LLM-Assisted Malware Analysis for Newbies

ECE 8813

OBJECTIVE:

Your goal in this assignment is to design and implement an automated workflow that uses outputs from static and dynamic analysis tools to guide a Large Language Model (LLM) in identifying and bypassing malware evasion techniques. In short, you will enable an LLM to:

1. Locate where the malware decides whether to execute its malicious behavior.
2. Modify the sandbox environment to trigger that behavior.

This project will test your ability to synthesize context from multiple tools commonly used by malware analysts (e.g., disassembly and sandbox logs) and use that information in an automated fashion.

DUE DATE: Weds, November 5th, 10pm.

Class time on Tuesday, November 4th has been set aside to provide any last-minute guidance / feedback to teams, where possible.

TEAMING: You are required to work in groups of two. No cross-team collaboration is allowed.

Description:

You are interning as a malware reverse engineer at a prestigious security firm. Your manager, a senior analyst, gives you three suspicious samples that seem benign in the sandbox, even though she suspects they are not. Based on her experience, these samples likely include evasion logic that prevents malicious activity unless specific conditions are met. Rather than spending hours on end manually teaching you the tricks of evasion detection, she challenges you to automate a reverse-engineer's typical workflow [1] using an LLM-assisted approach.

Your task is to take the static and dynamic artifacts she provides—outputs from IDA Pro (disassembly) and CAPE Sandbox (API trace)—and create a system that uses an LLM to perform two key tasks:

- *Task 1:* Identify Evasion Checkpoint — Locate the virtual address where the malware checks conditions before executing its payload.
- *Task 2:* Generate Sandbox Bypass — Create a prompt that enables the LLM to produce a Python script to modify the sandbox and reveal malicious behavior.

DELIVERABLE

You must submit code that takes as input both static disassembly and dynamic sandbox traces and produces text (prompt + context) that enables the LLM to perform Tasks 1 and 2. Your LLM prompts will be evaluated based on consistency and correctness.

Specifically, to successfully complete your objectives, you will have to find the most effective prompt and context that allows the LLM to consistently identify the correct answer to the task at hand. For Task 1, you must produce a prompt that recovers the correct virtual address. For Task 2, you must produce a prompt that generates a python script that correctly bypasses the malware.

LEADERBOARD SUBMISSIONS

Because she's busy with other more pressing day-to-day responsibilities, your manager has set up a leaderboard to track your progress. By submitting your prompts to this system, you can receive automated feedback on whether they correctly solve Task 1 or Task 2. You will compete against other interns with similar levels of experience as you. To encourage local testing before submission, a "cool down" period (5 minutes between submissions) is enforced. The scoring rewards teams that achieve correct results with fewer submissions.

DATA:

All the data needed for the assignment can be found on Canvas: https://gatech.instructure.com/courses/463964/files/folder/Assignment%202%20-%20LLM-assisted%20Malware%20Analysis . Please make sure to read this document in its entirety, as well as following the instructions in the README file.

For each of the three real-world malware samples that are hiding malicious behaviors, you are given two files. The first is from the (static) disassembly of the files containing the assembly code (produced by IDA Pro) that the malware is comprised of. The second is (dynamic) API call information, describing the APIs the sandbox observed when the malware executed.

- Static Disassembly

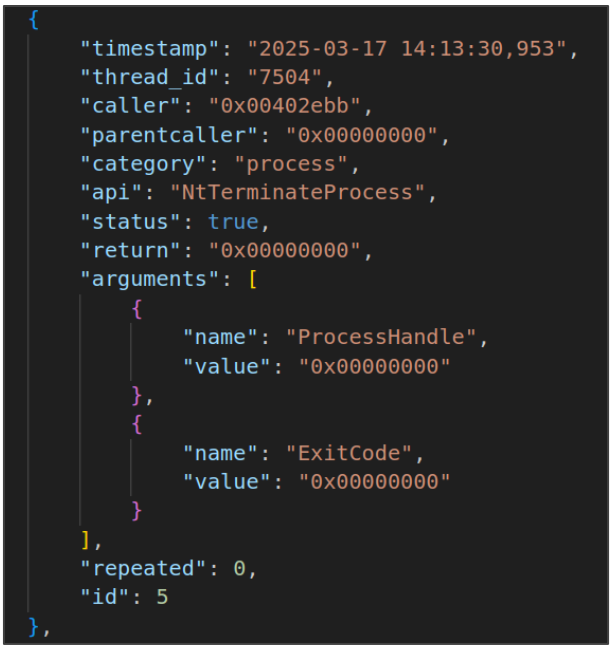
```
00404517: call    DeleteFileA
0040451C: test    eax, eax
0040451E: jnz     short loc_404536
00404520: call    GetLastError
00404525: cmp     eax, 2
00404528: jz      short loc_404536
0040452A: push    64h ; 'd'; dwMilliseconds
```

The **static disassembly** is provided as a text file that lists each function and its corresponding assembly code.

- Every function begins with a line labeled **"Function: <function name>"**, followed by one line per assembly instruction within that function.
- Each instruction line starts with its **virtual address**, which represents the memory location of that instruction. This information is critical for **linking static disassembly data with dynamic sandbox traces**.
- A hint for Task 1: your answer should be the virtual address of an instruction that conditionally "jumps" away to avoid malicious activity. These instructions are prefixed with a "j". In the above image, both "0x40451E" and "0x40452A" are example virtual addresses of conditional jumps.

