Speed Up Your Reverse Engineering Investigations

With Ghidra and OpenAI

Author:

Charles Lomboni

Date:

June 2023

Threat Intelligence & Incident Response Team

J O E S

Table of Contents

What is OpenAI?

OpenAI is an artificial intelligence research organization that was founded in 2015. The organization's mission is to create advanced AI systems that benefit humanity. To accomplish this mission, OpenAI has developed a range of cutting-edge AI models and tools used by researchers, developers, and businesses worldwide.

One of the most well-known AI models developed by OpenAI is GPT-3, a language model capable of generating natural language text that is often indistinguishable from text written by humans. GPT-3 has been used for various applications, including language translation, chatbots, content generation, and more.

In addition to GPT-3, OpenAI has developed a range of other AI models used in various applications. These models include computer vision models like DALL-E, which can generate images based on textual input, and reinforcement learning models like OpenAI Five, which can play complex games like Dota 2 at a high level.

OpenAI API

OpenAI API is a cloud-based platform that provides access to powerful artificial intelligence models created by OpenAI. The API allows developers to integrate these models into their own applications.

To use the OpenAI API, developers need to sign up for an API key and then use it to authenticate their requests. The API supports several programming languages, including Python, Java, and Ruby, and provides easy-to-use client libraries that simplify the integration process. Once authenticated, developers can send requests to the API to perform various AI tasks and receive the results in a structured format, such as JSON.

The OpenAI API is designed to be flexible and scalable, allowing developers to choose the level of complexity they need for their projects. Depending on their specific use case and budget, they can select from various models and sizes. The API also provides various pricing plans, including a free tier for developers who want to experiment and test their applications before deploying them to production.

What is Ghidra?

Ghidra is a powerful software reverse engineering (SRE) suite developed by the National Security Agency (NSA) and released as open-source software in 2019. The tool allows security researchers and developers to analyze and understand software code by reverse engineering binaries, disassembling executables, and decompiling applications.

The NSA's Research Directorate initially developed Ghidra in the early 2000s as a proprietary tool for the agency's internal use in conducting cyber operations and analyzing malware. However, in 2011, the agency decided to release the tool to the public through a technology transfer program in order to make it more widely available for academic and research purposes.

Ghidra's development was guided by the NSA's experience in dealing with complex software systems and conducting cyber operations, making it a highly capable and versatile SRE tool. It offers many features, including support for multiple processor architectures, a powerful disassembler and decompiler, and a user-friendly graphical interface.

The tool has become increasingly popular among security researchers, software developers, and reverse engineers, due to its robust and reliable functionality, its open-source licensing, and the strong community support. Ghidra can be used to reverse engineer a wide variety of software and systems, including malware, firmware, and proprietary applications. It can also help to identify and fix vulnerabilities in software systems, making it a valuable tool for improving the security of digital systems.

In summary, Ghidra is a powerful software reverse engineering suite developed by the NSA, released as open-source software in 2019. It is widely used by security researchers, software developers, and reverse engineers to analyze and understand software code, including malware and proprietary applications. Its features include support for multiple processor architectures, a powerful disassembler and decompiler, and a user-friendly graphical interface, making it a highly capable and versatile tool.

Ghidra Plugins?

Definition

Ghidra Plugins are extensions that can be added to the Ghidra software to enhance its functionality and provide additional features. Plugins can be used to add support for new file formats, improve the disassembly and decompilation capabilities of Ghidra, automate everyday tasks, and more. The community develops some plugins, while the NSA or other organizations develop others.

Setting up the environment

Ghidra

To install an official pre-built multi-platform Ghidra release:

- Install JDK 17 64-bit
  - Export the JDK path to JAVA_HOME
- Download a Ghidra release file
- Extract the Ghidra release file
- Launch Ghidra: **./ghidraRun** (or **ghidraRun.bat** for Windows)

Python 3.11

To install Python 3.11

```
sudo add-apt-repository ppa:deadsnakes/ppa
sudo apt install python3.11 python3.11-distutils
curl -sS https://bootstrap.pypa.io/get-pip.py | python3.11
```

PyCharm

PyCharm is an Integrated Development Environment (IDE) for Python programming. PyCharm provides a comprehensive set of tools for Python developers, including code editing, debugging, testing, version control integration, and more. It has a user-friendly interface and a wide range of features that make it popular among developers.

