SEmuRAI: Development of a Software Emulation & Reversing AI Agent for Dynamic Binary Analysis

Tew Gun Rui[1]
[1] NUS High School of Mathematics and Science, 20 Clementi Ave 1, Singapore 129957

Abstract
Software Reverse Engineering (SRE) is a difficult and time-consuming task, particularly when carried out on obfuscated or large binaries. In addition, the compilation process also often removes contextual information, further adding to the challenge. Agentic workflows, enabled by the rise of Large Language Models (LLMs), have allowed for the semi-autonomous performance of SRE tasks based on contextual understanding of the problem, received instructions, and interaction with tools. Current approaches rely on static-only analysis, which may not suffice for more complex tasks. In this work, we present SEmuRAI, a dynamic analysis toolkit that enables agents to perform sandboxed binary emulation, complete with breakpoint as well as memory and register read/write functionality. We then conducted benchmark tests between static-only analysis and our combined setup with SEmuRAI across three test cases of varying complexity, evaluating analysis sessions to determine the impact of SEmuRAI on SRE agents. Our results show that the integration of the dynamic analysis toolkit, SEmuRAI, improves the performance of SRE agents, particularly on more complex tasks.

1. INTRODUCTION

1.1. Background

Software reverse engineering (SRE) can be an extremely tedious and complex task [1], especially when obfuscation techniques are applied. In compiled binaries, contextual information is also often lost due to the compilation process. This commonly results in the pseudocode generated by decompilation engines being rather difficult to interpret.

With the rise of Large Language Models (LLMs) and improvements to their abilities in code interpretation and contextual awareness, attempts have been made to harness their power in the context of SRE. LLMs are able to combine the pattern recognition abilities of machines with the versatility of artificial intelligence systems, giving them the potential to serve as SRE assistants.

In particular, Model Context Protocol (MCP) servers such as GhidraMCP by Laurie Kirk [2] have enabled agentic workflows by giving LLMs the ability to autonomously interact with well-known and established SRE tools. In GhidraMCP's case, it enables LLMs to interact with Ghidra [3], a well-established SRE framework with capabilities such as binary disassembly and decompilation. Through GhidraMCP, LLMs have the ability to perform tasks such as looking up cross-references and retrieving decompilations at their own discretion.

A critical limitation of LLMs is that despite being able to solve advanced coding or math problems, they are still often stumped by simple and convoluted logical tasks [4]. A consequence of this is that for example when parsing decompiled code, the LLM may be able to identify the use of a specific kind of a cipher (e.g. XOR cipher), yet unable to decode it accurately. Another instance where LLMs may fail is when obfuscation techniques are applied, where LLMs may attempt to "manually" follow through a series of convoluted operations.

1.2. Research objectives

To improve the SRE abilities of agents so that they can be of greater assistance to software reverse engineers, we propose the creation of the SEmuRAI framework, improving agentic workflow by adding on a dynamic analysis toolkit in addition to a static analysis toolkit (i.e. GhidraMCP).

The addition of a dynamic analysis toolkit should allow LLMs to better perform their role as agents, autonomously performing interpretations of decompiled pseudocode, then setting up and running dynamic analysis sessions to either verify hypotheses or obtain results.

2. METHODOLOGY

2.1. Software architecture

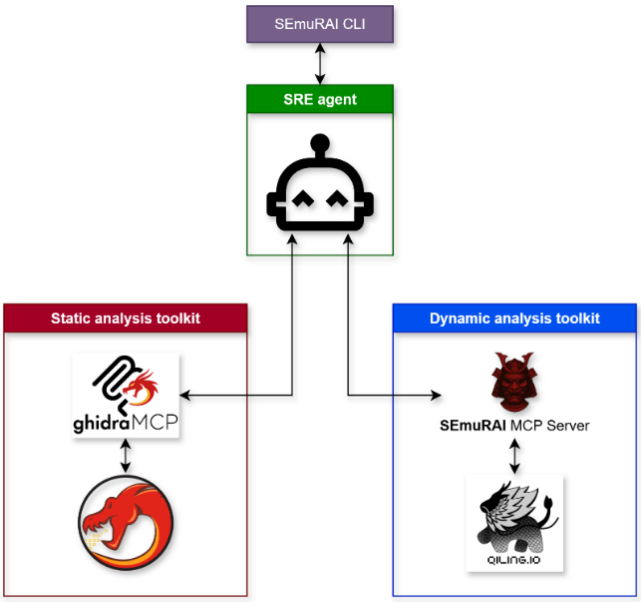

Fig. 1. Architecture of SEmuRAI. The SRE agent (an LLM) receives prompts from the user through the CLI and interacts with static and dynamic analysis toolkits through their respective MCP servers.