You are **not expected to fully understand assembly code**. Instead, your goal is to **construct the right context** so that the LLM can interpret the disassembly and identify the relevant information automatically.

- Dynamic API Calls



The **dynamic API call trace** is generated by the sandbox and records the **Windows API functions executed during runtime**. Each entry captures key details about an observed API call, which are essential for understanding the malware's behavior and linking it to the static disassembly.

The most important fields are:

- **`api`** – The name of the Windows API function that was invoked.
- **`caller`** – The virtual address of the instruction immediately following the API call.
- **`arguments`** – The names and values of the parameters passed to the API function.
- **`return`** – The value returned by the API call after execution.

Rubric

Your performance will be graded based on the following criteria:

- Prompt Correctness (80%):
  - **Effectiveness of prompts for solving Tasks 1 and 2 across all samples.** Note that your solution will be run multiple times to check how consistently you generate the correct answer.
- Cost Effectiveness (10%):
  - Because your boss has a limited monthly budget for using AI-assisted tools, points are assigned based on the estimated cost of the input and output tokens that were used in your submission.
- Code Clarity (10%):

- To help her more easily understand how you did what you did, points are assigned based on the clarity of the code submitted (e.g., appropriate comments are provided).

Submitting Guesses to the Leaderboard (80% of Grade)

Please pay close attention to the guidelines below.

Your submission should be written in the provided **solution.py** file. To submit to the leaderboard, use the **submit.py** file, providing the necessary information. To submit your answer OR use this website, you MUST be on the Georgia Tech VPN, even when on campus.

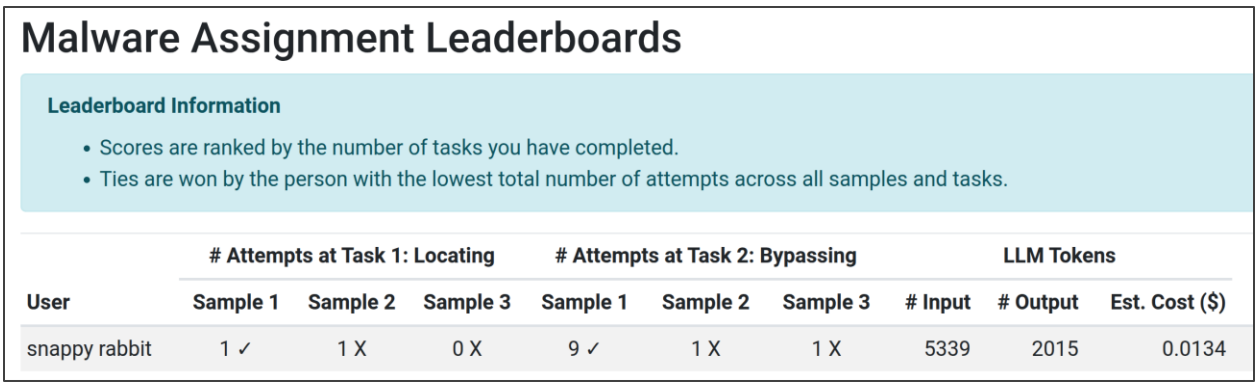
To view the results of your submissions, as well as your place on the leaderboard, select the malware analysis assignment at https://riposte-stu-20.storm.gatech.edu/assignments.

Leaderboard and Conduct Guidelines

This website is for 8813 purposes ONLY. To login to the site, you will need to provide the requested information on the website. That information generates an internal submission key stored on our server. No GT-related information is stored.

All website access is logged and monitored. Use only your assigned credentials. Do not attempt to exploit or interfere with the leaderboard system. Report any issues to the instructor. Violations of these guidelines will be treated as academic misconduct.

The live leaderboard displays performance metrics for each submission:



Malware Assignment Leaderboards

Leaderboard Information

- Scores are ranked by the number of tasks you have completed.
- Ties are won by the person with the lowest total number of attempts across all samples and tasks.

	# Attempts at Task 1: Locating			# Attempts at Task 2: Bypassing			LLM Tokens		
User	Sample 1	Sample 2	Sample 3	Sample 1	Sample 2	Sample 3	# Input	# Output	Est. Cost ($)
snappy rabbit	1 ✓	1 X	0 X	9 ✓	1 X	1 X	5339	2015	0.0134

Final Solution Submission Instructions:

Prior to the deadline, you must upload (to Canvas) a zip file [with the naming convention lastname1-lastname2-assignment2.zip] containing:

- The completed final_submission_details.txt file.

- All the code to complete tasks 1 and 2. Note that we will test your code on a new malware sample to see how successful it is in both identifying where the malware is evading the sandbox and generating python scripts to bypass the evasion.

We look forward to seeing you climb up the leaderboard!

References

[1] Miuyin Yong Wong, Matthew Landen, Manos Antonakakis, Douglas M. Blough, Elissa M. Redmiles, and Mustaque Ahamad. 2021. An Inside Look into the Practice of Malware Analysis. In Proceedings of the ACM SIGSAC Conference on Computer and Communications Security.