C2 Framework Project Documentation

# Project Proposal

Title: A Modular Command-and-Control (C2) Framework with Evasion and Post-Exploitation Capabilities

Authors: Md. Wahid Sadiq Ratul, Md. Riyad Ahmed

Program: Professional Master's in Information and Cyber Security

University: University of Dhaka

## Abstract

This project focuses on designing and implementing a modular Command-and-Control (C2) framework for offensive cybersecurity simulations. The framework will consist of a customizable C2 server and multiple client-side payloads written in Python and C. The primary goal is to simulate Advanced Persistent Threat (APT)-style behavior in a safe, controlled lab environment. Key features include encrypted communication, modular tasking (e.g., shell access, screenshots), sandbox evasion, and MITRE ATT&CK technique mapping. This framework will serve as both a learning tool and a research contribution to the field of red teaming and adversary emulation.

## Problem Statement

Most existing C2 frameworks used by red teams are either proprietary, heavily monitored by security vendors, or complex for educational use. Additionally, many open-source tools lack built-in evasion techniques and clear modularity for academic research. There is a gap in lightweight, modular, and extensible frameworks suitable for research and training in adversary simulation.

## Objectives

- Develop a modular and extensible C2 framework using Python and C.

- Implement encrypted communication (HTTPS or DNS-based).

- Simulate post-exploitation tasks: reverse shell, keylogging, screenshot capture.

- Integrate evasion features such as sandbox detection and obfuscation.

- Map techniques to the MITRE ATT&CK framework.

- Validate the framework in a virtual lab environment.

## Tools and Technologies

Programming Languages: Python, C  
Libraries: OpenSSL, Requests, Socket  
Platforms: Windows, Linux  
Lab Environment: VirtualBox/VMware  
Frameworks: MITRE ATT&CK, ELK stack (for detection testing)  
Others: GitHub, Wireshark, PyInstaller

## Expected Outcomes

- A fully functional modular C2 framework for red team simulations.  
- A tested and documented set of payloads and post-exploitation modules.  
- Academic report and potential publication-ready material.

# Payload Design and Specifications

To ensure the versatility and effectiveness of the proposed Modular C2 Framework, we define a comprehensive set of payloads that will be developed and tested across both Windows and Linux environments. These payloads will cover core access capabilities, evasion, post-exploitation, and stealth-based attack vectors.  
  
1. Core Payloads (Cross-Platform)  
- Reverse Shell:  
 - Windows: Winsock-based shell or PowerShell  
 - Linux: Bash-based TCP reverse shell  
- File Upload/Download Module: Base64 encoded transfers  
- Command Execution Module: Executes OS-specific commands  
  
2. Post-Exploitation Payloads  
- Screenshot Capture (Windows: GDI+/PyAutoGUI, Linux: scrot/xwd)  
- Keylogger (Windows: SetWindowsHookEx, Linux: /dev/input/\*)  
- Credential Dumping  
  
3. Evasion Techniques  
- Sleep and Jitter  
- Environment Awareness  
- String Obfuscation  
- LOLBins (mshta, certutil, etc.)  
  
4. Advanced Payloads  
- Process Injection (Windows: CreateRemoteThread, Linux: ptrace)  
- Persistence Mechanisms (Registry, crontabs, etc.)  
- Privilege Escalation Checkers  
  
5. Optional Covert Payloads  
- DNS Tunneling Agent  
- Webcam Capture  
  
6. Payload Formats  
- EXE, DLL, ELF, Python Script, Shellcode

# C2 Framework Implementation Guide

C2 Framework Architecture Overview  
- Attacker (C2 Server) <-> Encrypted Channel <-> Victim Agent  
  
Components:  
1. C2 Server (Python)  
2. Agent Payloads (Python/C)  
3. Encryption Module (AES/TLS)  
4. Post-Exploitation Modules  
5. Evasion Features  
  
Step-by-Step Implementation:  
  
Phase 1: C2 Server (Python)  
- Socket or Flask API server to handle agents  
  
Phase 2: Agent / Payload  
- Python-based client that connects to the server and executes commands  
  
Phase 3: Encryption Layer  
- AES encryption of tasking and output  
  
Phase 4: Post-Exploitation Modules  
- Screenshot, keylogger, file transfer  
  
Phase 5: Evasion Techniques  
- Anti-VM checks, LOLBins, jitter, obfuscation  
  
Phase 6: Testing Environment  
- Virtual machines with ELK, Wireshark, AV  
  
Folder Structure:  
c2\_framework/  
├── server/  
├── agent/  
├── crypto/  
├── modules/  
├── docs/

# Advanced Red Team Tactics

## 1. Multi-Stage Payload Delivery

\*\*Description\*\*: Use a lightweight stager to download and execute the full agent. This simulates real-world APTs like FIN7 or APT29 and reduces initial detection risk.

