

MARVEL: Multi-Agent RTL Vulnerability Extraction using Large Language Models

Luca Collini*
NYU Tandon
Brooklyn, NY 11201
lc4976@nyu.edu

Baleegh Ahmad*
NYU Tandon
Brooklyn, NY 11201
ba1283@nyu.edu

Joey Ah-kiow
NYU Tandon
Brooklyn, NY 11201
ja4844@nyu.edu

Ramesh Karri
NYU Tandon
Brooklyn, NY 11201
rkarri@nyu.edu

Abstract

Hardware security verification is a challenging and time-consuming task. For this purpose, design engineers may utilize tools such as formal verification, linters, and functional simulation tests, coupled with analysis and a deep understanding of the hardware design being inspected. Large Language Models (LLMs) have been used to assist during this task, either directly or in conjunction with existing tools. We improve the state of the art by proposing MARVEL, a multi-agent LLM framework for a unified approach to decision-making, tool use, and reasoning. MARVEL mimics the cognitive process of a designer looking for security vulnerabilities in RTL code. It consists of a supervisor agent that devises the security policy of the system-on-chips (SoCs) using its security documentation. It delegates tasks to validate the security policy to individual executor agents. Each executor agent carries out its assigned task using a particular strategy. Each executor agent may use one or more tools to identify potential security bugs in the design and send the results back to the supervisor agent for further analysis and confirmation. MARVEL includes executor agents that leverage formal tools, linters, simulation tests, LLM-based detection schemes, and static analysis-based checks. We test our approach on a known buggy SoC based on OpenTitan from the Hack@DATE competition. We find that 20 of the 48 issues reported by MARVEL pose security vulnerabilities.

1 Introduction

Detection of hardware security vulnerabilities in Register-Transfer Level RTL code is a time-consuming and challenging process⁵. Considerable research efforts have been dedicated to making this validation process easier. These include using deterministic methods like formal methods^{10;18;17}, information flow tracking^{9;3}, fuzzing^{16;21}, and, more recently, non-deterministic methods involving the use of large language models (LLMs)^{1;8;20}. LLMs have been used as debuggers/linters⁶. They may use general guidelines for the detection of bugs, such as Common Weakness Enumerations (CWEs), or may use external information (outside of source code), such as design specifications, to aid in detection^{19;2}. They may use external tools such as linters, formal verification, and/or simulators to assist them in debugging²³. While these approaches show great promise, they still require a preset action plan. The LLM’s decision-making process may be used to determine if a segment of code is insecure, but it is not in control of the workflow of tool usage or gathering information.

This gap in autonomous thinking can be filled by using the LLMs in an agentic workflow. By giving an LLM the ability to develop a plan, use a tool when appropriate, and control the flow within a framework, the LLM acts as an agent capable of making *human-like* decisions²². This better simulates the thought process of an RTL designer or verification engineer while debugging a digital design. Furthermore, the bug-hunting process might involve using multiple tools to localize the reason for a misbehaving design iteratively. This process can be simulated in a multi-agent framework¹³.

*Luca Collini and Baleegh Ahmad contributed equally to this work.

MARVEL is a multi-agent framework that implements a unified approach to decision-making, tool usage, and reasoning towards the goal of RTL bug detection. It uses a *Supervisor-Executor* architecture¹¹. The supervisor manages communication and coordination among specialized executor agents. Each executor uses a unique bug detection strategy coupled with the tools required to implement its strategy. **Linter Agent**, **Assertion Agent**, **CWE Agent**, **Similar Bug Agent**, **Anomaly Agent**, and a **Simulator Agent** are the executor agents. The supervisor agent identifies the security objectives relevant to the design by traversing through directories, source code, and design specification documents. Then it calls one executor agent at a time to determine if a given file of code has a vulnerability that violates the security objective. The supervisor may use multiple executors before determining whether the security objective is satisfied or not. In this process, security bugs are identified, and a report is provided to the user with a summary of the security issues. An overview of the multi-agent supervisor-executor flow is shown in Figure 1.

While a similar strategy has been used for automated bug repair for software¹², research efforts for hardware description languages (HDLs) do not present a comprehensive approach. We take inspiration from software works and develop targeted solutions for RTL by integrating RTL static analysis tools and handling the unique challenges faced with HDLs. These include identifying hardware security objectives, using hardware CWEs, forming security assertions, and reasoning about the outputs from these tools from a digital design point of view. The main contributions of this work are:

- First comprehensive framework that uses a multi-agent bug detection framework (MARVEL) for bug detection in complex hardware designs Section 2.
- Evaluation of the MARVEL framework on a known buggy dataset taken from the Hack@DATE 2025 based on the OpenTitan SoC Section 4.
- Open sourcing implementation and results for the community through our anonymized repository.

2 MARVEL

We propose MARVEL, i.e., **Multi-Agent RTL Vulnerability Extraction using LLMs**. We implement MARVEL with a *Supervisor-Executor* architecture. The components of MARVEL are in Figure 2. Each LLM-based block can be implemented with different models and parameters. We ran the framework with different models and parameters on a few instances to identify which of the currently available models performed better at for the specific agents. Our model and parameters selection is specified below for each agent. As models improve, they can simply be added to MARVEL.

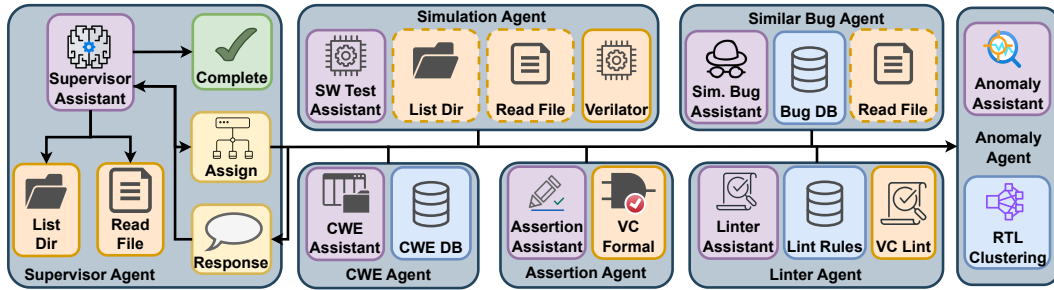


Figure 2: MARVEL’s Multi-Agent Framework. Purple denotes LLM assistants, Orange denotes tools, and Blue denotes RAG databases. The Supervisor Agent has the ability to read from files and assign tasks to executor agents. From the responses, it may decide to continue assigning tasks or determine that the security analysis is complete. The Simulator, Similar Bug, CWE, Assertion, Linter and Anomaly agents are the six executor agents. Each executor agent takes a security objective and source rtl file from the Supervisor Agent and determines whether the security objective is met.

2.1 Overview

LLM-based agents are typically explained using four elements: Profile, Memory, Planning, and Action. Not only does MARVEL use these modules holistically, but all seven agents of MARVEL utilize

these components individually as well. Each agent consists of an assistant and tools. The assistant is an LLM agent with a specific objective, responsible for decision-making, capable of calling one or more tools. The tools are responsible for carrying out the assistant’s recommended actions when called and replying with the result or with an error message.

Profile: describes the roles of an agent which are usually indicated in the prompts to the LLM. This may include priming the LLM as a domain expert and providing it with its overall big picture task. For MARVEL, all agents (supervisor and executors) are profiled using system prompts.

Memory: stores information obtained from the environment and leverages the recorded memories to facilitate future actions. Memory helps the agent to accumulate experiences, evolve, and behave in a more consistent and effective manner. LLM agents use two kinds of memory. Short-term memory is the information present in the context window that the LLM sees in the prompt. Long-term memory is the external vector storage that agents can query and retrieve from. For MARVEL, the external vector storages are 1) CWE Database used by the CWE-Agent, 2) known Bug Database used by the Similar Bug-Agent, and 3) Lint Rules Database used by the linter agent. Retrieval Augmented Generation (RAG) is used by the respective agents to retrieve relevant information from these databases.

Planning: is the decision-making process based on the information available to the LLMs (either from the initial prompt or from previous external tool calls). LLMs use this reasoning ability to make judgments about the workflow path to take, the tools to use, and when to conclude that a task is done. For the supervisor agent, this planning is illustrated in deciding which executor agent to use, deciding whether there is enough information to make a conclusion about a reported security issue, and deciding that all security objectives have been evaluated and that it is time to produce the final report. For each executor agent, the corresponding assistant does this planning, determining whether the tool needs to be called and whether there is enough information to report back to the supervisor.

Action: translates the agent’s decisions into specific outcomes. This can be done through using external tools or by the LLMs themselves. The tools for the supervisor agent include but are not limited to the executor agents, and the tools for the executor agents are the linter, formal, simulator, and clustering tools.

2.2 Supervisor Agent

The *supervisor* orchestrates the *executor agents* to perform the security analysis. The supervisor agent’s system prompt is shown in Figure 3. The system prompt hints at the order and usage of the executor agents, but the supervisor agent may call on them in any order and multiple times. The supervisor can explore the system design by listing folder content and reading files. Files can be retrieved with and without annotated file numbers. The latter is helpful for code files, helping the agent report buggy line numbers correctly. Ultimately, the supervisor agent produces a report on the security issues found in the design. We used gpt-4.1 as the LLM for the supervisor agent, selecting a temperature of 0.14. We found the supervisor agent would exceed the context window of smaller models, so we opted for this new OpenAI model, which has a context window of 1,047,576 tokens. Exploratory experiments showed that a low, non-zero temperature was beneficial for MARVEL.

Supervisor’s System Prompt

You are a supervisor agent in a multi-agent system focused on identifying hardware security vulnerabilities in RTL code. Your objective is to analyze the given SoC and generate a detailed security report.

You have access to the following tools: <tool_list>.

Each tool specializes in a specific task:

<tools description> Instructions for analysis:

- Read the documentation to identify security features and register interface policies.
- Use Verilator, Assertion, Anomaly and Linter agents to uncover initial issues.
- If a bug is detected but not localized, use the CWE Agent to further inspect the related security aspect in the surrounding RTL.
- After detecting any bugs, use the Similar Bug Agent to scan similar files (of the same or of different IPs) for similar vulnerabilities.

Output Format:

<output_format_instructions> When your analysis is complete, end your response with "END".

Figure 3: Supervisor Agent’s System Prompt. It is instructed to analyze given SoC for security bugs. It is provided information about the executor agents and is tasked to produce a security report.