To install PyCharm on Ubuntu, you must install the snap package beforehand.

```
sudo apt install snapd
```

After that, we can install PyCharm.

```
sudo snap install pycharm-community –classic
```

Pyhidra

Pyhidra is a Python library that provides direct access to the Ghidra API within a native CPython interpreter using jpype. As well, Pyhidra contains some conveniences for setting up analysis on a given sample and running a Ghidra script locally. It also contains a Ghidra plugin to allow the use of CPython from the Ghidra user interface.

To install it, we can follow the GitHub instructions.

1. Download and install Ghidra to a desired location.
2. Set the **GHIDRA_INSTALL_DIR** environment variable to point to the directory where Ghidra is installed.
3. Install **pyhidra**.

```
pip install pyhidra
```

Ghidra Pyi Generator

The Ghidra .pyi Generator generates .pyi type stubs for the entire Ghidra API. Those stub files can later be used in PyCharm to enhance the development experience.

The release contains PEP 561 stub package, which can simply be installed with `pip install ghidra-stubs*.whl` into the environment in which the real Ghidra module is available. Any conformant tool will then use the stub package for type analysis purposes.

Get the last version from

- `https://github.com/VDOO-Connected-Trust/ghidra-pyi-generator/releases`

Development Methodology

In our workshop, we'll adopt a development process that integrates REMnux, PyCharm, Ghidra, and Ghidra's Pyi Stubs. We chose REMnux as our operating system because it's equipped with a variety of tools that aid in analyzing and reverse engineering software.

Our primary workspace is PyCharm. This is where we'll write our scripts. PyCharm is also where the Pyi Stubs come into play. These stubs will give us better guidance as we code, offering valuable suggestions and feedback in real time. They serve as a bridge between our IDE and Ghidra, making it easier to develop scripts that utilize Ghidra's API.

Once we've developed a script in PyCharm, we will immediately deploy it in Ghidra for testing. This fast development-test cycle is one of the key benefits of our workflow. It reduces the time spent switching between coding and testing, allowing us to quickly see the results of our work and make necessary adjustments.

With these tools and this process, we have a powerful and efficient way to develop, test, and refine scripts for Ghidra, improving our overall productivity in reverse engineering tasks.

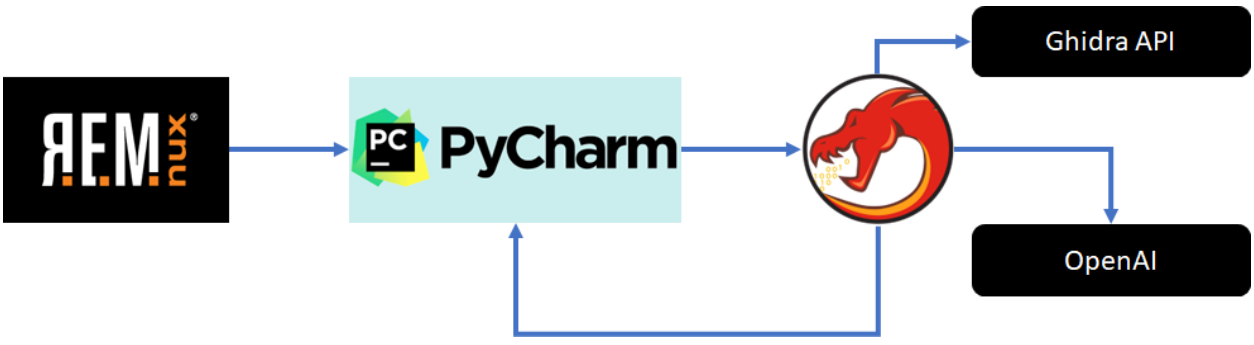

Figure 1 - The Development Methodology process

Using the API Guide

To open the Ghidra API documentation within the Ghidra interface:

1. Open Ghidra: Launch Ghidra from your system's applications menu or by running the ghidraRun script from the command line.
2. Open the Ghidra API documentation: Once Ghidra is open, click on the "Help" menu and select "Ghidra API Help". This will open the Ghidra API documentation within Ghidra.
3. Get information in Ghidra Github