\*\*How to Implement\*\*: Initial script fetches the encrypted second-stage payload and decrypts it in memory using AES.

## 2. Beaconing Behavior Mimicry

\*\*Description\*\*: Introduce randomized sleep intervals and stealthy beacon patterns. Optionally trigger beaconing through out-of-band methods like checking for a specific URL.

\*\*How to Implement\*\*: Implement `time.sleep(random.randint(10, 60))` and use special HTTP triggers for callbacks.

## 3. Malleable Communication Profiles

\*\*Description\*\*: Obfuscate traffic by modifying HTTP headers, user-agent strings, and embedding data in image formats or custom content types.

\*\*How to Implement\*\*: Use POST requests with `Content-Type: image/gif` and encode tasking inside a PNG payload or query string.

## 4. Modular Evasion Plugins

\*\*Description\*\*: Introduce evasion modules that can be plugged into the agent dynamically to handle VM detection, string obfuscation, and PowerShell downgrades.

\*\*How to Implement\*\*: Design an `evasion.py` module to perform anti-VM checks and AMSI bypass prior to execution.

## 5. Dynamic Code Loading

\*\*Description\*\*: Allow agents to fetch and execute scripts dynamically, increasing modularity and reducing on-disk footprint.

\*\*How to Implement\*\*: Use Python `exec()` or Bash `eval` to run payloads pulled from the server.

## 6. Advanced C2 Protocols

\*\*Description\*\*: Implement support for alternate channels such as DNS tunneling, ICMP, or HTTP/2 to increase stealth.

\*\*How to Implement\*\*: Send base64 tasking in DNS TXT records and interpret responses inside agent logic.

## 7. In-Memory Execution (Fileless)

\*\*Description\*\*: Run shellcode or scripts entirely in memory to avoid AV detection.

\*\*How to Implement\*\*: Use `ctypes` with `VirtualAlloc` and `CreateThread` for in-memory payload execution on Windows.

# Project Execution Manual

## Step 1: Prepare the Environment

On the attacker machine (Kali Linux / Ubuntu):  
- Ensure Python 3.x is installed:  
 sudo apt update  
 sudo apt install python3 python3-pip -y  
  
- Install required Python libraries:  
 pip3 install flask pycryptodome requests pynput pyautogui  
  
- (Optional) Install PyInstaller to generate Windows EXEs:  
 pip3 install pyinstaller  
  
- On the victim machines, use Windows 10 and Ubuntu Desktop/Server.

## Step 2: Setup the C2 Server

Option A: Socket-Based Server  
- Navigate to the 'server' directory and run:  
 python3 c2\_server.py  
  
Option B: Flask REST API Server  
- Use the Flask-based HTTP server:  
 export FLASK\_APP=flask\_server.py  
 flask run --host=0.0.0.0 --port=8000  
  
- Ensure port 8000 is open on firewall and router.

## Step 3: Configure and Deploy the Agent

Modify the agent script with your server's IP and port:  
SERVER\_IP = "your.server.ip"  
SERVER\_PORT = 4444  
  
Run on Linux victim:  
 python3 agent.py  
  
Compile EXE for Windows:  
 pyinstaller --onefile --noconsole agent.py  
  
Run the generated agent.exe on the Windows VM.

## Step 4: Execute Commands and Post-Exploitation Tasks

From C2 server:  
- Issue basic commands:  
 whoami, ipconfig, netstat  
  
- Trigger modules:  
 screenshot, keylogger\_start, file upload/download  
  
- Agent returns the output or files to the C2 server.

## Step 5: Implement Encrypted Communication

Use AES encryption between agent and server:  
- Import and use functions from aes\_crypto.py  
  
- Example usage:  
 enc = encrypt('whoami', key)  
 dec = decrypt(enc, key)  
  
- Share static key manually or use key exchange logic.

## Step 6: Evasion and Stealth Testing

On Windows VM:  
- Turn on Windows Defender  
- Run agent.exe and observe alerts  
- Techniques to improve evasion:  
 - Add sleep intervals: time.sleep(random.randint(10, 60))  
 - Check for VM artifacts (e.g., VBox, MAC address)  
 - Use LOLBins like mshta, regsvr32 for execution

## Step 7: MITRE ATT&CK Mapping

Document all tactics and techniques used:  
- Example:  
 T1059 - Command Line Execution  
 T1027 - Obfuscated Files or Information  
 T1218 - LOLBins Execution  
  
- Use attack.mitre.org for mapping  
- Record what technique each payload simulates and its effectiveness.