2.3 Linter Agent

The *linter agent* consists of the *linter assistant*, *lint tags retriever*, and the *lint checker*. Inputs to the linter agent are the security objective and the source code file to be analyzed for the security objective. We use gpt-4.1-mini with a temperature of 0.2 for this agent due to its favorable

cost-to-performance. The linter assistant has the ability to make calls to either the lint tags retriever or the lint checker until the task is completed or the maximum number of iterations, i.e. 6, is reached.

When used, the linter assistant first queries the lint tags retriever to obtain lint tags relevant to the specified security objective. The lint tags retriever uses the security objective to search 1255 lint tags from Synopsys's VC SpyGlass linting tool's Rules Reference guide and extract tags relevant to the security objective. We keep an upper limit of 20 selections per query. We construct a text file with each line consisting of the tag identifier, followed by a small description of what the tag is checking for. Then, the linter assistant uses the lint tags to execute the lint checker on the source code file and obtain potential lint violations. The lint checker tool populates a template tcl script for running VC SpyGlass Lint¹⁴ with the identified lint tags and top module under analysis. The results output file contains the lint checks violated by the digital design. The contents of the results file are sent back to the linter assistant. If any violations are found, the linter assistant completes further analysis to filter out false positives. This analysis is crucial to reducing the false positives observed from linter tools and is a key advantage of our linter agent. If issues arise while running the tool, such as unknown lint tags or top module identifiers, a message with an exception is sent back to the linter assistant for debugging. An example of the flow of the linter agent's operation on an improper FSM in the ADC Control FSM module is shown in Figure 4.

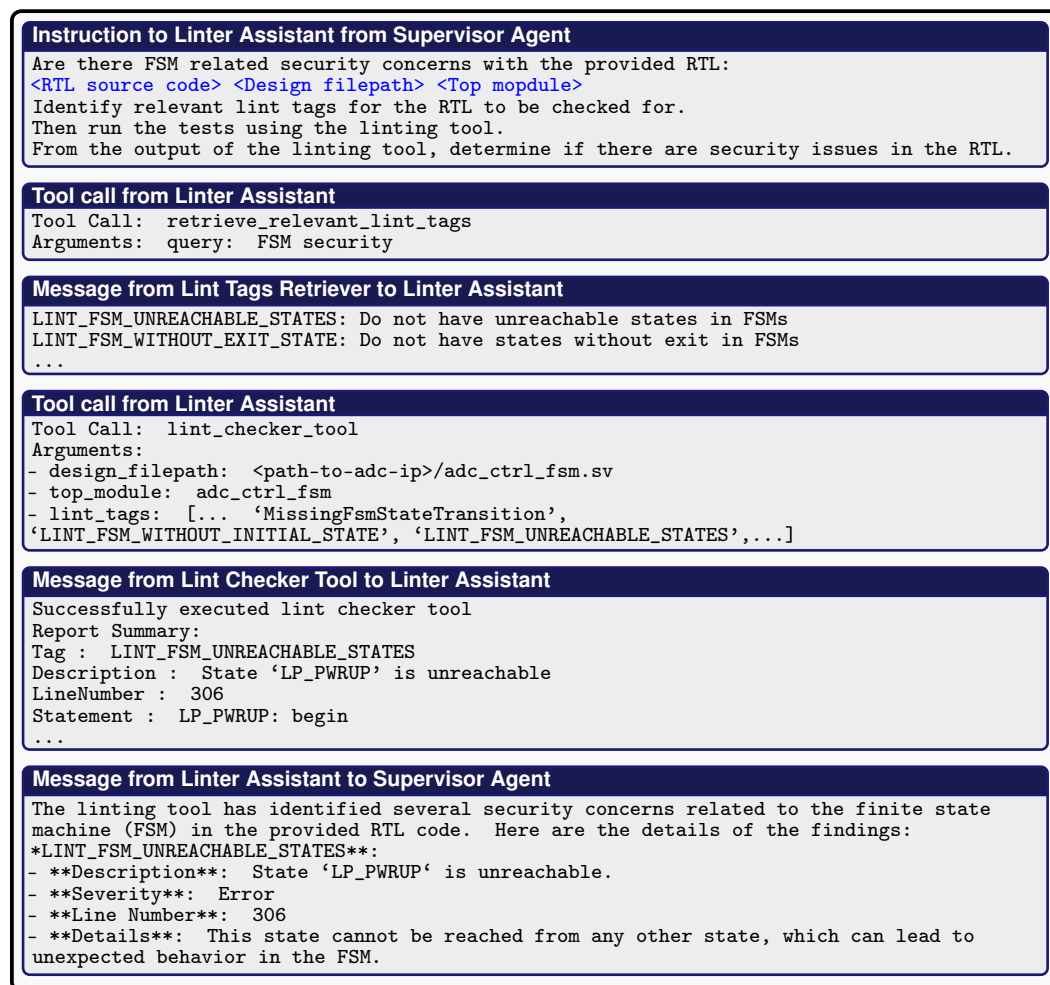


Figure 4: Example linter agent flow for Improper FSM on ADC Control FSM module. Linter assistant receives instructions to find FSM security concerns. It calls the lint tags retriever to obtain FSM-related tags and then calls the lint checker to execute the lint tags on the ADC Control FSM. From the output of the checker tool (LINT_FSM_UNREACHABLE_STATES tag violated), the assistant determines that state LP_PWRUP is unreachable and points to line 306 as the bug location.

2.4 Assertion Agent

The *assertion agent* consists of the *assertion assistant* and the *assertion checker tool*. The inputs to the assertion agent are the security objective and the source code file. We use Sonnet 3.7 with a temperature of 0 because we found it performed better in forming meaningful assertions. The assertion assistant forms relevant System Verilog assertion(s) for the security objective to check the RTL for. The assertion checker then checks these assertions. The assertion agent can call the assertion checker until it produces a falsified assertion or until the max iterations (6) is reached. The assertion assistant uses an example of the typical structure for a concurrent assertion to form meaningful assertion(s). Then it runs the assertions using the assertion checker tool. From the output of the assertion checker tool, the assertion assistant determines if there are security issues. If no falsified assertions result from the assertion checker tool, the RTL has no identified security issues. The assertion checker tool binds the assertions to the source code, obtains all the dependencies on the source code, populates the Tcl template script with the top module, clock, and reset signals, and runs Synopsys' Formal Property Verification (FPV)⁷ tool. The results output file contains the assertions falsified by the digital design. Its contents are sent back to the assertion assistant. If there was some issue running the tool, such as syntactically incorrect assertions, a message with an exception is sent back to the assertion assistant. An example of the flow of assertion agent's operation on `hmac_reg_top` module is shown in Section A.1.1.

2.5 CWE Agent

The *CWE agent* comprises the *CWE assistant* and the *CWE details retriever tool*. The inputs to the CWE agent are the security objective and the source code file. We use the Sonnet 3.7 model with a temperature of 0.2, as we found it yielded the best results in reasoning about the relevance of CWEs to insecure code instances. The CWE assistant can call the CWE details retriever until the task is completed or the number of maximum iterations, i.e. 6, is reached. Typically, the assistant is asked to identify the CWE relevant to the security issue for the RTL. Details of the CWE are augmented with the task of determining whether there are security issues relevant to this CWE in the RTL.

The CWE details retriever does two operations. First, it uses the security objective to identify the relevant CWE-ID. Then, it uses the CWE-ID to obtain the extended description and coded examples of CWE instances and corresponding repairs. We use MITRE's website to obtain the CWE name, description, extended description, and examples. We construct a text file with each segment consisting of the CWE identifier and description of the CWE. This text file is chunked using a recursive character text splitter into sizes of 50, an overlap of zero, and a custom separator to indicate separation between CWEs. These chunks are stored in a vector database so the retriever can obtain information using the embedding scores of the query and the chunks. We query for one selection to choose the most relevant CWE. Once the relevant CWE is identified, its extended description and examples are augmented, and the output is sent to the CWE assistant. If the tool ran into an issue, such as an unknown CWE-ID, a message with an exception is sent to the CWE assistant. An example of the flow of the CWE agent's operation on the OTBN MAC Bignum module is shown in Section A.1.2.

2.6 Similar Bug Agent

The *similar bug agent* consists of the *similar bug assistant* and the *similar bug tool*. The inputs to the agent are a known buggy line of RTL and a target RTL file to search for similar bugs. We use the Haiku-3.5 model for this agent due to its reliable performance in identifying semantically similar RTL constructs and lower cost than Sonnet 3.7. The similar bug assistant can call the similar bug tool to retrieve lines in an RTL file similar to a provided one. Typically, the assistant is asked to identify RTL lines similar to a previously identified bug and determine whether these lines also constitute bugs. The similar bug tool reads the RTL and splits it into individual lines. The lines are embedded using OpenAI embeddings and stored in an in-memory vector store. A retriever is constructed on top of this store to return the top 10 matches semantically closest to the original bug line based on embedding similarity. The tool annotates the retrieved lines with the respective numbers and returns them to the assistant. The assistant then evaluates these returned lines to determine if they exhibit the same bug pattern as the input line. The assistant can also decide to inspect the file to find the context of the identified lines before making a final decision. A corresponding message is returned if no similar lines are found or the tool fails due to issues like a missing file. Otherwise, the assistant provides a list of confirmed buggy lines with line numbers as its final output. An example of the flow of the similar bug agent's operation on AES is shown in Section A.1.3.

2.7 Anomaly Agent

The *anomaly detection agent* consists of the *anomaly detector assistant* and the *RTL clustering tool*. The inputs to the anomaly agent are the security objective and the RTL source code file. This agent is designed to detect anomalous constructs in RTL code by clustering semantically similar lines and identifying outliers that may indicate potential security issues. We use `gpt-4.1-mini` as the primary reasoning model for the assistant, as it demonstrated effective performance in analyzing clustered constructs and identifying potential security issues. The anomaly detector assistant has the ability to call the RTL clustering tool to aid its task. The anomaly detector tool performs two main operations. First, it extracts all `assign` statements from the RTL. Then, it generates embeddings for each construct using OpenAI’s `text-embedding-3-small` model. These embeddings are clustered using DBSCAN with cosine similarity as the distance metric to form groups of semantically similar constructs. Constructs that do not belong to any cluster (i.e., outliers) are considered anomalous. These anomalies, along with the clusters they were compared against, are returned to the Assistant. The Assistant evaluates whether any of the anomalous lines may represent a security vulnerability. If no anomalies are found or if the tool encounters an error (e.g., missing file), a corresponding message is returned to the assistant. Otherwise, the assistant reasons over the anomaly data to determine if the anomalous lines constitute a security issue and outputs its final assessment. An example flow of the anomaly agent’s operation on HMAC Register Top module is shown in Section A.1.4.

