Pop Quiz! Can a Large Language Model Help With Reverse Engineering?

The paper focuses on using the increasingly popular Large Language Models in the context of reverse engineering. It provides a framework for analyzing LLMs in identifying the purpose of an artifact, and extracting information out of it. This framework was employed in the experiments conducted by the authors in the real-world domain of malware and industrial control systems.

In the context of malware, reverse engineering would involve information regarding the specific mechanisms for attack and evasion. These information would help a defender craft strategies to prevent such attacks and provide additional information such as the place of origin of the malware sample, and other such statistical information. In the domain of industrial control systems, reverse engineering is employed in more innocent contexts such as maintaining and refactoring legacy programs by extracting the purpose of the code, the mathematical equations, the parameter values, etc.

The paper uses a quizzing framework, where a model that has taken the decompiled version of a program as input is queried with questions regarding the program. The questions prompted by the authors were about the code's functionality and the variables it uses. For the purposes of the experiments in this paper, the authors have used OpenAI's code-davinci-001 which incidentally is the LLM on which ChatGPT is built. The prompt given to LLM always starts with a language hint which indicates what language the program is using. After this, the program is provided followed by the questions that the models need to answer.

An LLM would have different parameters that can be modified during the experiments. One of the parameter that influences the responses given by the model is its temperature. Temperature is a value that ranges from 0 to 1 with 0 focusing only on the most probable answer(can be replicated across queries) whereas 1 involves a randomness to it that might output the lesser probable words. Another such parameter is the top_p value that limits the number of most probable choices that the model would select.

The LLM model was able to extract the information efficiently, and even managed to succeed in doing so after randomizing the code to an extent (variable name, internal function name, etc were randomized). As the code started becoming more obscure the model started giving out the wrong answers, with the answers related to the functioning of the program being almost entirely incorrect when the stripped version of the code was used. However, it did still manage to keep track of the variable contents as well as responded with the right answer when prompted with true or false questions about the program's functioning. It was also shown that the model performs with similar efficiency in a basic industrial controller algorithm.

For a systematic evaluation process, the authors considered three classes of programs - cybersecurity relevant code, ICS algorithms and real malware source code obtained from sources like vx-underground. The queries to be asked was generated based on a list of capabilities, in the format "the code above does <capability>", "the code above does not <capability>". Along with these questions, some open ended questions that asked the value based on certain assumptions were also queried.

Based on the experiments, it was observed that code-davinci-001 functions better when the questions asked are in the same domain where the program exists. Moreover, it was found that LLM is able to identify the answer to certain questions better than others. However, it was also observed that LLM sometimes answers the same way for both the positive and negative version of a question. The key point of this paper is that the model works in zero-shot setting, that is the model has no prior exposure to the domain of malware programs, or ICS programs but is trained on a large corpus of general text. In such a setting code-davinci-001 was able to answer 72,754 questions out of 136,260 questions (53.39% correct). This is a pretty good result when viewed in the right context of the zero-shot setting.

Discussion

- The first point of discussion was on ChatGPT and on how it is built on top of davinci LLM. It was also discussed on how ChatGPT provides its services for free, which leads to the basic inference that the user (data that user provides) is the product. It's the same strategy that Google uses with its Captcha to obtain data for image classification. The discussion also turned into whether ChatGPT can pass the turing test? It was stated that passing turing test has been achieved long back by machines that can imitate or simulate human behaviour well, and no longer serves as a good metric on evaluating the new models.
- The major point of the paper is the zero-shot setting and how a general model like that could function well in a specific domain such as reverse engineering.
- The model can act in the form of an oracle, that can answer questions for the lower-skilled reverse engineering humans. This would also mean that the gap that prevented regular users from accessing security related tasks would reduce and open up a whole new area of user-base. A regular use with general technical know-how, who doesn't know the "security language" would be able to access the LLM and ask it questions like "What does this file do?" or "What directories does this file change?"
- The major obstacle that LLM model would need to overcome to do tasks like reverse engineering better, would be to have logical processing unit in the pipeline, something that a typical LLM model lacks. A basic example of this fact is that an LLM model would not be able to decode a base64 string because the model does not have a processing unit to do so, but only statistics. For this reason, the biggest progress in GPT4 according to many is the addition of the plugins which leads the potential to add extra functionalities.
- There are many other security tasks that can be performed by an LLM model apart from malware reverse engineering, such as log analysis, finding bugs in the code, finding vulnerabilities, etc. Towards the attacker side, technologies such as these can be used in phishing attacks to imitate a human being, to customize and personalize the phishing attack based on the victim's persona and

profile. It could also be used to replace common phishing words with words that are not so common to fool the spam/phishing detection systems.

- Also, on the defensive side it could be used to perform sentiment analysis on the phishing text as part of the phishing detection process. Most phishing texts operate on a couple of key factors such as urgency and threat of consequence which can be identified through sentiment analysis.
- As mentioned before, LLM can make huge progress in usable security where the regular users would be able to do security tasks such as setting up a firewall using regular human language.
- This might also mean the obsoletion of coding languages, at least in the form that we know them, which could mean a radical shift in the job profile of programmers and computer students.
- Another question was whether GPT can print out an existing malware file. The major issue in this case is that the model would probably not be trained on malware data and hence would lack the understanding to do such a task. A person with a lot of malware files can possibly create a GPT version trained specifically on malware files for malicious purposes.