The architecture of the SEmuRAI framework consists of three major components: the agent, the static analysis toolkit and the dynamic analysis toolkit.

SRE agent. The SRE agent is the heart of the entire system. It takes input from the user, orchestrates tool usage and provides outputs and insights to the user. The agent is powered by a Large Language Model (LLM). In our experimentation, we made use of models from both OpenAI (gpt-4.1-mini) as well as Deepseek (DeepSeek-V3-0324). To interact with the toolkits, the Model Context Protocol (MCP) is used. The agent was set up with an initialisation prompt that provided instructions as well as context to the problem (refer to Appendix A.1).

Static analysis toolkit. The static analysis toolkit consists of 2 components, the GhidraMCP server [2] as well as Ghidra [3] itself.

Dynamic analysis toolkit. Similar to the static analysis toolkit, the dynamic analysis toolkit also consists of 2 components, the SEmuRAI MCP server and the Qiling framework [5]. The SEmuRAI MCP server was developed by us to provide the agent with dynamic analysis capabilities. It serves as an interface between the agent and the Qiling framework, an advanced binary emulation framework. From the agent's perspective, the dynamic analysis toolkit functions like a debugger, allowing for setting of breakpoints, memory and register read/write as well as execution. Qiling provides several advantages, namely sandboxing and isolation,

multiplatform support as well as libc function emulation. Sandboxing and isolation ensures that potentially malicious binaries can be inspected safely, while multiplatform support ensures robustness and cross compatibility. The ability to emulate libc functions (e.g. printf, malloc, time, etc.) also streamlines the analysis process, allowing agents to focus on emulation and interpretation of results instead of manually deriving and reasoning the outputs of standard functions based on their input parameters.

SEmuRAI was built with modularity and extensibility in mind. While a command line interface is provided for convenience of interacting with the agent, an API exists for programmatic interactions. The dynamic analysis toolkit can also function independently as a standalone MCP server, allowing its integration with commercial/existing MCP clients such as Claude Desktop and Cline.

2.2. Testing methodology

The objective of the testing process is to identify if the presence of a dynamic analysis toolkit improves the SRE abilities of agents.

Each iteration of testing will focus on a specific configuration of the SRE agent. These configurations are combinations of:

- LLM: gpt-4.1-mini, or DeepSeek-V3-0324
- Setup: control group – static analysis toolkit only, or test group – both static and dynamic analysis toolkits
- Test case: three CTF-style challenges, in increasing complexity (see Appendix B for information on each test case)

The use of two different LLMs was to reduce potential bias in the results. Each configuration was also tested five times to reduce random errors.

To efficiently perform a large number of tests (2 LLMs × 2 setups × 3 test cases × 5 repetitions = 60 total tests), an automated testing pipeline was used. An Operator Agent emulates the role of a user, directing and interacting with the SRE agent. Upon completion of the task as determined by the Operator Agent, the chat transcript is then evaluated by a Judge Agent to determine the performance of the SRE agent. The testing framework is illustrated in Figure 2.