2.8 Simulator Agent

The *simulator agent* comprises the *simulator assistant* and the *Verilator tool*, and is responsible for identifying potential security issues in RTL through simulation. The agent takes as input the name of an IP block and uses Verilator as the underlying simulation backend. We employ the Sonnet 3.7 model for this agent due to its robust capabilities in interpreting simulation results and logs. The simulator assistant can invoke the Verilator tool iteratively until no further tool calls are needed or the task is completed. The Verilator tool functions in two stages. First, it retrieves all available simulation software tests for the target IP by executing a `bazel query` command. Next, it constructs and runs a `bazel test` command to simulate the design using Verilator. The output, including logs for failing tests, is returned to the assistant. The assistant reviews this output, particularly the logs of failed tests, and evaluates whether any failures point to security-related bugs in the RTL. If no tests are found or the test execution fails, the agent reports accordingly. Otherwise, it summarizes the security issues identified, including explanations and locations in the RTL design. An example flow of the Simulator Agent’s operation on the HMAC module is shown in Section A.1.5.

3 Experimental Evaluation

3.1 Experimental Setup

We implemented MARVEL in Python, modeling the agentic framework using LangGraph. Our implementation is open source at anonymized repository. MARVEL is fully automated. The flow is independent of the LLMs used for the different agents. New models can be plugged in easily. We selected the model for each agent running the flow on a subset of three benchmarks, evaluating different models and temperature parameters. We settled on the cheapest model available through APIs that provided satisfactory results. We did not employ reasoning models as they are expensive, slower, and do not necessarily provide better results in the electronic design automation domain⁴.

3.2 Benchmark

We evaluate MARVEL on a vulnerable OpenTitan *earlgrey* System-on-Chip (SoC) design¹⁵ obtained from the finals of the Hack@DATE 2025 competition. Hack@DATE is a premier hardware security capture-the-flag competition. They provide contestants with an SoC design with manually inserted vulnerabilities akin to those found in real, deployed products. The earlgrey SoC is a high-quality, open-source Root-of-Trust design that provides robust hardware security features. It uses the *Ibex* RISC-V processor as the main core and integrates various intellectual property (IP) blocks as peripherals, including crypto accelerators for AES, system management units for clock, power, and reset, and I/O protocols like SPI. The SoC has an array of security features, including end-to-end data integrity, secure boot, and side-channel first-order masking.

The IPs we analyze with MARVEL are summarized in Table 1. We report the total number of files, design files, and design LoC of each IP. Design files only include those used to implement the IP (i.e., excludes test files) and design LoC is the line count in those files excluding comments and whitespace. We select these 12 IPs because they represent a wide range of functionality, ranging from cryptography (e.g., AES) to I/O (e.g., ADC) and system management (e.g., lifecycle controller).

Table 1: Design IPs from OpenTitan earlgrey SoC used to evaluate MARVEL and their design size.

Design IP	Description	Total Files	Design Files	Design LoC
adc_ctrl	Control and filter logic for dual Analog-to-Digital Convertor analog block.	59	7	4159
aes	Cryptographic accelerator for the Advanced Encryption Standard (AES).	203	37	10425
csrng	Supports deterministic (DRNG) and true random number generation (TRNG) compliant with FIPS and CC.	69	12	5722
entropy_src	FIPS and CC compliant entropy source used by csrng.	91	20	7750
hmac	SHA-2 hash-based authentication code generator.	80	4	3613
keymgr	The key manager implements the hardware component of the identities and root keys strategy of OpenTitan.	75	14	5257
kmac	Keccak-based message authentication code generator.	202	16	7571
lc_ctrl	Controller to manage product device lifecycle and associated functionality/access control.	101	11	4027
otbn	Co-processor for asymmetric cryptographic operations like Elliptic Curve Cryptography (ECC).	440	24	8279
otp_ctrl	Controller for the One-Time Programmable (OTP) memory.	136	15	8612
prim	Basic blocks used to implement the design; They are often technology-dependent and can have multiple implementations.	501	164	14988
tlul	Main system bus to interface the main processor core with peripherals; implements the TileLink protocol.	87	21	2628

4 Results

The results are summarized in Table 2. For each of the 12 IPs, we evaluate the security properties and issues identified by MARVEL. The security properties are either correctly or incorrectly identified. Overall, 104 out of the 109 security properties are valid. The security issues are also classified as correctly and incorrectly identified. MARVEL correctly identified 20 out of 48 reported issues, giving a success rate of 41.7%. Correctly classified issues are further classified as correctly and incorrectly localized. Of the 20 correctly identified security issues, 11 are correctly localized to their line numbers, five are incorrectly localized, and four are not localized. The runtimes are reasonable, with the longest run taking 23 minutes. The runtime depends on the tool calls, with simulation and assertion verification being the two most time-consuming. The cost of one run spans between \$1.2 and \$2. The kind of analysis carried out by MARVEL would require several hours for an experienced engineer. This highlights the potential of LLMs to speed up hardware security evaluations.

Table 2: Results summary for the 12 Design IPs in buggy OpenTitan earlgrey SoC. - indicates that the number is not applicable, e.g., no bugs were reported by MARVEL for keymgr and lc_ctrl IPs.

Design IP	Runtime [min.]	Security Properties Identified		Security Issues Identified		Security Issues Localized (from correctly identified issues)		
		Correct	Incorrect	Correct	Incorrect	Correct	Incorrect	Not localized
adc_ctrl	8.8	8	0	1	4	1	0	0
aes	18.3	8	0	4	3	2	2	0
csrng	20.4	8	0	2	2	0	1	1
entropy_src	23.0	12	1	1	4	1	0	0
hmac	8.3	12	0	5	1	3	0	2
keymgr	19.1	11	2	-	-	-	-	-
kmac	16.7	11	2	1	4	-	-	1
lc_ctrl	8.9	7	0	-	-	-	-	-
otbn	7.1	7	0	4	2	4	0	0
otp_ctrl	13.7	8	0	1	2	0	1	0
prim	5.7	6	0	1	3	0	1	0
tlul	7.5	6	0	0	3	-	-	-
overall	157.5	104	5	20	28	11	5	4

The initial prompt contains the path to the SoC base directory. From the example sequences of action in Section A.5 we see that the supervisor starts by exploring the SoC file structure and reading the documentation files to identify the security properties. Then it starts calling the available tools to check the identified security properties. The agent might inspect design files based on the tools' feedback to confirm and localize the bugs. Figure 5 shows the normalized and absolute number of actions performed by the supervisor agent. In every run, the supervisor agent calls each tool at least

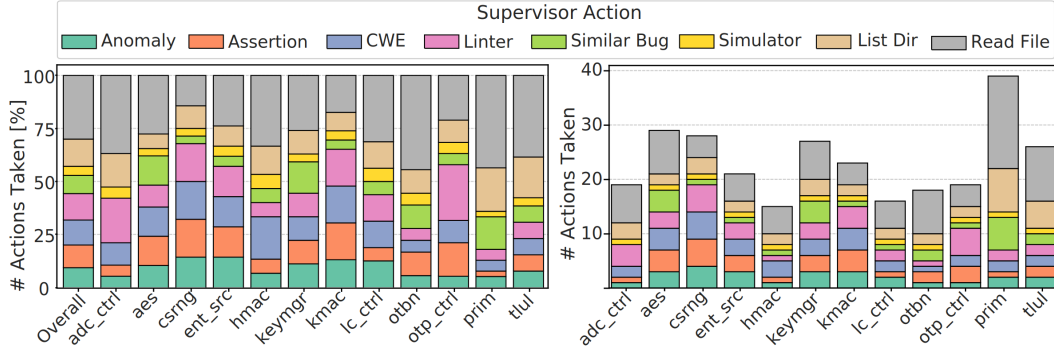


Figure 5: Normalized and Absolute Supervisor Action Distribution, Overall and for single IPs

once. Prim has the lowest runtime and the highest number of actions. This is due to the high number of file reads and directory listings, which makes sense as prim contains multiple basic blocks and has the highest number of files in the SoC.

The roles of each agent in bugs reported by MARVEL are illustrated in Figure 6. Here we focus only on the actions that contributed to a result in the final report. Agents might not find any issues, in this case, the action will not contribute to the final report. If the agent is used to determine a valid security issue, it is described as the Determinator. If it localizes the problem in addition to deciding the issue, it is described as the Determinator and Localizer. If it is used to identify the bug but is not the final determinator, it is described as the Helper. If it is used in the flow of incorrectly identifying a security issue, it is defined as a False Identifier. Note that more than one agent might count as Helper or False Identifier. Sometimes, the supervisor inspects a file and takes a decision in support or independently of an executor agent therefore it is also listed. An agent may also not be used at the supervisor’s discretion. The linter agent was used 11 times, once as the Determinator and Localizer, once as a Helper, and 9 times as the False Identifier. CWE agent was the most used agent: 2 times as the Determinator, once as Determinator and Localizer, 7 times as a Helper, and 14 times as the False Identifier. The assertion and similar bug agents were used half the time for correct identification and half the time as false identifiers. The simulator and anomaly agents were the least used agents, used 3 times each for bug determination. Every agent was used at least once to identify a security issue, and 6 of the 7 agents (all but the anomaly agent) were used in the final determination of a security issue. This supports the use of each agent in MARVEL flow. Section A.2 and Section A.3 report the list of reported issues and respective agent roles.

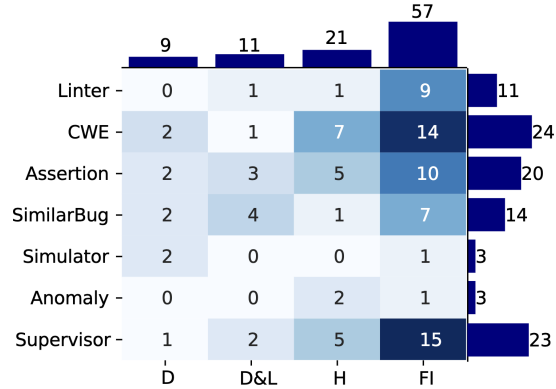


Figure 6: Roles of agents in bugs reported by MARVEL. D=Determinator, L=Localizer, H=Helper, FI=False Id.

