Cuckoo Watchtower

Domenic Lo Iacono
Department of Computing Security
Rochester Institute of Technology
Rochester USA
dml3483@rit.edu

1 INTRODUCTION

Cuckoo Watchtower is a Large Language Model reporting and analysis tool utilizing GPT-4 from OpenAI. CW (Cuckoo Watchtower) is perfect for taking in the large amount of data generated from Cuckoo and displaying it dynamically in a chat-bot style. The application does this by gathering data from the JSON report generated by Cuckoo and creating smaller but still comprehensive text files on the general, signature, network, and static analysis performed. Once the text files have been created they are passed to the RAG pipeline. Retrieval Augmented Generation is the process of adding additional context to the Large Language Model so that it may answer questions more effectively or gain knowledge to new information. In this case, the new information is the text files that contain the various results of the malware analysis. All of this information is stored as vectors inside of a vector database called Weaviate. Weaviate is an open-source, cloud-native vector database that provides advanced search capabilities through the use of machine learning models.

Following the process of the RAG pipeline, the model is then given an important system message.

The message is as follows: "You are an assistant for malware event analysis. If the user asks questions that are completely unrelated to malware event analysis refuse to answer their question and state that your purpose is to answer questions related to malware and analyze the malware event data. If you don't know the answer, just say that you don't know. Question: question Context: context Answer:"

This system message is extremely important as it sets what role the model will take which in this case is an assistant for malware event analysis. Another important aspect of this system message is the inclusion of the question which is the users query, and the context which is the information passed in through the RAG pipeline that is retrieved as a user asks a question.

Finally, the user is prompted to ask a question. For this tools purposes questions that revolve only around the context given, the context being the text files provided, are suitable to be asked. The system message also instructs the model to not answer unrelated questions which keeps the results on track and more consistent.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 USE CASE

The use case for this tool is well defined while also having room for the freedom of expanding the complexity of the queries. The most intuitive use case is asking for a comprehensive report on the program analyzed. CW preforms this task extremely well providing a response that is easy to read and includes specific information passed in through the RAG pipeline. Furthermore, by adding additional stipulations on report format, being clear on the audience the report is meant for, length requirements, and more a report can be created for almost any situation.

Another intuitive use case for this tool is simply asking the model for more information or asking more clarifying questions. For example, if a signature was detected that the analyst/user was unfamiliar with the user could ask the model for more information.

Finally, another use case that can speed up the efficiency of an analyst's job is the ability for the model to assist in pivoting. Because the model can display and search for certain information much faster than a human can pivoting and conducting an investigation can be significantly sped up.

3 LIMITATIONS

As useful as this tool can be there are certain limitations to be aware of. Firstly, the model only has as much information about the program's analysis that is passed into it through the RAG pipeline. Unfortunately, passing all of the data collected during the process of analysis is impossible. In one example the JSON file that was utilized for the testing of the application had over 700000 lines. OpenAI limits the requests size limit for embedding the information passed through the RAG pipeline and the embedding of information can also be expensive based on which model is utilized. This limitation was dealt with within the application by utilizing multiple functions that take out the most important information.

Another limitation to be aware of relates to if this application were to be used outside of a testing environment. The nature of the sensitive information could require that OpenAI's policies on using chat/training data may require for a different model to be utilized for the application. Utilizing a different model such as Llama, BLOOM, and other open-source models on a private server or system is possible and would still allow for the utilization of the RAG pipeline and similar capabilities.

4 FUTURE WORK

As the accessibility to data is the biggest limitation to the tools usefulness future work for this application would revolve around adding additional data sources from Cuckoo into the RAG pipeline. Providing memory snapshots, adding BSON files that relate to each PID observed in the analysis, and more could be added. However, carefully observing what impact the additional information has on the models responses and quantifying if the responses are better, worse, or accurate is a significant challenge.

Another interesting but much more advanced way to increase the accessibility to data would be giving the ability to run Cuckoo scans to the model through integration with a code execution environment. This would then increase CW's usefulness in pivoting and analysis as it would allow for a user to streamline the process of running additional analyses.

5 CONCLUSION

In conclusion, Cuckoo Watchtower is an application utilizing a LLM to accelerate the malware analysis process and has proved to be effective at improving the efficiency of malware analysis. Although dependent on the data collected during the initial analysis, the ability to create readable, accurate reports that can be customized to their audience is highly useful. The ability to assist in the pivoting and investigation related to the malware sample is also an important role of the application. Lastly, the room to grow and improve the tool as discussed in the future work section is clear.

6 CHAT OUTPUT/TESTING

The following question and response pairs are shown below to showcase and test the applications ability. The JSON file utilized will also be available for recreation of the results on the project's Github page. Note that the application was implemented as a Jupyter notebook using Google Colab. To use the tool, open the notebook in Google Colab and change the path used throughout the cells to the path of the JSON file desired. An OpenAI API key is also required to complete the embedding and requests.

Note: the malware area for this project is not limited to any particular type. For the testing of the application a malware file available from RLES was utilized.