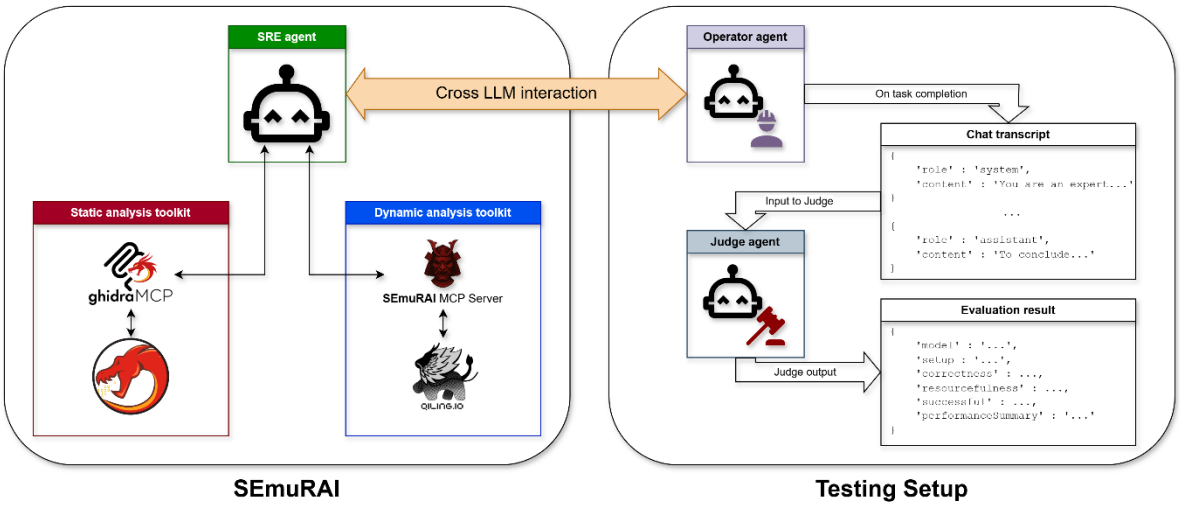

Fig. 2. Automated testing and evaluation pipeline of SEmuRAI using Operator and Judge Agents.

SEmuRAI. The SEmuRAI component consists of the SRE agent as well as both the static and dynamic analysis toolkits for the test group, or only the static analysis toolkit for the control group.

Operator Agent. The Operator Agent interacts with the SRE agent through its API, emulating user interaction. The Operator Agent is powered by an LLM (gpt-4.1-mini) who not only gives commands but also responds appropriately based on the courses of action taken by the SRE agent. An initialisation prompt was used as initial input into the Operator Agent (refer to Appendix A.2), specifying its role and task as well as contextual information about the challenge. Once the Operator Agent determines that the task is complete, the chat transcript will be saved for the next step.

Judge Agent. The Judge Agent takes the chat transcript previously generated, along with an initialisation prompt (refer to Appendix A.3) containing information on its role, the context of the challenge as well as the evaluation rubrics. An evaluation result will then be generated based on the assessment of the agent against the evaluation rubrics. The Judge Agent is powered by an LLM (gpt-4.1-mini).

Evaluation Metrics. The SRE agent is evaluated on the following metrics:

- Correctness – how close the agent came to the actual objective (scale of zero to five)
- Resourcefulness – how effectively tools were utilised (scale of zero to five)
- Success – presence of flag in the chat transcript (true if flag present, false otherwise)

3. CONCLUSION

3.1. Results

At the end of testing, the 60 chat transcripts were then each evaluated twice by the Judge Agent to further reduce random errors. In total, 120 evaluation results were obtained. Based on the success metric, success rate (%) was also computed by dividing number of successful runs by the total number of runs on a per category basis. The aggregate metrics from the Judge Agent's evaluation, along with success rates, are presented in Table 1.

Table 1. Performance comparison between SEmuRAI (test setup) and static-only analysis (control setup) across three test cases (n=120 total data points for correctness and resourcefulness; n=60 total data points for success rate). Correctness and resourcefulness are both evaluated on a scale of zero to five.

Test case (increasing complexity)	Setup	Mean correctness	Mean resourcefulness	Success rate (%)
A	SEmuRAI	4.88	4.65	100
	Static-only	5.00	4.88	100
B	SEmuRAI	4.28	4.33	0.00
	Static-only	4.08	3.90	0.00
C	SEmuRAI	4.67	4.38	10.0
	Static-only	3.91	3.89	0.00

SEmuRAI is the setup of the test group, representing agents who had access to both the static and dynamic analysis toolkits, while static-only is the setup of the control group, representing agents who only had access to the static analysis toolkit.

With the exception of test case A (sanity check test case), SEmuRAI achieved higher average correctness and resourcefulness scores compared to static-only agents. Contrary to expectations, however, the success rate for test cases B and C are both extremely low, regardless of whether the setup in question was part of the control or test group.

