INLIGHN TECH

EXPERIENCE. LEARN. THRIVE

Prerequisites for the SSH Botnet Project

By: **Inlighn Tech**

Overview

The SSH Botnet project requires students to develop a Python script that manages a network of compromised SSH clients (bots) to execute commands, run bash scripts, or launch a DDoS attack. The solution uses pexpect.pxssh for SSH connections, scapy for packet crafting (DDoS), colorama for colored output, json for botnet persistence, and os and sys for system operations, with a menu-driven interface. To write this script independently, students must master several key concepts and skills from the curriculum. These prerequisites, paired with practical Python code examples, ensure students understand how SSH botnets work and can implement the script effectively.

Prerequisite Knowledge and Skills

1. **SSH Fundamentals and Automation**
   - **What to Know**:
     - SSH (Secure Shell) enables remote access using credentials; botnets leverage this to control multiple systems.

- pexpect.pxssh automates SSH interactions, handling login and command execution.
- **How It Applies**:
  - The script uses connect_ssh() to establish SSH sessions with bots and maintain them in the botnet list.
- **Curriculum Source**:
  - "SSH Cracker (Solution)" Lesson 1 ("What is SSH.mp4"); extends to automation.
  - Note: Students need pip install pexpect.
- **Practical Prep**:
  - Test SSH automation with pxssh.
- **Python Code Block**:
-

```
# Basic SSH connection with pxssh
from pexpect import pxssh

host = "127.0.0.1"  # Replace with your SSH server
user = "testuser"   # Replace with valid user
password = "test"   # Replace with valid password
try:
    s = pxssh.pxssh()
    s.login(host, user, password, port=22)
    print(f"Connected to {host}")
    s.logout()
except Exception as e:
    print(f"Error: {e}")
```

-
  - **Run It**: Install pexpect (pip install pexpect), set up a local SSH server (e.g., OpenSSH), save as ssh_test.py, run with python ssh_test.py. Prepares for connect_ssh().

2. **Command Execution Over SSH**
   - **What to Know**:
     - Use sendline() to send commands and prompt() to wait for output; before retrieves the response.
     - Decode bytes to strings for display.
   - **How It Applies**:

        - send_command() executes commands on bot sessions and returns output.
    - **Curriculum Source**:
        - Implied in SSH automation; builds on "SSH Cracker" concepts.
    - **Practical Prep**:
        - Test command execution.
    - **Python Code Block**:

```
# Send a command over SSH
from pexpect import pxssh

host = "127.0.0.1"
user = "testuser"
password = "test"
s = pxssh.pxssh()
s.login(host, user, password, port=22)
s.sendline("whoami")
s.prompt()
output = s.before.decode()
print(f"Output: {output}")
s.logout()
```

    -
        - **Run It**: Save as ssh_cmd_test.py, run with python ssh_cmd_test.py (requires SSH server). Prepares for send_command().

3. **File I/O and JSON Persistence**
    - **What to Know**:
        - Read/write JSON files with json.load() and json.dump() to store botnet data.
        - Use os.system() for system commands (e.g., clear screen).
    - **How It Applies**:
        - save_botnet() and load_botnet() persist and restore the botnet list in botnet.json.
    - **Curriculum Source**:
        - "PYTHON BASICS" Lesson 11 (IO with Basic Files).
        - "WORKING WITH DATA STRUCTURES & FILE HANDLING" Lesson 8 (Working with JSON).

- **Practical Prep**:
  - Test JSON file operations.
- **Python Code Block**:

```
# Save and load JSON data
import json

data = [{"host": "127.0.0.1", "port": 22}]
with open("test.json", "w") as f:
   json.dump(data, f)
with open("test.json", "r") as f:
   loaded = json.load(f)
   print(f"Loaded: {loaded}")
```

-
  - **Run It**: Save as json_test.py, run with python json_test.py. Prepares for save_botnet() and load_botnet().

4. **Exception Handling**
   - **What to Know**:
     - Use try/except to catch SSH connection errors (e.g., authentication failures, timeouts).
     - Return None or skip actions on failure.
   - **How It Applies**:
     - connect_ssh() and other functions handle SSH errors gracefully.
   - **Curriculum Source**:
     - "PYTHON BASICS" Lesson 21 (Errors and Exception Handling).
   - **Practical Prep**:
     - Test error handling with SSH.
   - **Python Code Block**:

```
# Handle SSH connection errors
from pexpect import pxssh

host = "127.0.0.1"
user = "wronguser"
password = "wrong"
try:
    s = pxssh.pxssh()
    s.login(host, user, password, port=22)
    print("Success")
    s.logout()
except Exception as e:
    print(f"Error: {e}")
```

-
    - **Run It**: Save as ssh_error_test.py, run with python ssh_error_test.py. Prepares for robust error handling in connect_ssh().