## Step 8: Reporting and Demonstration

- Capture screenshots or record video of:  
 - C2 and agent communication  
 - Post-exploitation features (keylogger, screenshot)  
 - Detection bypass or AV logs  
  
- Include logs from both attacker and victim VMs  
- Summarize results, issues, and improvement ideas

# Explanation of Agent, Server, and Modules

## 1. Agent – The Payload That Lives on the Victim

The agent is the program (payload) deployed on the victim machine. It continuously connects back to the C2 server and executes any instructions it receives. It is responsible for carrying out tasks like taking screenshots, logging keystrokes, or running shell commands, and sending the results back to the server.

🔹 Typical OS: Windows or Linux (target system)

🔹 To run: On the victim machine, execute the compiled agent or run the script manually.

## 2. Server – The Brain of the Operation

The C2 server acts as the command center. It communicates with one or more agents, issues commands, and receives results. It maintains session data and can manage multiple targets simultaneously.

🔹 Typical OS: Kali Linux / Ubuntu (attacker machine)

🔹 To run: On the attacker system, execute the Python C2 server (socket or Flask-based).

## 3. Modules – Functional Tools Used by the Agent

Modules are task-specific scripts or functions used by the agent to perform advanced actions like taking screenshots, logging keystrokes, or transferring files. They can be invoked when specific tasks are requested by the C2 server.

🔹 Where used: Executed on the victim machine by the agent

🔹 Examples: screenshot.py, keylogger.py, file\_transfer.py, evasion.py

## 4. How They Work Together

1. The agent connects to the C2 server (beaconing).  
2. The server assigns a task to the agent (e.g., run screenshot).  
3. The agent loads the respective module and executes the task.  
4. The output (e.g., image, text) is sent back to the server securely.  
5. The server logs or displays the result.

This creates a loop where the server can continually task and control all connected agents.

## 5. Summary of Execution Locations

- Run the C2 server (server-side Python script) on the \*\*attacker machine\*\* (Kali/Ubuntu).  
- Run the agent (Python script or compiled executable) on the \*\*victim machine\*\* (Windows/Linux).  
- Modules are executed by the agent, so they must be available to or embedded in the agent code running on the \*\*victim\*\*.

# Expanded Payload Capabilities

# Core Components

* Modern Web Interface with secure session management and real-time agent/task control
* Shellcode Generator with multiple formats and auto-saving
* Advanced Post-Exploitation Modules (Screenshot, Keylogger, Process Injection)
* Integrated Evasion Techniques (Anti-VM, Sandbox, Traffic Obfuscation)

## Shellcode Generator API

Class: ShellcodeGenerator (modules/shellcode.py)  
- generate\_reverse\_shell(host, port)  
- generate\_bind\_shell(port)  
- generate\_exec(command)  
- encode\_shellcode(shellcode, encoding='base64')  
- decode\_shellcode(encoded\_shellcode, encoding='base64')  
- save\_shellcode(shellcode, filename=None)

## Web Interface Endpoints

(server/web\_interface.py)  
- /api/agents - List connected agents  
- /api/tasks - Manage tasks  
- /api/results/<task\_id> - Get task results  
- Real-time execution with secure session threads

# Usage and Installation

# Installation Steps

* Clone repository and install dependencies from requirements.txt
* Start web interface (python server/web\_interface.py)
* Start shellcode interface (python server/shellcode\_interface.py)

# Web Interface Usage

* Access dashboard at http://localhost:5001
* Manage connected agents and create tasks
* Monitor results in real time

# Shellcode Generator Usage

* Navigate to http://localhost:5001
* Select shellcode type and encoding format
* Generate and save shellcode

## Security Features

# Security

* Encrypted communications (AES) with session protection
* API Key-based authentication
* Evasion mechanisms: Process hiding, Traffic Obfuscation, Anti-Analysis

## Project Structure

Updated Project Structure:  
c2-framework/  
├── agent/  
│ └── agent.py  
├── docs/  
│ ├── mitre\_mapping.md  
│ └── technical\_documentation.md  
├── modules/  
│ ├── credential\_dump.py  
│ ├── dns\_tunnel.py  
│ ├── evasion.py  
│ ├── keylogger.py  
│ ├── persistence.py  
│ ├── post\_exploit.py  
│ ├── priv\_esc.py  
│ ├── process\_injection.py  
│ ├── shellcode.py  
│ └── webcam.py  
├── server/  
│ ├── server.py  
│ ├── shellcode\_interface.py  
│ ├── web\_interface.py  
│ └── templates/  
│ └── shellcode.html  
├── README.md  
└── requirements.txt