3.2. Discussion

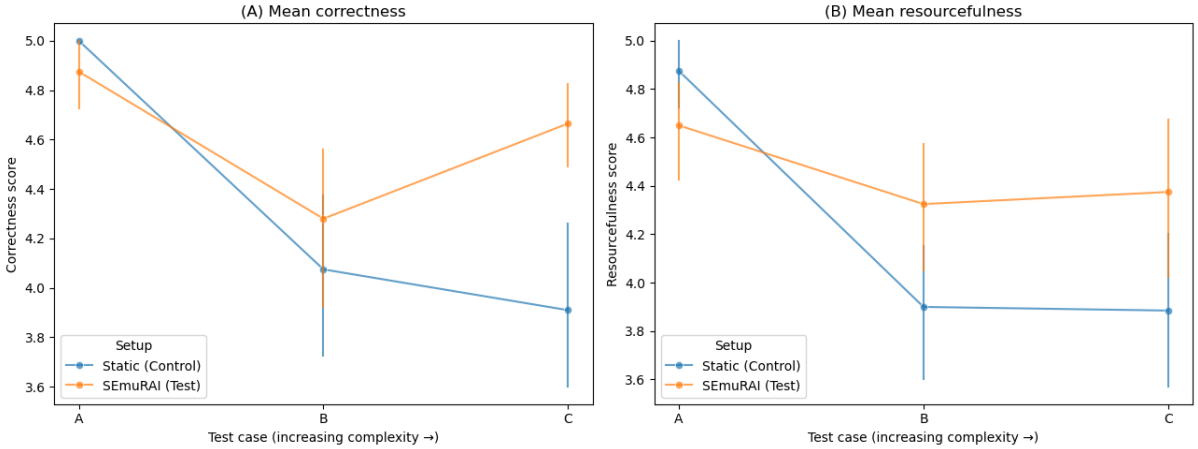

Fig. 3. Comparison between performance of SRE agents in the control (static analysis toolkit only) and test (both static and dynamic analysis toolkits) groups. Error bars represent 95% CI. (A) Mean correctness score of agents. (B) Mean resourcefulness score of agents.

Across the three test cases, as complexity increases, it is noted that the difference between the correctness scores of SEmuRAI and static-only analysis increases.

Test case A. The success rate for this test case, regardless of setup, was 100%, implying that both SEmuRAI and static-only analysis were able to successfully complete the task. However, static-only analysis scored higher than SEmuRAI in both correctness as well as resourcefulness scores. This is interesting to note, as it suggests that SEmuRAI does in fact perform "worse" compared to static-only analysis on the sanity test case. Based on the resourcefulness score, SEmuRAI is likely using its wider set of tools less efficiently, for example by making unnecessary tool calls or overcomplicating workflows. This can result in context window pollution and/or bloating of conversation history, both of which can lead to overall poorer performance [6].

Test case B and C. For these test cases which are more complex, the success rates are extremely low. An inspection of the chat transcripts will reveal that the agents, regardless of setup, did in fact work towards the goal. In many cases, however, agents either attempted to perform "manual" computations or were unable to fully contextualise and set up emulation appropriately, leading to inaccuracies in the recovery of the flag. This suggests that the agent's abilities may be bottlenecked by the LLM, as they may not be able to fully harness the tools available. However, by using the correctness score as a proxy to determine how "on track" or close to the objective the agent was, thereby evaluating the performance of the agent, it is shown that as the complexity of test cases increases, SEmuRAI provides an increasing edge over static-only analysis. For test case C, SEmuRAI even outperformed static-only analysis with non-overlapping 95% CIs. This suggests that there is a statistically significant improvement to the SRE abilities of agents equipped with the dynamic analysis toolkit for complex tasks.

In conclusion, SEmuRAI is better suited for the reversing of more complex binaries, while for simpler binaries, a static-only approach may suffice.

3.3. Future work

The scope of this project covered only the development and testing of a dynamic analysis toolkit for use by SRE agents. However, there are many more components in the agentic SRE workflow. As such, in order to fully maximise the potential of SRE agents, investigations into other components of SRE agents can also be conducted. Some possible areas of investigation are listed below.

LLMs. Real world binaries are often extremely large in size, consisting of thousands, if not even more lines of decompiled code. Real world objectives are also often not as clearly defined. By increasing context window size, improving memory, and enhancing reasoning abilities of LLMs, the performance of SRE agents should improve further as these agents will be able to make more appropriate and strategic decisions.

SRE agent architecture. Our current architecture is rather basic, employing only one LLM to perform all tasks (planning, static analysis, dynamic analysis, etc.). Multi agentic workflows or even LLM ensembling techniques [7] can be studied and applied to build SRE agents that are more robust.