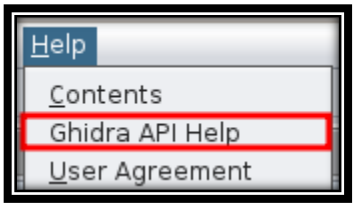

Figure 2 - Ghidra, open API Help

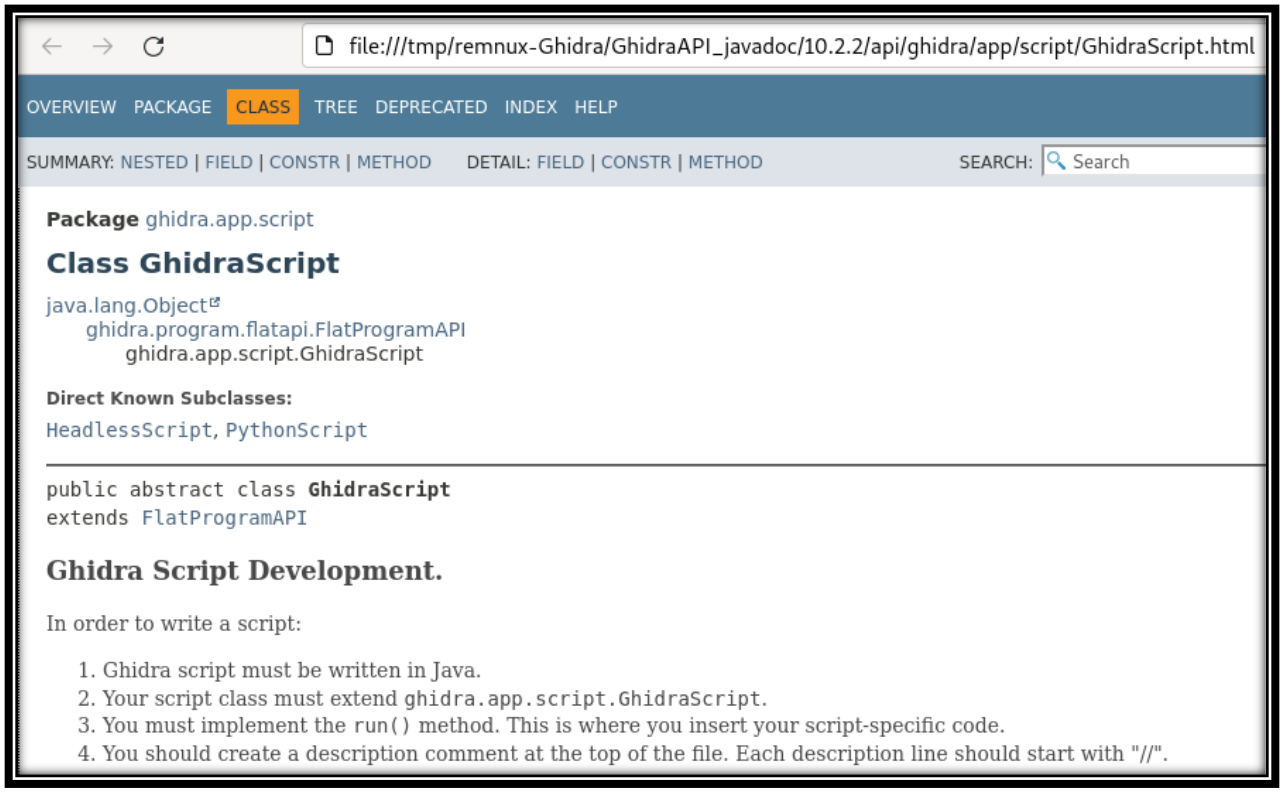

Figure 3 - Ghidra API Help first page

Alternatively, you can also access the Ghidra API documentation on the official website at https://ghidra.re/ghidra_docs/api/index-all.html. This website provides an online version of

the Ghidra API documentation that you can browse and search to find the classes and methods you need for your scripts and plugins.

By using the Ghidra API documentation, you can find detailed information on each class and method, including its purpose, syntax, and examples of how to use it. This will help you to develop more effective and efficient scripts and plugins for Ghidra.

Ghidra Scripting API

The Ghidra scripting API provides developers with a set of functions to automate tasks and extend the capabilities of the software. The API is divided into two parts: the Flat API `ghidra.app.decompiler.flatapi` and the remainder of the functions (http://ghidra.re/ghidra_docs/api/overview-tree.html).