5 Discussion

Strengths The benefits of the *Supervisor-Executor* architecture of MARVEL have been demonstrated through the interaction and coordination between agents. Based on the Design IP and corresponding documentation, MARVEL was able to derive security objectives relevant to the IP accurately. The supervisor agent is then able to call on the executor agents according to the file type and security objective as shown in Figure 7. The highest frequency is for the tuple access control and interface files (which implement most of the access control logic in the SoC), followed by FSM security for FSM and control logic files. This shows that the supervisor agent can correctly identify the appropriate security objectives at least for some file categories. The least used security objective is entropy, which is suitable only for very few modules. Section A.4 explains how we assigned security objectives and files to the respective classes. If one agent fails to provide useful information or fails to run correctly, the supervisor executes another agent until enough information is obtained to make a judgment about the violation of the security objective under consideration. The detailed logs of the agentic flow revealed that MARVEL was able to iteratively improve calls to a tool until the syntactically correct call was made. Such an example can be found in Section A.5, the multiple calls in a row to the assertion agent on the same file are due to the tool call failing, and the supervisor agent trying again, fixing format and

assertions. This highlights the benefit of our agentic approach, which allows the supervisor to correct its actions. A single-shot approach without the ability to iterate would result in unrecoverable failing tool calls. We have demonstrated the use of hardware description language (HDL) specific tools in the form of VC SpyGlass Lint as the linter, VC Formal as the assertion tool, and Verilator as the simulator. LLMs can automate the scripting of these tools, with the help of templates, showing that multi-agentic systems can be used for hardware code debugging, just like they can be used for software debugging.

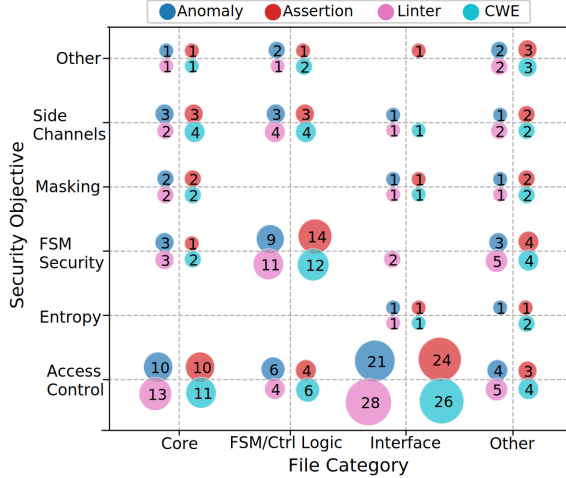


Figure 7: Agent activity frequency for each security objective and file category tuple. We only show agents requiring a specific security objective and file category.

is constantly calling the right agent is also challenging. To make this determination, we would need to compare the decision-making flow of expert RTL verification engineers to MARVEL. We also observe that assertion generation proves difficult for LLMs. While syntactically correct assertions are often produced, meaningful assertions, corresponding to the verified security objective, are much rarer. Some easy-to-detect bugs that individual agents can catch are not caught by MARVEL e.g., in `otbn.sv` no signal is assigned to `mems_sec_wipe_o` port in `otbn_core` instantiation. This could be easily caught if the supervisor agent had run either of the linter or the CWE agent on the `otbn.sv` file.

Ethical Considerations Ethical considerations must be considered when working with cybersecurity topics. For instance, the possibility of malicious use of the tool should be considered. This includes both the use to find vulnerabilities with harmful intent and changes to the system prompts that may allow the objective to be changed from bug detection to bug insertion. In both scenarios, the user would need access to the design files, which in the industry are only accessible by trusted employees. In the hardware domain, these scenarios are less of a concern than in the software domain.

6 Conclusion and Future Work

We developed and evaluated an LLM-based multi-agent framework for identifying security issues in RTL code on the OpenTitan-based Hack@DATE SoC. In our *Supervisor-Executor* flow, an agent can be the final Determinator of a Security issue, Determinator and Localizer, or just a Helper. Our results show that each agent contributes to the security analysis at least as helper. MARVEL reported 48 potential security issues, of which we manually evaluated 20 as valid issues. Of the 20 correctly identified security issues, 11 are correctly localized to their line numbers, five are incorrectly localized, and four are not localized. Our work highlights LLMs’ potential to speed up hardware security evaluations. Yet they augment, not replace, human expertise for critical applications. Experienced professionals must conduct the final assessment and critical decision-making.

Future Work A natural extension of MARVEL would be to add more executor agents. These could include agents using other techniques used for RTL security bug detection, like an Information Flow Tracking Agent and a Fuzzing agent. Another direction for exploring multi-agent systems for RTL security bug detection would be to employ an architecture where executor agents can communicate directly with each other. Alternatively, a multiple hierarchy could have agents between the supervisor and executor agents that are primed to use broad categories of tools. A Static Analysis agent could control flow between the Linter, Assertion, and Anomaly agents, and a Known Bug agent could

Limitations While MARVEL identified 20 security issues correctly, it falsely identified 28. From a security perspective, a precision of 41.7% is good because the cost of a security issue propagating downstream in the silicon design lifecycle increases exponentially. That being said, a significant number of false positives is still a limitation of this work. Most of the false positives are caused by the LLMs conjuring up a security issue when there is none, and some are stylistic issues in the code that do not pose a security concern. Another area for improvement is in localization. While MARVEL identified 20 security issues, it only localized 11 of them accurately. Sometimes the LLM would identify locations related to the parameter/signal definitions or the start of FSMs, but not the actual assignments and conditional statements containing the bug. Determining whether the supervisor agent

include the CWE and Similar Bug agents. Other improvements could include a separate localizer that relies on non-LLM-based techniques to localize the bugs based on information from the Supervisor Agent. These could include embedding and keyword searches.

References

- [1] Baleegh Ahmad, Benjamin Tan, Ramesh Karri, and Hammond Pearce. FLAG: Finding Line Anomalies (in code) with Generative AI, June 2023. URL <http://arxiv.org/abs/2306.12643>. arXiv:2306.12643 [cs].
- [2] Mohammad Akyash and Hadi Mardani Kamali. Self-HWDebug: Automation of LLM Self-Instructing for Hardware Security Verification, May 2024. URL <http://arxiv.org/abs/2405.12347>. arXiv:2405.12347.
- [3] Christopher Brant, Prakash Shrestha, Benjamin Mixon-Baca, Kejun Chen, Said Varlioglu, Nelly Elsayed, Yier Jin, Jedidiah Crandall, and Daniela Oliveira. Challenges and Opportunities for Practical and Effective Dynamic Information Flow Tracking. *ACM Computing Surveys*, 55(1):17:1–17:33, November 2021. ISSN 0360-0300. doi: 10.1145/3483790. URL <https://doi.org/10.1145/3483790>.
- [4] Luca Collini, Andrew Hennessee, Ramesh Karri, and Siddharth Garg. Can reasoning models reason about hardware? an agentic hls perspective, 2025. URL <https://arxiv.org/abs/2503.12721>.
- [5] Ghada Dessouky, David Gens, Patrick Haney, Garrett Persyn, Arun Kanuparthi, Hareesh Khattri, Jason Fung, Ahmad-Reza Sadeghi, and Jeyavijayan Rajendran. Hardfails: Insights into Software-Exploitable Hardware Bugs. In *Proceedings of the 28th USENIX Conference on Security Symposium*, SEC’19, pages 213–230, Santa Clara, CA, USA, 2019. USENIX Association. ISBN 978-1-939133-06-9.
- [6] Zhigang Fang, Renzhi Chen, Zhijie Yang, Yang Guo, Huadong Dai, and Lei Wang. LintLLM: An Open-Source Verilog Linting Framework Based on Large Language Models, February 2025. URL <http://arxiv.org/abs/2502.10815>. arXiv:2502.10815 [cs].
- [7] VC Formal. VC Formal: Formal Verification Solution | Synopsys, 2024. URL <https://www.synopsys.com/verification/static-and-formal-verification/vc-formal.html>.
- [8] Weimin Fu, Kaichen Yang, Raj Gautam Dutta, Xiaolong Guo, and Gang Qu. LLM4SecHW: Leveraging Domain-Specific Large Language Model for Hardware Debugging. In *2023 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, pages 1–6, December 2023. doi: 10.1109/AsianHOST59942.2023.10409307. URL <https://ieeexplore.ieee.org/abstract/document/10409307>.
- [9] Wei Hu, Armaiti Ardeshtiricham, and Ryan Kastner. Hardware Information Flow Tracking. *ACM Computing Surveys*, 54(4):83:1–83:39, May 2021. ISSN 0360-0300. doi: 10.1145/3447867. URL <https://dl.acm.org/doi/10.1145/3447867>.
- [10] Vighnesh Iyer, Donggyu Kim, Borivoje Nikolic, and Sanjit A. Seshia. RTL bug localization through LTL specification mining (WIP). In *Proceedings of the 17th ACM-IEEE International Conference on Formal Methods and Models for System Design*, MEMOCODE ’19, pages 1–5, New York, NY, USA, October 2019. Association for Computing Machinery. ISBN 978-1-4503-6997-8. doi: 10.1145/3359986.3361202. URL <https://doi.org/10.1145/3359986.3361202>.
- [11] LangGraph. Multi-agent supervisor, 2024. URL https://langchain-ai.github.io/langgraph/tutorials/multi_agent/agent_supervisor/.
- [12] Cheryl Lee, Chunqiu Steven Xia, Longji Yang, Jen-tse Huang, Zhouuixin Zhu, Lingming Zhang, and Michael R. Lyu. A Unified Debugging Approach via LLM-Based Multi-Agent Synergy, October 2024. URL <http://arxiv.org/abs/2404.17153>. arXiv:2404.17153 [cs].