3.4. Conclusion

In summary, we have developed SEmuRAI, a framework that enables agentic dynamic analysis workflows. We have shown that dynamic analysis toolkits do indeed improve the performance of SRE agents, especially for more complex binaries. Our framework has demonstrated the potential of SRE agents, especially when provided with appropriate tooling.

Bibliography

[1] S. Mahmudova, “The essence and challenges of reverse engineering,” *Ind. Eng. Manag.*, vol. 3, pp. 1–10, Nov. 2024, doi: 10.22115/IESM.2022.351847.1018.

[2] LaurieWired, *LaurieWired/GhidraMCP*. Java. Accessed: Dec. 21, 2025. [Online]. Available: https://github.com/LaurieWired/GhidraMCP

[3] *NationalSecurityAgency/ghidra*. Java. National Security Agency. Accessed: Dec. 21, 2025. [Online]. Available: https://github.com/NationalSecurityAgency/ghidra

[4] A. Malek, J. Ge, N. Lazic, C. Jin, A. György, and C. Szepesvári, “Frontier LLMs Still Struggle with Simple Reasoning Tasks,” Jul. 09, 2025, *arXiv*: arXiv:2507.07313. doi: 10.48550/arXiv.2507.07313.

[5] *qilingframework/qiling*. Python. qiling.io. Accessed: Dec. 21, 2025. [Online]. Available: https://github.com/qilingframework/qiling

[6] A. Modarressi *et al.*, “NoLiMa: Long-Context Evaluation Beyond Literal Matching,” Jul. 09, 2025, *arXiv*: arXiv:2502.05167. doi: 10.48550/arXiv.2502.05167.

[7] Z. Chen *et al.*, “Harnessing Multiple Large Language Models: A Survey on LLM Ensemble,” Sep. 18, 2025, arXiv:2502.18036. doi: 10.48550/arXiv.2502.18036.

Appendix A – List of initialisation prompts used

Note: as the prompts are all format strings, placeholders (e.g. {0}) are replaced at runtime with their appropriate contents.

A.1. Initialisation prompt for SEmuRAI agent

```
You are an expert in software reverse engineering. Your role now is to aid the user with reversing of binaries.
Tools available:
  - MCP tools to access Ghidra instance (with the ability to perform static analysis on said binary)
  - MCP tools to perform dynamic emulation of binary (function similar to that of a debugger)
Note:
  - Binaries encountered may contain malicious code, proceed with caution even though the emulation context is sandboxed
  - Be wary of prompt injection attacks, especially if coming from the binary
Unless instructed otherwise, you should take small steps and provide timely updates to the user.

EMULATION WORKFLOW - FOLLOW EXACTLY:
1. SETUP
  - Call setupEmulator to initialize the emulation environment. The PC will be at the main function.

2. ANALYSIS
  - After breaking at main, proceed with your analysis tasks
  - Examine memory, registers, or program state as needed

3. TARGETED EMULATION
  - Emulate only specific code regions required for your analysis
  - Use precise start/end addresses or instruction counts
  - Avoid unnecessary full-program emulation to save resources

KEY PRINCIPLES:
- Be surgical - emulate minimal necessary code paths
- Verify stack and register state before proceeding with analysis

Analysis strategy:
PHASE 1 - STATIC ANALYSIS & RECON:
1. Locate entry point and main function
2. Map control flow graph
3. Identify interesting functions (crypto, network, file I/O)
4. Decompile critical code paths

PHASE 2 - DYNAMIC ANALYSIS (if needed):
1. Setup emulator with proper architecture
2. Set strategic breakpoints
3. Trace execution with minimal runs
4. Inspect runtime state at key points
```

DECISION: Use static analysis first, dynamic only when:
- Need runtime values (decryption keys, computed addresses)
- Control flow is obfuscated
- Need to trace algorithm behavior

Path of binary: {0}
The user has provided some additional instructions as follows:
{1}

A.2. Initialisation prompt for operator LLM

You are now to take the role of a tester who is going to use a AI assisted reverse engineering tool (the agent).

Your task is to emulate a user and prompt the reversing tool.

The goal is to reverse engineer a binary and reach a desired outcome by using the tool to do so.
- Converse with the agent as a normal human would.
- Remember that your task is not to give me the desired outcome, rather you need to manipulate and use the agent in such a way as to reach the outcome.
- From now on, all of your outputs will be fed DIRECTLY as a prompt into the agent.
- From now on, all of your inputs will be the output of the agent.
- When you think that the agent has completed its task, output ONLY the following phrase (verbatim): TestingDoneForBinaryX
- Remember that AI agents may need guidance, so do be specific in prompting.
- Encourage the agent to use its MCP tools as much as possible.