The **Flat API** provides a simplified interface for interacting with the decompiler, allowing developers to easily access high-level information about the code. The other functions in the API provide more complex functionality, such as access to the program database and the ability to modify instructions and symbols.

If you're new to Ghidra scripting, this reference can be a useful tool for identifying the function you require and locating its prototype in the documentation.

Definition	Method
Memory Addresses	`addEntryPoint`, `addInstructionXref`, `createAddressSet`, `getAddressFactory`, `removeEntryPoint`
Code Analysis	`analyze`, `analyzeAll`, `analyzeChanges`, `analyzeAll`, `analyzeChanges`
Code Listing	`clearListing`
Data Declaration	`createAsciiString`, `createAsciiString`, `createBookmark`, `createByte`, `createChar`, `createData`, `createDouble`, `createDWord`, `createDwords`, `createEquate`, `createUnicodeString`, `removeData`, `removeDataAt`, `removeEquate`, `removeEquate`, `removeEquates`
Get Data from Memory	`getInt`, `getByte`, `getBytes`, `getShort`, `getLong`, `getFloat`, `getDouble`, `getDataAfter`, `getDataAt`, `getDataBefore`, `getLastData`, `getDataContaining`, `getUndefinedDataAfter`, `getUndefinedDataAt`, `getUndefinedDataBefore`, `getMemoryBlock`, `getMemoryBlocks`, `getFirstData`
References	`createExternalReference`, `createStackReference`, `getReference`, `getReferencesFrom`, `getReferencesTo`, `setReferencePrimary`
Data Types	`createFloat`, `createQWord`, `createWord`, `getDataTypes`, `openDataTypeArchive`
Set Memory Address	`setByte`, `setBytes`, `setDouble`, `setFloat`, `setInt`, `setLong`, `setShort`

Work with Fragments	`getFragment`, `createFragment`, `createFunction`, `createLabel`, `createMemoryBlock`, `createMemoryReference`, `createSymbol`, `getSymbol`, `getSymbols`, `getSymbolAfter`, `getSymbolAt`, `getSymbolBefore`, `getSymbols`, `getBookmarks`
Disassemble Bytes	`disassemble`
Transactions	`start`, `end`
Find Values	`find`, `findBytes`, `findPascalStrings`, `findStrings`
Function Level	`getGlobalFunctions`, `getFirstFunction`, `getFunction`, `getFunctionAfter`, `getFunctionAt`, `getFunctionBefore`, `getFunctionContaining`, `getLastFunction`
Program Level	`getCurrentProgram`, `saveProgram`, `set`, `getProgramFile`
Instruction Level	`getFirstInstruction`, `getInstructionAfter`, `getInstructionAt`, `getInstructionBefore`, `getInstructionContaining`, `getLastInstruction`
Equates	`getEquate`, `getEquates`
Remove Things	`removeBookmark`, `removeFunction`, `removeFunctionAt`, `removeInstruction`, `removeInstructionAt`, `removeMemoryBlock`, `removeReference`, `removeSymbol`
Comments	`setEOLComment`, `setPlateComment`, `setPostComment`, `setPreComment`, `getPlateComment`, `getPostComment`, `getPreComment`, `getEOLComment`, `toAddr`
Decompile Bytes	`FlatDecompilerAPI`, `decompile`, `getDecompiler`
Miscellaneous Functions	`getMonitor`, `getNamespace`, `getProjectRootFolder`

Building your first script

Create a Python Script

Open Ghidra and load the binary or project you want to work with. In the "Script Manager" window, which can be accessed from the "Window" menu.

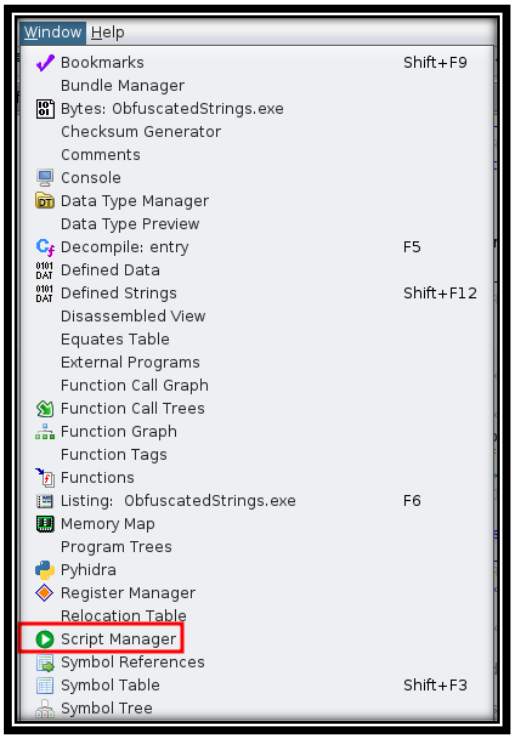

Figure 4 - Script Manager in Ghidra

Click on the "New Script" button.