- [13] Xinyi Li, Sai Wang, Siqi Zeng, Yu Wu, and Yi Yang. A survey on LLM-based multi-agent systems: workflow, infrastructure, and challenges. *Vicinagearth*, 1(1):9, October 2024. ISSN 3005-060X. doi: 10.1007/s44336-024-00009-2. URL <https://doi.org/10.1007/s44336-024-00009-2>.
- [14] VC SpyGlass Lint. Synopsys VC SpyGlass Lint, 2022. URL <https://www.synopsys.com/verification/static-and-formal-verification/vc-spyglass/vc-spyglass-lint.html>.
- [15] lowRISC contributors. Open source silicon root of trust (RoT) | OpenTitan, 2023. URL <https://opentitan.org/>.
- [16] Sujit Kumar Muduli, Gourav Takhar, and Pramod Subramanyan. Hyperfuzzing for SoC security validation. In *Proceedings of the 39th International Conference on Computer-Aided Design, ICCAD '20*, pages 1–9, New York, NY, USA, November 2020. Association for Computing Machinery. ISBN 978-1-4503-8026-3. doi: 10.1145/3400302.3415709. URL <https://doi.org/10.1145/3400302.3415709>.
- [17] Sayak Ray, Nishant Ghosh, Ramya Masti, Arun Kanuparthi, and Jason Fung. INVITED: Formal Verification of Security Critical Hardware-Firmware Interactions in Commercial SoCs. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pages 1–4, June 2019. ISSN: 0738-100X.
- [18] Cynthia Sturton, Matthew Hicks, Samuel T. King, and Jonathan M. Smith. FinalFilter: Asserting Security Properties of a Processor at Runtime. *IEEE Micro*, 39(4):35–42, July 2019. ISSN 1937-4143. doi: 10.1109/MM.2019.2921509.
- [19] Shams Tarek, Dipayan Saha, Sujan Kumar Saha, Mark Tehranipoor, and Farimah Farahmandi. SoCureLLM: An LLM-driven Approach for Large-Scale System-on-Chip Security Verification and Policy Generation, 2024. URL <https://eprint.iacr.org/2024/983>. Publication info: Preprint.
- [20] Shams Tarek, Dipayan Saha, Sujan Kumar Saha, and Farimah Farahmandi. BugWhisperer: Fine-Tuning LLMs for SoC Hardware Vulnerability Detection, 2025. URL <https://eprint.iacr.org/2025/546>. Publication info: Published elsewhere. IEEE VLSI Test Symposium (VTS) 2025.
- [21] Aakash Tyagi, Addison Crump, Ahmad-Reza Sadeghi, Garrett Persyn, Jeyavijayan Rajendran, Patrick Jauernig, and Rahul Kande. TheHuzz: Instruction Fuzzing of Processors Using Golden-Reference Models for Finding Software-Exploitable Vulnerabilities. *arXiv:2201.09941 [cs]*, January 2022. URL <http://arxiv.org/abs/2201.09941>. arXiv: 2201.09941.
- [22] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6): 186345, March 2024. ISSN 2095-2236. doi: 10.1007/s11704-024-40231-1. URL <https://doi.org/10.1007/s11704-024-40231-1>.
- [23] Ke Xu, Jialin Sun, Yuchen Hu, Xinwei Fang, Weiwei Shan, Xi Wang, and Zhe Jiang. MEIC: Re-thinking RTL Debug Automation using LLMs. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design, ICCAD '24*, pages 1–9, New York, NY, USA, April 2025. Association for Computing Machinery. ISBN 9798400710773. doi: 10.1145/3676536.3676801. URL <https://dl.acm.org/doi/10.1145/3676536.3676801>.

A Technical Appendices and Supplementary Material

A.1 Illustrative Examples for Executor Agents

This section includes examples of the executor agents' detection of security issues not covered in the main pages of this manuscript.

A.1.1 Example for Assertion Agent

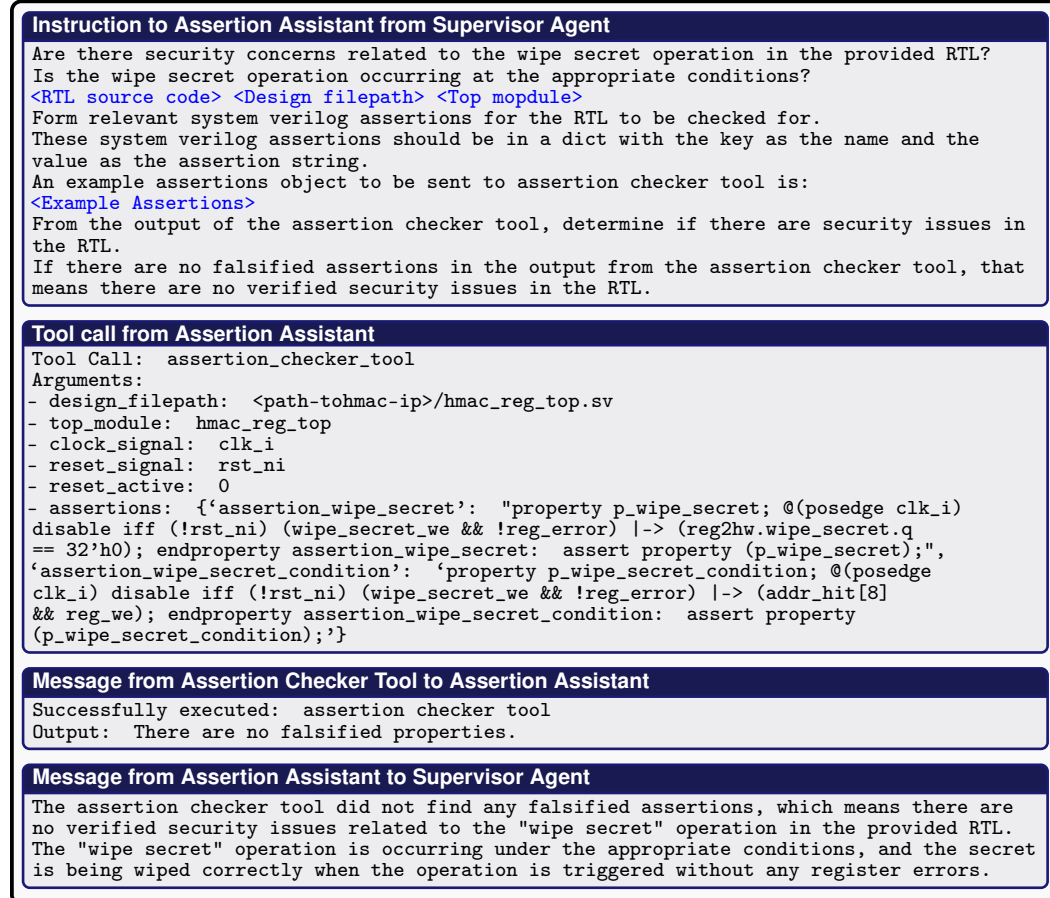


Figure 8: Example Assertion Agent flow for wipe secret security objective for HMAC Register Top module.

A.1.2 Example for CWE Agent

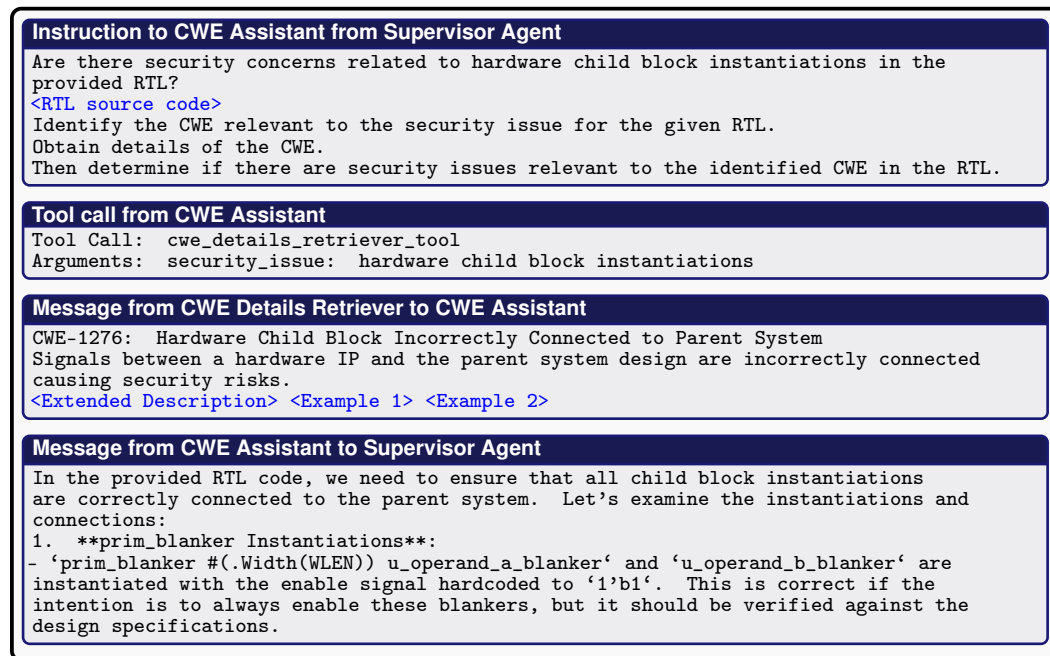
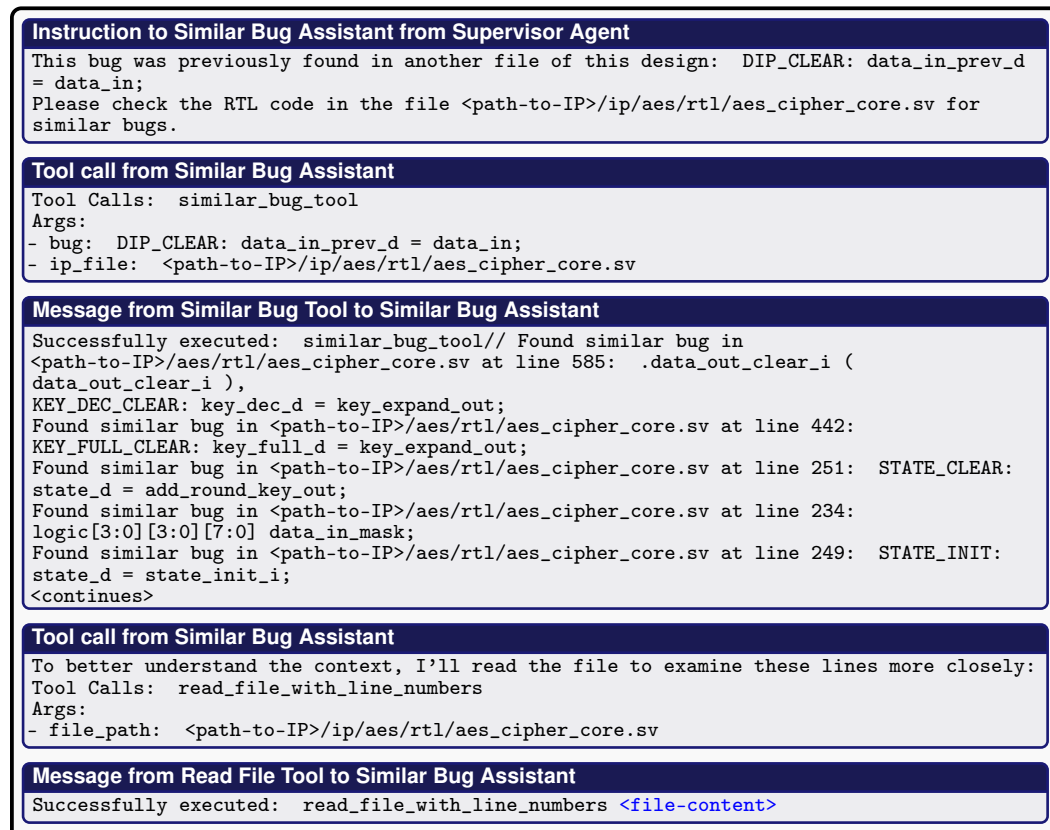