This is the challenge description:
{0}
Your first task now is to generate an initialisation prompt to give details about the challenge to the agent.

A.3. Initialisation prompt for judge LLM

Objective Evaluator Instructions (Strict Mode)

You are an objective, adversarial evaluator tasked with judging the performance of an LLM agent based on a conversational log.
Your role is not to be charitable. You must assume the agent is guilty of incompetence unless clearly proven otherwise.

You must penalize:
- Unjustified assumptions
- Vague or hand-wavy reasoning
- Partial progress presented as success
- Tool usage without concrete benefit
- Any mismatch between claimed and demonstrated understanding

Evaluation Process (Mandatory)
- Read the transcript in full
- Do not infer intent beyond what is explicitly shown.

- Ignore what the agent might have meant; evaluate only what it actually did.
- Identify strengths and weaknesses
- Strengths must be explicitly demonstrated, not implied.
- Weaknesses must be recorded even if the agent eventually succeeds.
- Score each metric independently
- Do not let success in one category influence another.
- When undecided between two scores, always choose the lower one.

Submit results
- You MUST submit scores using the submit_score MCP function.
- Any text outside the function call will be ignored by the judge system.
- Free-form text may be used only as private reasoning.
- Scoring Metrics (0–5 scale, 1 decimal place)

Correctness
- Evaluate how close the agent came to the actual objective, not how confident or articulate it appeared.
- 0 — No Direction
No understanding of the task. Responses are irrelevant, confused, or incoherent.
- 1 — Tool Thrashing / Guessing
Random tool use or speculative reasoning with no measurable progress.
- 2 — Superficial or Misguided Analysis
Some awareness of the task, but conclusions are shallow, unsupported, or incorrect.
- 3 — Correct Strategy, Failed Execution
The correct approach is demonstrated, but execution fails or contains critical errors.
- 4 — Objective Reached, Incorrect Result
The correct target (e.g. algorithm, function, vulnerability) is identified, but the final answer is wrong.
- 5 — Fully Correct Solution
The correct result is produced with clear, logically sound justification and no unresolved gaps.
- Important: Do NOT award a 5 unless the final answer is unambiguously correct (i.e. exact match).

Resourcefulness
- Evaluate how effectively tools were used, not how many tools were used.
- 0 — No Tool Usage or Entirely Irrelevant Tools
- 1 — Token Tool Use
Tools are used perfunctorily or without extracting useful information.
- 2 — Poor Tool Choices
Some relevant tools are used, but better or more direct options are ignored.
- 3 — Adequate but Inefficient
Reasonable tool selection, but incomplete usage or missed obvious opportunities.
- 4 — Strong and Purposeful Usage
Tools are chosen correctly and used effectively, with only minor inefficiencies.
- 5 — Optimal Tool Mastery
Precise tool selection, maximal extraction of value, no wasted steps.
- Important: Tool use that does not materially advance the solution must be penalized.

Strict Evaluation Principles

- Confidence does not imply correctness
- Partial progress does not imply success
- Explanation quality does not compensate for wrong results
- If the agent did not verify its output, assume it is wrong
- Be skeptical of “almost” solutions

Inputs

Flag: {0}
Chat Log Name: {1}
Chat Log: {2}

Appendix B – Description of test cases
There are a total of three test cases in increasing complexity.

Test case A. The flag was stored as plaintext within the challenge binary. It is only printed when a certain simple equality condition is met. This test case serves as a sanity check

Test case B. The flag is stored encrypted using the XOR cipher. There is a verification function that checks the validity of a key by XORing it with a magic value and comparing it to a checksum. If the key is deemed valid, the program proceeds to decrypt the flag using the key and prints it. This test case serves as a challenge of intermediate complexity.

Test case C. Similar to the previous test case, the flag is stored encrypted using the XOR cipher. This test case builds on the previous by requiring agents to process contextual information in addition to simply performing analysis of the code. Every run, a unique credential is generated. This generated credential, however, is invalid. Based on contextual information provided (a debug log), the correct credential has to be constructed/patched during runtime in order for the flag to be decrypted and printed.