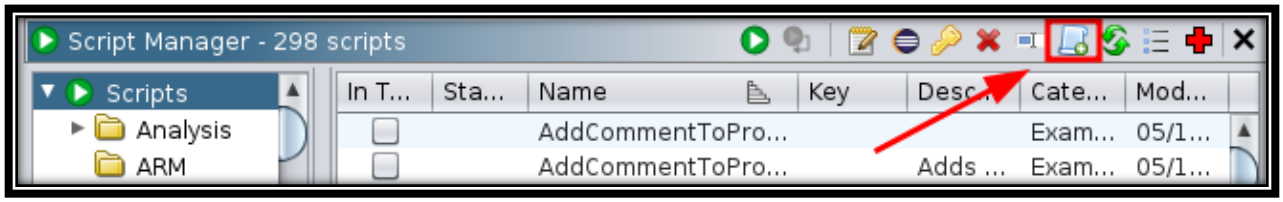

Figure 5 - Create new script file

Select Python

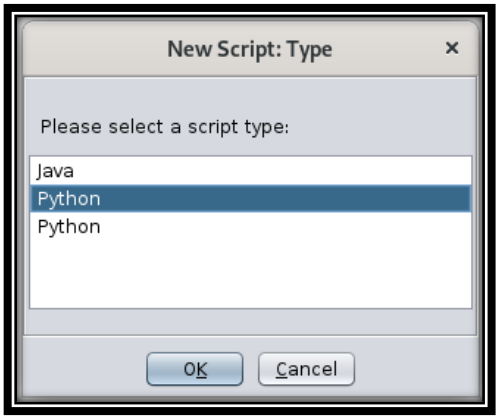

Figure 6 - New Script type

Provide a name for your script and click "OK."

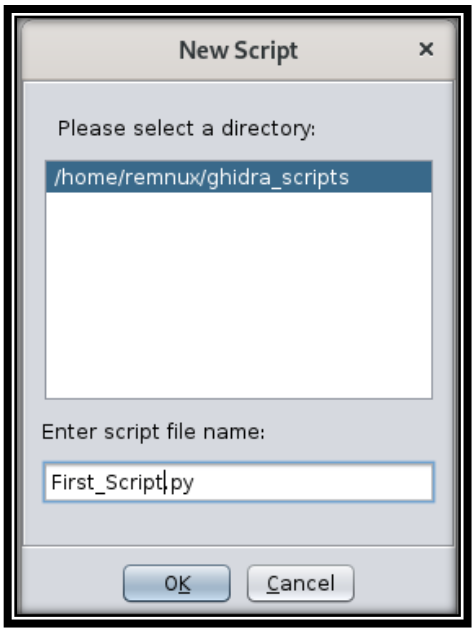

Figure 7 - Script name

Python Code

```
# Search for all XOR with some hardcode Hex value in the program
# @author Charles Lomboni (charlesl@securityjoes.com)
# @category SecurityJoes
# @keybinding CTRL SHIFT ALT 1
# @menupath
# @toolbar

import ghidra

try:
    from ghidra.ghidra_builtins import *
except:
    pass


def run():
    # Get all instruction from the binary
    instructions = currentProgram.getListing().getInstructions(True)

    for ins in instructions:
        mnemonic = ins.getMnemonicString()
        if mnemonic == "XOR":
            operand1 = ins.getOpObjects(0)
            operand2 = ins.getOpObjects(1)
            if '0x' in str(operand1) or '0x' in str(operand2):
                print(f"[+] Address: {ins.address} - {ins}")


run()
```

Let's break down each part of this code.