Related directly to the comprehensive report use case the first question and response is displayed below as well as a secondary question that pivots off the report:

6.1 Question 1

Question: Provide a comprehensive report on the program analyzed. Response: Based on the provided documents, the analyzed program exhibits several characteristics and behaviors indicative of potentially malicious activity. Here's a comprehensive report summarizing the findings:

Process Information:

PID: 516 with no child processes identified. Signatures Detected:

(1) Console Output: Command line console output was observed, indicating potential command-line interface manipulation or execution.
(2) Authenticode: The executable is signed, suggesting an attempt to appear legitimate.
(3) Anti-VM Techniques: The program checks the amount of memory available, possibly to detect virtual machines.
(4) Unknown PE Resource Name: Contains an unknown PE resource name, which could indicate packing or obfuscation techniques.
(5) Allocates RWX Memory: Allocates read-write-execute memory, typically for unpacking or executing in-memory payloads.
(6) Anti-Sandbox Techniques: Attempts to delay analysis through sleep functions.
(7) Browser Hijacking: Attempts to modify Internet Explorer's start page.
(8) Suspicious Process Creation: Creates processes that may be considered suspicious.
(9) Executable Dropping: Drops executables to the user's AppData folder, a common tactic for persistence and evasion.
(10) Stealth Techniques: Creates a hidden window, possibly to evade detection by the user or security software.
(11) Anti-VM Network Adapter Checks: Checks network adapter addresses, likely to detect virtual environments.
(12) Use of Windows Utilities: Utilizes Windows utilities for potentially malicious purposes.
(13) No DNS Lookup Communication: Communicates with hosts without performing DNS queries, indicative of direct IP communication to avoid detection.
(14) Info Stealer Keylogger: Implements a keylogger functionality, monitoring keyboard input.
(15) Multiple User Agents: Exhibits network activity with more than one unique user agent, suggesting attempts to mimic different systems or browsers.
(16) Process Injection: Resumes a suspended thread in a remote process, indicative of process injection techniques.
(17) ICMP Traffic: Generates ICMP traffic, which could be used for network reconnaissance or signaling.
(18) Dead Host Communication: Attempts to connect to IP addresses that do not respond, possibly indicating command and control communication attempts with fallback mechanisms.

PE Analysis:

Timestamp: 2017-11-22 00:45:43 Imported DLL Count: 10 PE ImpHash: 460c803d3984135d17296da91c93d9fd Significant DLLs Loaded: KERNEL32.dll, USER32.dll, GDI32.dll, ADVAPI32.dll, SHELL32.dll, ole32.dll, OLEAUT32.dll, SHLWAPI.dll, WININET.dll Notable Imports: Functions related to process and thread management, network communication, file manipulation, and registry access, which are common in malware for execution, persistence, and data exfiltration. Top API Calls:

The program makes numerous calls to APIs such as GetSystemTimeAsFileTime, LdrLoadDll, RegOpenKeyExW, and SetUnhandledExceptionFilter, among others. These calls are consistent with behaviors such as dynamic loading of libraries, registry access for persistence or configuration, and exception handling to maintain stability or evade analysis. Conclusion: The analyzed program demonstrates a range of behaviors that are strongly indicative of malicious intent, including evasion, persistence, stealth, and data exfiltration techniques. The presence of anti-VM and anti-sandbox characteristics, alongside the use of authenticode signing and suspicious network behavior, suggests a sophisticated threat actor aiming to avoid detection and analysis. Further investigation and appropriate defensive measures are recommended.

6.2 Question 2

Question: Can you explain why the dead host communications detected as a signature are important? Response: Dead host communications detected as a signature are significant because they can indicate a malware's attempt to communicate with a command and control (C2) server or other network resource that is no longer active. This behavior can be a red flag for analysts for several reasons:

(1) Attempted C2 Communication**: Malware often communicates with servers controlled by attackers to receive commands, exfiltrate data, or download additional payloads. If a malware sample attempts to contact a host that is no longer responding, it might suggest that the malware's C2 infrastructure has been taken down or has moved, yet the malware itself has not been updated to reflect this change.

(2) Indicator of Compromise (IoC)**: Communications to known dead hosts can serve as an IoC, helping to identify infected machines within a network. This is particularly useful in network forensic analysis, where identifying malicious traffic patterns is crucial for understanding the scope of an infection.

(3) Analysis and Threat Intelligence**: Identifying attempts to communicate with dead hosts can contribute to threat intelligence efforts. By analyzing these attempts, researchers can potentially uncover patterns of behavior associated with specific malware families or campaigns, even leading to the discovery of previously unknown malicious infrastructure.

(4) Malware Evolution**: Observing malware attempting to communicate with dead hosts can provide insights into the lifecycle and evolution of malware campaigns. It can indicate how long certain infrastructure has been decommissioned and how quickly malware authors move to new infrastructure.

In summary, the detection of dead host communications is important because it can provide valuable insights into the behavior and operational aspects of malware, aiding in the detection, analysis, and mitigation of cyber threats.

REFERENCES