5. **Menu-Driven Interface**
    - **What to Know**:
        - Use input() for user choices and conditionals (if/elif) to handle options.
        - Loop with while True for continuous interaction.
    - **How It Applies**:
        - The script's main loop and display_menu() provide an interactive botnet control interface.
    - **Curriculum Source**:
        - "PYTHON BASICS" Lessons 5-7 (Conditionals and Loops).
    - **Practical Prep**:
        - Test a simple menu.
    - **Python Code Block**:

```
# Simple menu interface
while True:
    print("1. Option 1\n2. Option 2\n3. Exit")
    choice = input("Choose an option: ")
    if choice == "1":
        print("Selected Option 1")
    elif choice == "2":
        print("Selected Option 2")
    elif choice == "3":
        break
    else:
        print("Invalid choice")
```

- ○
    - **Run It**: Save as menu_test.py, run with python menu_test.py. Prepares for the main loop and display_menu().

6. **Colored Output with colorama**
    - **What to Know**:
        - colorama provides ANSI color codes (e.g., Fore.GREEN) for terminal output; init(autoreset=True) resets colors automatically.
    - **How It Applies**:
        - The script uses colors to highlight success, errors, and bot output.
    - **Curriculum Source**:
        - Not explicitly taught; assumes a brief intro (common in Python projects).
        - Note: Students need pip install colorama.
    - **Practical Prep**:
        - Test colored output.
    - **Python Code Block**:

```
# Use colorama for colored output
from colorama import init, Fore

init(autoreset=True)
print(Fore.GREEN + "Success!")
print(Fore.RED + "Error occurred")
```

- ○
    - **Run It**: Install colorama (pip install colorama), save as color_test.py, run with python color_test.py. Prepares for colored feedback throughout the script.

7. **DDoS Basics with scapy**
    - **What to Know**:
        - scapy crafts and sends network packets; IP(), TCP(), and Raw() build a SYN flood packet; send() transmits it.
        - A DDoS attack floods a target with traffic (e.g., SYN packets) to overwhelm it.

- **How It Applies**:
  - command_for_all() sends a SYN flood to a target IP and port using scapy.
- **Curriculum Source**:
  - "Network Scanner (Solution)" Lessons 2-3 (scapy usage); extends to DDoS.
  - Note: Students need pip install scapy and root/admin privileges.
- **Practical Prep**:
  - Test a simple packet send (requires permission).
- **Python Code Block**:

```
# Send a simple SYN packet with scapy
from scapy.all import *

ip = IP(dst="127.0.0.1")
tcp = TCP(sport=12345, dport=80, flags="S")
packet = ip / tcp
send(packet, verbose=1)
print("Packet sent")
```

-
  - **Run It**: Install scapy (pip install scapy), save as scapy_test.py, run with sudo python scapy_test.py (Linux) or as admin (Windows). Prepares for command_for_all(). **Caution**: Use only on authorized targets.

How the SSH Botnet Works

- **Concept**:
  - An SSH botnet controls multiple compromised systems via SSH to execute commands or launch attacks (e.g., DDoS).
  - This simulates real-world botnet behavior for educational purposes.
- **Workflow**:
  - Load existing bots from botnet.json and reconnect SSH sessions.

    - Display a menu to list bots, run commands, execute bash scripts, add bots, or launch a DDoS attack.
    - Manage bot sessions with pxssh, execute commands, and save the botnet state on exit.
    - Use scapy for a basic SYN flood DDoS attack.
- **Why SSH-Based?**:
    - Leverages SSH's remote access capabilities, common in real botnets targeting misconfigured servers.

How to Write the SSH Botnet Script

Using these prerequisites, students can build the script step-by-step:

1. **Setup**:
    - Import sys, os, json, colorama, pexpect.pxssh, and scapy.all.
2. **SSH Functions**:
    - Write connect_ssh() to establish SSH sessions and send_command() to execute commands.
3. **Botnet Management**:
    - Define botnet_command() to run commands on all bots, add_client() to add bots, and bash() for bash execution.
4. **Persistence**:
    - Implement save_botnet() and load_botnet() for JSON storage and session reconnection.
5. **DDoS**:
    - Create command_for_all() for a SYN flood with scapy.
6. **Interface**:
    - Build display_menu(), ask_for_command(), and the main loop for user interaction.
7. **Main Execution**:
    - Initialize botnet, load it, and run the menu loop.

Notes for Students

- **Setup**: Install dependencies (pip install pexpect colorama scapy). Test on a local SSH server (e.g., 127.0.0.1) or controlled VMs with SSH enabled. Create a botnet.json manually if needed (e.g., [{"host": "127.0.0.1", "port": 22, "user": "test", "password": "test"}]).
- **Engagement**: Run each code block to practice SSH automation, JSON handling, and packet crafting. Test the script with python ssh_botnet.py, adding a local bot (option 4) and running commands (option 2).
- **Tips**: Use a test network or VMs to avoid legal/ethical issues; ensure SSH servers are running (sudo systemctl start ssh on Linux). For DDoS, target a controlled device (e.g., a local router) with permission.

Jump to the SSH Botnet Project Description if you are ready now.