1. **import ghidra**:
    - This line imports the **ghidra** module, which allows us to use the Ghidra API.
2. **try-except** block for importing **ghidra_builtins** module:
    - This block tries to import the **ghidra_builtins** module, which provides additional features for Ghidra.
    - As this builtins are only signatures, we put it just to pycharm recognize the Pyi Stubs.
3. Definition of the **run()** function:
    - This function contains the main logic of our script.
4. Iterating over all the instructions:
    - We go through each instruction one by one.
5. Checking for **XOR** instructions with immediate operands:
    - We check if the instruction we are currently examining is an XOR instruction.
    - We want to find XOR instructions that work with immediate values.
6. Retrieving operands and checking for immediate values:
    - If we find an XOR instruction, we retrieve its operands.
    - We check if either operand contains a hexadecimal value, which is often represented with a **0x** prefix.
7. Printing the address and instruction details:
    - If we find an XOR instruction with an immediate value, we print its address and details.
    - The address tells us where the instruction is located in the binary.
    - The details give us more information about the instruction itself.

The code analyzes the binary, looking for **XOR** instructions that work with immediate values, and prints information about them. The purpose is to help identify specific instructions within the binary that may be of interest for further analysis or investigation.

Building your second script

The steps you should take are the same as we have taken to the first script. The code to the second script is here.

Python Code

```
# Decode strings used in custom sample from CONFidence workshop
# @author Charles Lomboni (charlesl@securityjoes.com)
# @category SecurityJoes
# @keybinding CTRL SHIFT ALT 2
# @menupath
# @toolbar

import ghidra

# For type checking
try:
    from ghidra.ghidra_builtins import *
except:
    pass
```

```
# translate string
from ghidra.program.model.data import StringDataInstance
from ghidra.program.model.data import TranslationSettingsDefinition
from ghidra.program.util import DefinedDataIterator


# *************************** TRANSLATE STRING ****************************

def customize_str(s):
    decoded_str = decode_str(s)
    print(f"[+] {decoded_str}")
    return f"[+] {decoded_str}"


def translate_string():
    if currentProgram is None:
        return

    monitor.initialize(currentProgram.getListing().getNumDefinedData())
    monitor.setMessage("[+] Deobfuscation strings...")

    data_iterator = DefinedDataIterator.definedStrings(currentProgram, currentSelection)
    for data in data_iterator:

        if monitor.isCancelled():
            break

        str_instance = StringDataInstance.getStringDataInstance(data)
        s = str_instance.getStringValue()
        if s:
            TranslationSettingsDefinition.TRANSLATION.setTranslatedValue(data,
customize_str(s))
            TranslationSettingsDefinition.TRANSLATION.setShowTranslated(data, True)
            monitor.incrementProgress(1)


# *************************** DECODE STRING ****************************
def decode_str(encoded_str):
    xored_str = []

    for x in encoded_str:
        xored_str.append(chr(ord(x) ^ 2))

    return ''.join(xored_str)


# *************************** MAIN ****************************
def run():
    if monitor.isCancelled():
        print(f"Operation canceled.")
        return

    if currentSelection is None:
        print(f"You should select the encrypted strings")
        return

    translate_string()


run()
```

Essentially, we took the Ghidra TranslationString.java file and converted it to Python. In the customized section, we replaced the original return code, which was `"TODO" + s + "TODO"`, with our own implementation.

Let's break down each part of this code.

This code snippet is used to deobfuscate and translate strings in the Ghidra.

1. The code begins by importing the necessary modules from the Ghidra library.
2. Next, there is a function called **customize_str(s)** which takes an encoded string **s** as input. It decodes the string by performing a bitwise **XOR** operation on each character and returns the decoded string. The decoded string is then printed and returned.
3. Another function called **translate_string()** is defined. This function is responsible for **deobfuscating** and translating strings in the current program. It checks if there is a valid program available and initializes a progress monitor. It then iterates over the defined string data instances in the program, retrieving the string value of each instance. If a string value exists, it applies the **customize_str()** function to the string, sets the translated value of the data instance, and shows the translated string. The progress monitor is updated after each iteration.
4. The **decode_str(encoded_str)** function is used internally by **customize_str(s)**. It performs a bitwise **XOR** operation with the value **2** on each character of the encoded string and returns the decoded string.
5. Finally, the **run()** function is defined. It checks if the operation has been canceled or if a valid selection has been made. If everything is in order, it calls the **translate_string()** function.