Figure 9: Example CWE Agent flow on OTBN MAC Bignum module

A.1.3 Example for Similar Bug Agent



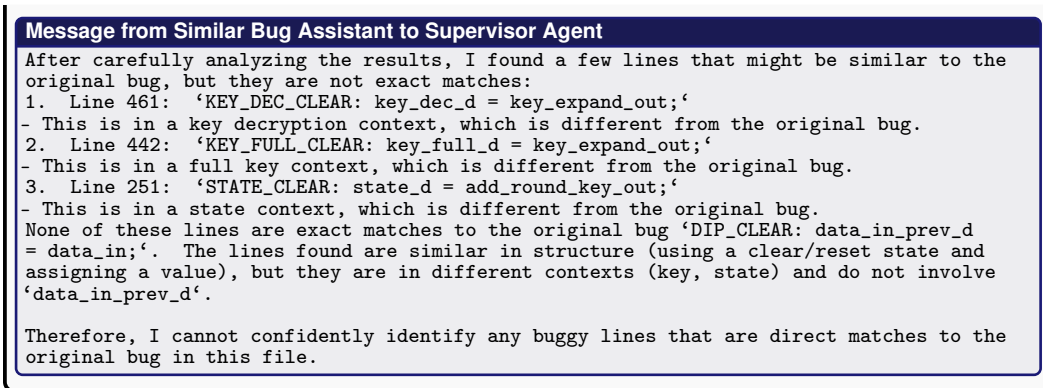


Figure 10: Example Similar Bug Agent flow for a similar bug previously found in aes_core.sv.

A.1.4 Example for Anomaly Agent

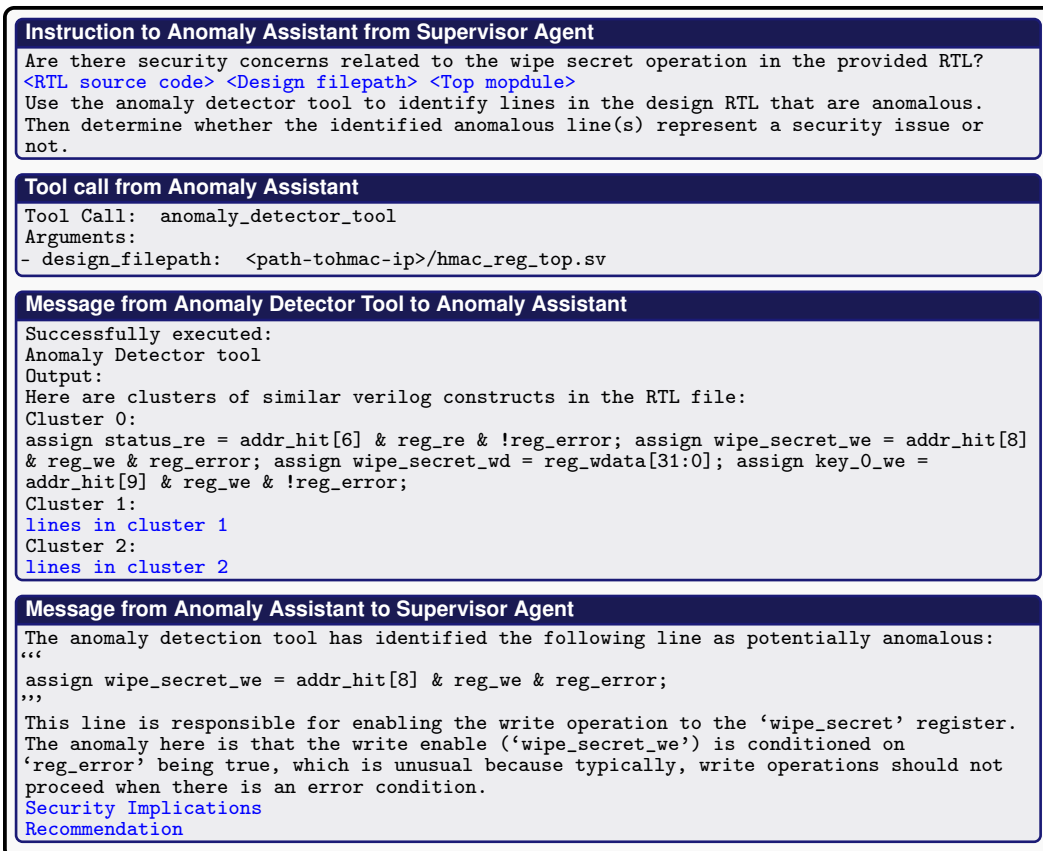


Figure 11: Anomaly Agent flow for wipe secret security objective for HMAC Register Top module.

A.1.5 Example for Simulator Agent

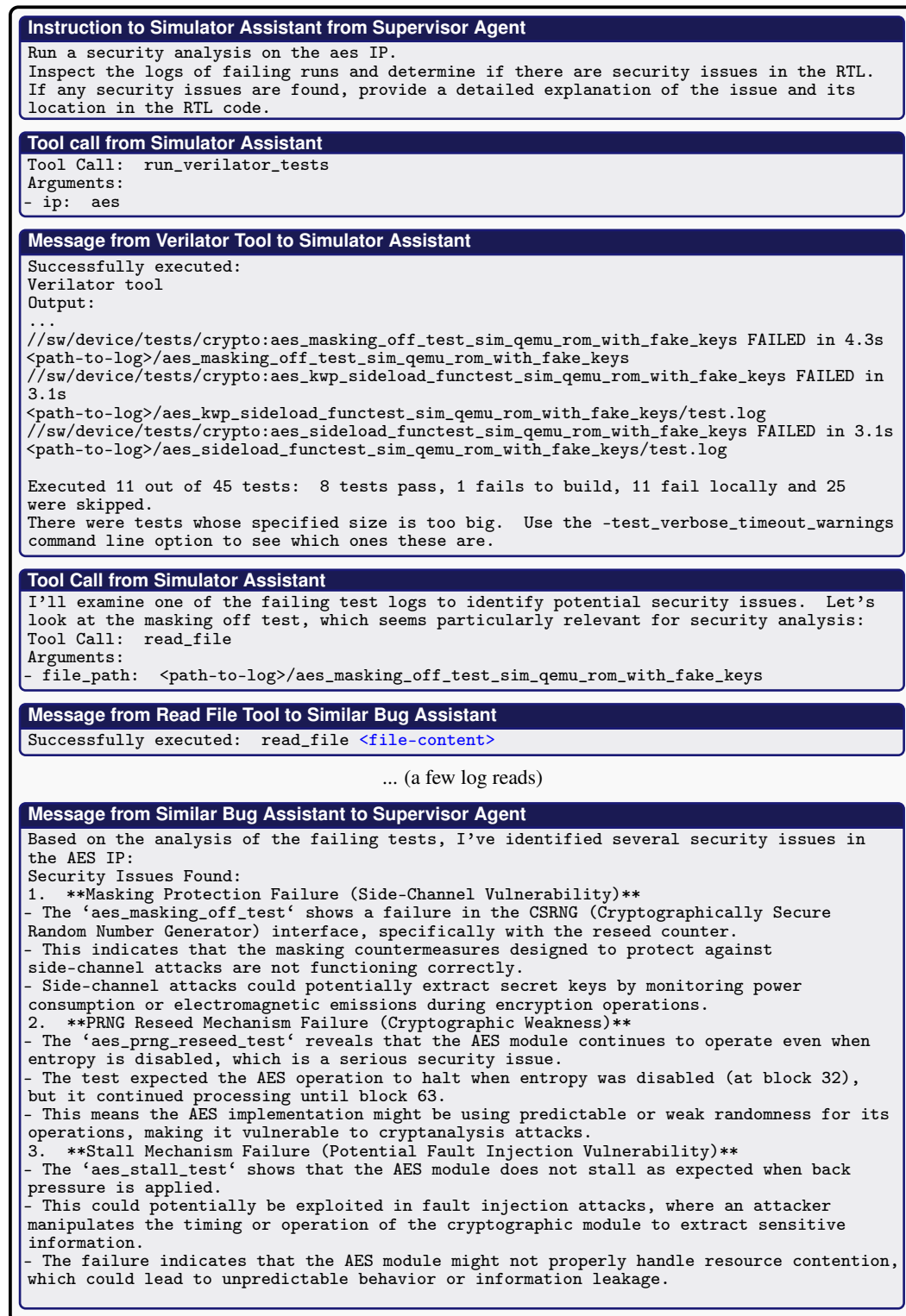


Figure 12: Example Simulator Agent for AES.

A.2 Reported Issues

Table 3: Reported Issues.

