tratam os erros do banco de dados são mais difíceis de identificar vulnerabilidades e explorá-las.

**5.3 - Privilégios de usuários do banco de dados**

No momento da criação de usuário para a aplicação acessar o banco de dados, deve-se conceder o mínimo de privilégios possíveis, permitindo que a aplicação manipule apenas os dados pertencentes a ela. Cada aplicação deve conter seu próprio usuário para acessar o banco de dados, isso previne que, caso a aplicação seja vulnerável, todo o ambiente do servidor seja exposto.

**5.4 - Encriptar informações sensíveis**

Todos os dados sensíveis devem ser encriptados, por exemplo, os passwords dos usuários devem ser encriptados antes de serem salvos no banco de dados, isso permite que, caso ocorra um acesso indevido ao banco de dados, as senhas dos usuários não serão expostas. É uma boa prática concatenar as senhas dos usuários com uma frase segura antes de encriptar as senhas, pois alguns usuários utilizam senhas fracas que podem ser descobertas via força bruta.

**5.5 - Implementar controles de segurança para aplicações web**

Implementar controles de segurança previne diversas vulnerabilidades que podem conter na aplicação. Um exemplo são os WAF(Web Application Firewall), um firewall de aplicação web que monitora e bloqueia o tráfego HTTP de aplicativos e sites da web.

**6 - Resultados**

Ao explorar a vulnerabilidade de injeção SQL nas aplicações, foram extraídos dados sensíveis, com informações de login, números de cartões de créditos e informações do ambiente da base de dados. Com essas informações foi possível realizar acessos indevidos nas aplicações com os dados de outros usuários, extrair informações de outras aplicações e roubar dados sensíveis como os números e códigos dos cartões de créditos que estavam armazenados na base de dados da aplicação. Com a experiência obtida ao realizar a exploração da vulnerabilidade, foi gerado uma lista com as melhores práticas que tornam as aplicações seguras a este tipo de vulnerabilidade.

**Referências**

BERTOGLIO, Daniel Dalalana; ZORZO, A.F. **Tramonto: Uma estratégia de**

**recomendação para testes de penetração**. Anais do SBSeg, 2016.

GIAVAROTO, Sílvio César Roxo; SANTOS, G.R**. Backtrack Linux Auditoria e teste de invasão em redes de computadores**. Ciência Moderna, 2013.

MACIEL, Denys Pereira. **Detecção e análise de vulnerabilidades em um servidor OJS utilizando teste de penetração**. 2019.

DA SILVA, Denise Ranghetti Pilar; STEIN, Lilian Milnitsky. **Segurança da Informação: uma reflexão sobre o componente humano**. Ciências & Cognição, v. 10, 2007.

WEIDMAN, Georgia. **Testes de Invasão: Uma introdução prática ao hacking**. Novatec Editora, 2014.

OWASP, 2022. Disponível em: <https://owasp.org/>. Acesso em: 12/12/2022.

UDEMY, 2022. Disponível em: <https://www.udemy.com/>. Acesso em: 12/12/2022.

Hack the Box, 2022. Disponível em: <https://www.hackthebox.com/>. Acesso em: 12/12/2022.

Documentos do SQL, 2022. Disponível em: <https://learn.microsoft.com/pt-br/sql/?view=sql-server-ver16>. Acesso em: 12/12/2022.

BRASIL. Lei nº 12.737, de 30 de novembro de 2012.

BRASIL. Lei nº 12.965, de 23 de abril de 2014.

BRASIL. Lei nº 13.709, de 14 de agosto de 2018.

|  |
| --- |
|  |