By running this code within the Ghidra environment, it deobfuscates and translates selected strings, providing a more readable and understandable representation of the obfuscated data.

Integrating OpenAI in Ghidra

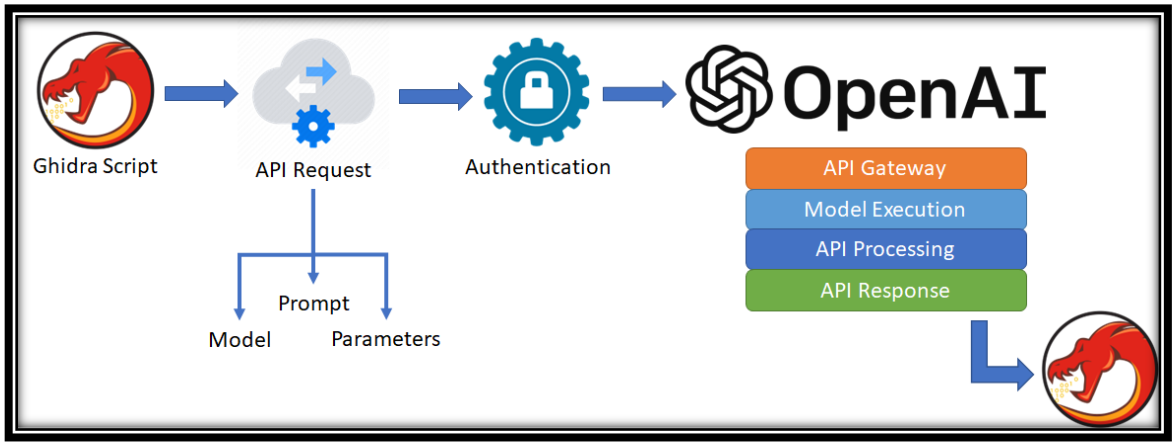

Figure 8 - Ghidra and OpenAI integration flow

Integrating OpenAI with Ghidra allows you to increase the power of natural language processing and machine learning to enhance your reverse engineering workflows. By integrating OpenAI's language model into Ghidra, you can automate certain tasks, generate code snippets, assist with code analysis, and even provide contextual information for better understanding of the codebase.

Here's a step-by-step guide on how to integrate OpenAI in Ghidra:

1. **Set up OpenAI API access:** To begin, you'll need to sign up for OpenAI and obtain an API key.
2. **Import the OpenAI library in Ghidra:** In your Ghidra script, import the OpenAI library by adding the following line of code at the beginning: **import openai** or just use requests.
3. **Authenticate with the OpenAI API:** Before making any requests to the OpenAI API, you'll need to authenticate using your API key.
4. **Utilize OpenAI's language model:** Once authenticated, you can start utilizing OpenAI's language model within your Ghidra scripts.
5. **Incorporate OpenAI into your specific use cases:** Consider your specific reverse engineering use cases and how OpenAI can assist. For example, using AskJOE to speed up your reverse engineering investigations.
6. **Test and iterate:** As you integrate OpenAI into your Ghidra workflows, it's important to test and iterate on your implementation. Fine-tune the interactions with the language model, experiment with different prompts or queries, and refine the generated outputs to ensure they align with your desired objectives.

By integrating OpenAI into Ghidra, you can unlock new possibilities for automating tasks, improving code analysis, and enhancing your reverse engineering capabilities using the power of natural language processing and machine learning.

AskJOE

Description

AskJoe is an innovative tool that uses the capabilities of OpenAI to enhance the utility of Ghidra for researchers conducting malware analysis. This tool was originally inspired by the concept of Gepetto, embodying a similar innovative spirit. By integrating OpenAI's sophisticated functionality, AskJoe significantly streamlines the complex process of reverse engineering. This enhanced efficiency empowers researchers, providing them with more accurate and comprehensive threat detection capabilities, thereby facilitating more effective threat mitigation strategies.