Reported Issue ID	IP	Description (As produced by LLM)
1	ADC_CTRL	All critical configuration and control registers (including filter settings, enable controls, power management, and interrupt controls) are directly writable by any entity with bus access. There is no privilege-level check, lock bit, or write-once protection for these registers. This allows potential unauthorized modification of ADC controller behavior at runtime.
2	ADC_CTRL	In the 'LP_SLP' state, when 'wakeup_timer_cnt_q == cfg_wakeup_time_i', the FSM clears the timer but does not assign a next state. This could result in the FSM getting stuck in the 'LP_SLP' state, leading to a potential denial of service or unpredictable behavior.
3	ADC_CTRL	Filter control settings are directly assigned from the register interface without any privilege or lock checks. An attacker with register access could manipulate filter thresholds and enable bits, potentially bypassing security logic or causing false positives/negatives in ADC event detection.
4	ADC_CTRL	The FSM waits indefinitely for external signals (e.g., 'adc_d_val_i') without a timeout. If the expected signal never arrives, the FSM could hang, leading to a denial of service.
5	ADC_CTRL	Registers controlling ADC operation (e.g., filter settings, power management, interrupt enables) lack lock bits or write-once protection. This allows runtime modification of security-critical settings, which could be exploited to bypass filtering, trigger false wakeups, or suppress interrupts.
6	AES	Improper REGWEN Protection*: Assertion falsified indicating 'ctrl_aux_shadowed' can be written even when 'ctrl_aux_regwen_qs' is not set. This allows bypassing the intended lock on critical configuration.
7	AES	*Data Output During Alert*: Assertion failed—data output may still be enabled during alert conditions, risking data leakage.
8	AES	Key clearing mux
9	AES	*Key Share Separation Issue*: Assertion failed—two key shares are not always different when masking is enabled, weakening side-channel resistance.
10	AES	*PRNG Error Ignored*: The PRNG error signal is ignored, so critical failures (e.g., all-zero state) may go undetected.
11	AES	*Masking Bypass Risk*: If 'SecAllowForcingMasks' is enabled, masks can be forced to constants, disabling masking and exposing the core to side-channel attacks.
12	AES	*PRNG Reseed Bypass*: If 'SecSkipPRNGReseeding' is enabled, PRNG reseeding is skipped, leading to predictable masks and reduced security.
13	CSRNG	FSM can hang indefinitely in waiting states if external signals never arrive. No timeout or recovery mechanism.
14	CSRNG	Internal cryptographic state can be read even when the module is disabled or when read enable is not asserted.
15	CSRNG	Unregistered inputs/outputs and complex struct types may cause reliability/timing issues, which could indirectly impact security.
16	CSRNG	Test failure indicates possible bug in reseed counter update, which could weaken cryptographic strength.
17	ENTROPY_SRC	REGWEN/ME_REGWEN bypass, privilege escalation

18	ENTROPY_SRC	FIFO overflow/underflow, bit-width/overflow warnings
19	ENTROPY_SRC	Data path integrity, bus compare alert
20	ENTROPY_SRC	Firmware override control, privilege escalation
21	ENTROPY_SRC	Bit-width/overflow warnings in counters/FIFOs
22	HMAC	Multi-stream digest computation bug
23	HMAC	Integrity error steering failure
24	HMAC	Key register readback (info leak)
25	HMAC	No privilege-based access control
26	HMAC	No debug mode key clearing
27	HMAC	FSM/non-FSM logic mixing
28	KMAC	Shadowed register errors not always propagated
29	KMAC	SW can inject commands/messages during HW app, invalid key use
30	KMAC	FIFO masking allows zero mask, flush accepts new messages
31	KMAC	Hard-coded key length constants (maintainability)
32	KMAC	Clock gating failures (test-level)
33	OTBN	Secure wipe request hardwired to 0 (critical bug)
34	OTBN	FSM transitions to locked state, but secure wipe is disabled
35	OTBN	Side-channel blanking bypassed
36	OTBN	ISPR write commit always enabled
37	OTBN	Register interface security, integrity checks, access gating
38	OTBN	Secure wipe FSM logic, but undermined by controller bug
39	OTP_CTRL	Possible race/timing issue: partition may not always lock in error state (assertion failure)
40	OTP_CTRL	Struct ports, unregistered IOs, unused inputs; could cause mismatches affecting security
41	OTP_CTRL	Dead code, unique sampling issue on error_o; could affect error reporting
42	PRIM	The module disables strict checking of the one-hot write-enable vector by setting ‘.StrictCheck(0)’
43	PRIM	The ‘prim_onehot_check’ module allows the one-hot vector to be all zeros when ‘en_i’ is asserted if ‘StrictCheck’ is set to 0
44	PRIM	Address checking is disabled (‘.AddrCheck(0)’ for scalability reasons
45	PRIM	The module generates an error signal (‘err_o’) but relies on the instantiating module to properly handle this signal and trigger alerts
46	TLUL	if the register map or downstream logic expects sub-word accesses or different alignment, this could cause silent data corruption or access control bypass.
47	TLUL	Potential for integrity bypass if integrities are not enabled in higher-level design.
48	TLUL	Instruction fetches are only allowed if ‘en_ifetch_i’ is enabled. This is a positive security feature, but if ‘en_ifetch_i’ is incorrectly set, unauthorized instruction fetches could occur.

A.3 Roles of Agents in Reported Issues

For each reported issue, an agent may be involved in some capacity. If the agent was used in determining a security issue as valid, it is tagged as Determinator (D). If it localizes the issue in addition to determination of the issue, it is tagged as (D,L). If it is used in the flow of identifying the bug but was not the final determinator, it is tagged as Helper (H). If it is used in the flow of incorrectly identifying a security issue, it is tagged as False identifier (F). If an agent is not used at all, it is not tagged (-). The roles played by agents for each reported issue are shown in Table 4.

Table 4: Roles of Agents in Reported Issues.

Reported Issue ID	Valid?	Localized?	CWE	Similar	Assertion	Lint	Anomaly	Simulator	Supervisor
1	✗	-	F	F	-	-	-	-	F
2	✓	✓	H	D,L	-	-	-	-	H
3	✗	-	F	F	-	-	-	-	F
4	✗	-	F	F	-	-	-	-	F
5	✗	-	F	-	-	-	-	-	F
6	✗	-	F	-	F	F	-	-	-
7	✓	✓	H	D,L	H	-	-	-	-
8	✓	✗	H	D	H	-	-	-	-
9	✓	✗	H	D	H	-	-	-	-
10	✓	✓	D	-	H	-	-	-	H
11	✗	-	F	-	F	-	-	-	F
12	✗	-	F	-	F	-	-	-	F
13	✗	-	F	F	F	-	-	-	-
14	✓	✗	-	-	D	-	-	-	-
15	✗	-	-	-	-	F	-	-	-
16	✓	✗	-	-	-	-	-	D	-
17	✗	-	F	-	F	-	-	-	-
18	✓	✓	-	D,L	-	H	H	-	-
19	✗	-	-	-	-	-	F	-	F
20	✗	-	-	-	F	-	-	-	F
21	✗	-	-	-	-	F	-	-	-
22	✗	-	-	-	-	-	-	F	F
23	✓	✓	-	-	D,L	-	-	-	-
24	✓	✓	H	D,L	-	-	-	-	-
25	✓	-	D	-	-	-	-	-	-
26	✓	-	D	-	-	-	-	-	-
27	✓	✓	-	-	-	D,L	-	-	-
28	✗	-	F	-	F	-	-	-	-
29	✗	-	F	-	F	F	-	-	-
30	✗	-	-	-	F	F	-	-	-
31	✗	-	-	-	-	F	-	-	-
32	✓	✗	-	-	-	-	-	D	-
33	✓	✓	H	-	H	-	H	-	D,L
34	✓	✓	-	-	D,L	-	-	-	H
35	✓	✓	-	-	D,L	-	-	-	H
36	✓	✓	-	-	-	-	-	-	D,L
37	✗	-	-	-	F	F	-	-	-
38	✗	-	-	F	-	-	-	-	-
39	✓	✗	-	-	D	-	-	-	H
40	✗	-	-	-	-	F	-	-	-
41	✗	-	-	-	-	F	-	-	-
42	✗	-	F	F	-	-	-	-	F
43	✓	✗	H	H	-	-	-	-	D
44	✗	-	F	-	-	-	-	-	F
45	✗	-	F	-	-	-	-	-	F
46	✗	-	-	F	-	-	-	-	F
47	✗	-	-	-	-	-	-	-	F
48	✗	-	-	-	-	-	-	-	F

A.4 Security Objectives and File Category Classes

This appendix describes the classification of security objectives and design files undertaken to investigate the supervisor agent’s operational patterns across different runs. Our aim was to determine

if recurring tuples of agents and security objectives were present and if their selection followed a logical basis or stochastic distribution. The outcome of this investigation is shown in Figure 7 and analyzed in Section 5. The remainder of this section reports on the assignment methodology for each design file and security objective to their corresponding classes.

A.4.1 Design File Classification

OpenTitan uses standardized naming schemes for design files where the first part of the file name is the IP name, followed by the file type (e.g., `control`, `core`, `reg_top`). We classified files based on their postfix:

- **Interface:** `reg_top`, `core_reg_top`, `reg_we_check`, `adapter_reg`, `adapter_sram`, `lci`, `dai`, `kmac_if`.
- **Core:** `app`, `top`, `core`.
- **FSM/Control Logic:** `ctrl`, `controller`, `fsm`, `onehot_check`, `sm`, `main_sm`, `cipher_control`.
- **Other:** `part_buf`, `part_unbuf`, `intr`, `state_db`, `cmd_stage`, `msgfifo`, `prng_masking`, `ctr_drbg_cmd`.

A.4.2 Security Objective Classification

We collected all security objectives used by the supervisor agent and manually classified them:

- **FSM Security:** state machine security, FSM security, cryptographic core security, FSM, deadlock, FSM deadlock, improper state transitions, CWE mapping for state management, cryptographic state access control, error handling, state management, cryptographic state update, context switching, state machine, context switching anomalies, illegal state transitions, FSM stuck/faulty, FSM hardening, state transition, stuck states, error state handling, terminal error state enforcement, sparse encoding
- **Access Control:** register access, privilege escalation, reserved bits, interrupt security, register access security, register interface security, register access policy enforcement, register access control, improper access, key management, register interface, access anomalies, register interface access control, privilege enforcement, key register confidentiality, shadow register, access policy, interface security, register access policy, shadow register integrity, shadow register anomalies, shadowed register integrity, key handling, message integrity, privilege separation, integrity, illegal access prevention, memory integrity, access control, direct access interface security, partition locking, partition integrity, partition access control, life cycle interface access control, privilege escalation prevention, integrity error handling, partition lock enforcement, protocol compliance, memory access control, memory interface security, entropy integrity, register lock, data path integrity
- **Entropy:** entropy, entropy tracking, entropy usage, data leakage
- **Masking:** masking, masking bypass, masking enforcement, FIFO masking
- **Side Channels:** side-channel resistance, fault injection resistance, fault injection, state isolation, stuck-at faults, application interface isolation, clock bypass, volatile unlock, mutex, hardware state machine glitch attack, denial of service, command sequencing, command injection
- **Other:** PRNG reseed, reseed counter, reseed counter management, secure wipe, ECC, error propagation, ECC error propagation, token validation, FIFO overflow/underflow

A.5 Sequence of Actions Reports

In this appendix, we provide two reports containing the sequence of actions for ADC ctrl and AES. Note that these logs also contain actions from the executor agents. Figure 5 only considers the supervisor agent actions. All logs, containing the full outputs of the LLMs are available in our anonymized repository.