Features

AskJOE provides several prevailing features to enhance your coding experience and assist with code analysis and refactoring. Here are some of the key functionalities offered:

1. **askChoices**: With the **askChoices** feature, AskJOE allows you to interactively ask questions and receive multiple-choice answers.
2. **Explain selection**: AskJOE's **Explain selection** feature provides detailed explanations and insights for a selected portion of code. It helps you understand the functionality and purpose of specific code snippets within your project.
3. **Config file**: AskJOE now supports a config file where you can customize various settings and preferences to tailor the tool to your specific needs. This allows for a more personalized and efficient coding experience.
4. **Execute all**: The **Execute all** functionality in AskJOE enables you to execute and test multiple code snippets or functions simultaneously, saving time and effort during the development process.
5. **Stack String**: With the **Stack String** feature, AskJOE assists in managing and manipulating string data within your code. It provides utilities for concatenation, formatting, and other string-related operations.
6. **Rename function**: AskJOE's **Rename function** feature simplifies the process of renaming functions in your codebase. It ensures that all references to the renamed function are updated automatically, saving you from tedious manual updates.
7. **Changed color from function renamed**: AskJOE now supports the ability to change the color of a function after it has been renamed. This visual indicator helps in quickly identifying and tracking the renamed functions throughout your code.
8. **Explain function**: With the **Explain function** feature, AskJOE provides explanations and insights into the purpose, behavior, and usage of specific functions within your codebase. This helps in understanding complex code logic and improving documentation.
9. **Simplify code**: AskJOE's **Simplify code** functionality analyzes your code and offers suggestions to simplify complex expressions or code blocks. It helps in reducing redundancy and improving code readability.

10. **Set OpenAI answer to comment**: AskJOE allows you to automatically set the answer generated by OpenAI as a comment in your code. This feature helps in documenting the reasoning or explanation provided by AskJOE directly in your codebase.
11. **Monitor messages**: AskJOE provides informative messages and progress updates during code analysis and refactoring operations with the Monitor messages feature. It keeps you informed about the status and progress of AskJOE's tasks.

These features make **AskJOE** a valuable tool for your reverse engineering investigations.

Using AskJOE to speed up your RE process

How to contribute?

Introduction

Contributing to open-source projects on GitHub can be a rewarding experience that allows you to collaborate with a community, improve your skills, and positively impact a project. This guide will outline the correct way to contribute to the **AskJOE** project on GitHub, ensuring a flat and operative collaboration process.
Whether you're a beginner or an experienced developer, these steps will help you navigate the contribution process successfully.

Understand the AskJOE Project

Start by thoroughly understanding the **AskJOE** project by visiting its GitHub repository at https://github.com/securityjoes/AskJOE. Read the project's README file, contribution guidelines, and code of conduct. Gain a clear understanding of the project's goals, scope, and existing contributions.

Choose an Area to Contribute

Identify an area of the **AskJOE** project where you can make a meaningful contribution. It can be fixing a bug, implementing a new feature, improving documentation, or enhancing existing code. Consider your skills, interests, and the project's needs when selecting an area to work on.

Fork the AskJOE Repository

Fork the **AskJOE** project's repository by visiting https://github.com/securityjoes/AskJOE and clicking the "Fork" button on GitHub. This creates a copy of the project under your GitHub account, allowing you to freely make changes without affecting the original project.

Create a New Branch

Before making any changes, create a new branch in your local repository. Use a descriptive name that reflects the nature of your contribution. For example, **git checkout -b feature/your-feature-name**.

Do your Magic!

Make the necessary changes to the **AskJOE** codebase or documentation in your local branch. Follow the project's coding style guidelines and maintain consistency with the existing codebase. Commit your changes with clear and descriptive commit messages.

Test your Changes

Thoroughly test your changes to ensure they work as intended. Run tests, perform manual testing, and consider edge cases. If applicable, update or create new tests to validate your changes.

Push changes and create a Pull Request

Push your branch with the changes to your forked repository on GitHub using **git push**. Then, navigate to the original **AskJOE** project's repository at https://github.com/securityjoes/AskJOE and create a new pull request (PR) from your branch. Provide a descriptive title and comprehensive explanation of your changes in the PR description. Include any relevant context, such as references to issues or feature requests.

Enjoy your Contribution!

Once your contribution is reviewed and accepted, celebrate your achievement! Your code or documentation will become part of the **AskJOE** project, and you can proudly share your accomplishment with others.

Conclusion

Contributing to the **AskJOE** project on GitHub can be an enriching experience that helps you grow as a developer and contribute to a larger community. By following these steps, you can navigate the contribution process successfully, collaborate effectively, and make a positive impact on the **AskJOE** project. Happy contributing!