Sequence of Actions for ADC ctrl

```
Listing directory: <path-to-IP>/adc_ctrl
Listing directory: <path-to-IP>/adc_ctrl/doc
Reading file: <path-to-IP>/adc_ctrl/doc/theory_of_operation.md
Reading file: <path-to-IP>/adc_ctrl/doc/registers.md
Listing directory: <path-to-IP>/adc_ctrl/rtl
Reading file with line numbers: <path-to-IP>/adc_ctrl/rtl/adc_ctrl.sv
Reading file with line numbers: <path-to-IP>/adc_ctrl/rtl/adc_ctrl_core.sv
Reading file with line numbers: <path-to-IP>/adc_ctrl/rtl/adc_ctrl_fsm.sv
Running verilator agent on adc_ctrl
Running verilator tests for adc_ctrl
Running anomaly detector agent on <path-to-IP>/adc_ctrl/rtl/adc_ctrl_reg_top.sv for
adc_ctrl_reg_top with security objective: register access, privilege escalation, reserved
bits
Running llm cwe checker agent on <path-to-IP>/adc_ctrl/rtl/adc_ctrl_fsm.sv for adc_ctrl_fsm
with security objective: state machine security
Running llm cwe details retriever tool on state machine security vulnerabilities
Running assertions checker agent on <path-to-IP>/adc_ctrl/rtl/adc_ctrl_fsm.sv for
adc_ctrl_fsm with security objective: state machine security
Running linter agent on <path-to-IP>/adc_ctrl/rtl/adc_ctrl_fsm.sv for adc_ctrl_fsm with
security objective: FSM security
Running lint checker tool on <path-to-IP>/adc_ctrl/rtl/adc_ctrl_fsm.sv for adc_ctrl_fsm
with lint tags: ['FsmLiveLock', 'NoExitFsmState', 'UnreachableStatesInFSM-ML',
'LINT_FSM_UNREACHABLE_STATES', 'LINT_FSM_WITHOUT_EXIT_STATE']
Reading file with line numbers: <path-to-IP>/adc_ctrl/rtl/adc_ctrl_intr.sv
Running linter agent on <path-to-IP>/adc_ctrl/rtl/adc_ctrl_intr.sv for adc_ctrl_intr with
security objective: interrupt security
Running lint checker tool on <path-to-IP>/adc_ctrl/rtl/adc_ctrl_intr.sv for adc_ctrl_intr
with lint tags: ['ClockDomain', 'STARC05-3.3.6.2', 'ClockEnableRace']
Running linter agent on <path-to-IP>/adc_ctrl/rtl/adc_ctrl_core.sv for adc_ctrl_core with
security objective: register access security
Running lint checker tool on <path-to-IP>/adc_ctrl/rtl/adc_ctrl_core.sv for adc_ctrl_core
with lint tags: ['Rule_2', 'RegisterStuckInResetState-ML', 'W450L', 'UseLogic-ML']
Running llm cwe checker agent on <path-to-IP>/adc_ctrl/rtl/adc_ctrl_core.sv for
adc_ctrl_core with security objective: register access security
Running llm cwe details retriever tool on register access control hardware
Running llm cwe details retriever tool on hardware register protection
Running llm cwe details retriever tool on hardware register access control
Reading file with line numbers: <path-to-IP>/adc_ctrl/rtl/adc_ctrl_reg_top.sv
Running linter agent on <path-to-IP>/adc_ctrl/rtl/adc_ctrl_reg_top.sv for adc_ctrl_reg_top
with security objective: register access security
Running lint checker tool on <path-to-IP>/adc_ctrl/rtl/adc_ctrl_reg_top.sv for
adc_ctrl_reg_top with lint tags: ['STARC05-1.4.3.1b', 'W188', 'W450L', 'ClockEnableRace',
'STARC05-3.3.1.4a']
```

Sequence of Actions for AES

```
Listing directory: <path-to-ip>/aes/doc
Reading file: <path-to-ip>/aes/doc/theory_of_operation.md
Reading file: <path-to-ip>/aes/doc/registers.md
Listing directory: <path-to-ip>/aes/rtl
Running verilator agent on aes
Running verilator tests for aes
Reading file with line numbers: <path-to-ip>/aes/rtl/aes_reg_top.sv
Running linter agent on <path-to-ip>/aes/rtl/aes_reg_top.sv for aes_reg_top with security
objective: register interface security
Running lint checker tool on <path-to-ip>/aes/rtl/aes_reg_top.sv for aes_reg_top with lint
tags: ['RegInputOutput-ML', 'RegInput-ML', 'RegisterInfo-ML', 'InterfaceNameConflicts-ML',
'W256', 'Rule_41', 'W240', 'W500', 'STARC05-1.4.3.1b', 'W401', 'W215', 'STARC05-1.2.1.1a',
'STARC05-2.1.9.5', 'InterfaceWithoutModport-ML', 'DEBUG_LINT_PORT_STRUCT_SIGNAL', 'W188',
'CheckPortType-ML', 'PortOrder_C', 'AutomaticFuncTask-ML', 'STARC05-1.4.1.1']
Running assertions checker agent on <path-to-ip>/aes/rtl/aes_reg_top.sv for aes_reg_top
with security objective: register access policy enforcement
Running assertion checker tool on <path-to-ip>/aes/rtl/aes_reg_top.sv for aes_reg_top with
assertions: <assertions>
Running llm cwe checker agent on <path-to-ip>/aes/rtl/aes_reg_top.sv for aes_reg_top with
security objective: register access control, privilege escalation, improper access
Running llm cwe details retriever tool on register access control
Running llm cwe details retriever tool on privilege escalation hardware
Running llm cwe details retriever tool on improper access control hardware
Running linter agent on <path-to-ip>/aes/rtl/aes_core.sv for aes_core with security
objective: cryptographic core security, masking, key management
```

Running lint checker tool on <path-to-ip>/aes/rtl/aes_core.sv for aes_core with lint tags: ['STARCO5-1.4.3.1b', 'STARCO5-1.2.1.1a', 'STARCO5-1.3.2.1a', 'STARCO5-1.4.1.1', 'STARCO5-3.3.1.4a', 'STARCO5-1.4.3.1a', 'STARCO5-1.4.3.1a', 'STARCO5-1.1.2.2', 'STARCO5-2.1.4.5', 'STARCO5-1.2.1.1b', 'STARCO5-1.4.3.6']

Running assertions checker agent on <path-to-ip>/aes/rtl/aes_core.sv for aes_core with security objective: masking, key management, side-channel resistance

Running assertion checker tool on <path-to-ip>/aes/rtl/aes_core.sv for aes_core with assertions: <assertions>

Running assertion checker tool on <path-to-ip>/aes/rtl/aes_core.sv for aes_core with assertions: <assertions>

Running llm cwe checker agent on <path-to-ip>/aes/rtl/aes_core.sv for aes_core with security objective: side-channel resistance, key management, masking

Running llm cwe details retriever tool on side-channel attacks in cryptographic implementations

Running llm cwe details retriever tool on cryptographic key management vulnerabilities

Running llm cwe details retriever tool on masking in cryptographic implementations

Running linter agent on <path-to-ip>/aes/rtl/aes_cipher_control.sv for aes_cipher_control with security objective: FSM security, sparse encoding, fault injection resistance

Running lint checker tool on <path-to-ip>/aes/rtl/aes_cipher_control.sv for aes_cipher_control with lint tags: ['FsmUsage-ML', 'STARCO5-2.11.1.2', 'STARCO5-2.11.1.4', 'UseDefine', 'STARCO5-2.4.1.4', 'STARCO5-2.3.5.1', 'STARCO5-1.3.1.2', 'SepFSMLogic', 'STARCO5-1.3.2.1a', 'MergeFlops-ML', 'STARCO5-1.2.1.1a', 'STARCO5-1.4.3.1b', 'STARCO5-2.3.1.7a', 'STARCO5-1.2.1.3', 'STARCO5-1.6.1.2', 'SafeLatch-ML', 'STARCO5-1.4.3.6', 'W215', 'NoExitInFSM-ML', 'UseSVAAlways-ML']

Running assertions checker agent on <path-to-ip>/aes/rtl/aes_cipher_control.sv for aes_cipher_control with security objective: FSM security, sparse encoding, fault injection resistance

Running assertion checker tool on <path-to-ip>/aes/rtl/aes_cipher_control.sv for aes_cipher_control with assertions: <assertions>

Running assertion checker tool on <path-to-ip>/aes/rtl/aes_cipher_control.sv for aes_cipher_control with assertions: <assertions>

Running assertion checker tool on <path-to-ip>/aes/rtl/aes_cipher_control.sv for aes_cipher_control with assertions: <assertions>

Running llm cwe checker agent on <path-to-ip>/aes/rtl/aes_cipher_control.sv for aes_cipher_control with security objective: FSM, sparse encoding, fault injection

Running llm cwe details retriever tool on FSM state encoding vulnerabilities

Running llm cwe details retriever tool on Sparse encoding vulnerabilities

Running llm cwe details retriever tool on Fault injection vulnerabilities in hardware

Running anomaly detector agent on <path-to-ip>/aes/rtl/aes_reg_top.sv for aes_reg_top with security objective: register interface and access anomalies

Running anomaly detector agent on <path-to-ip>/aes/rtl/aes_core.sv for aes_core with security objective: masking, key management, side-channel resistance

Running anomaly detector agent on <path-to-ip>/aes/rtl/aes_cipher_control.sv for aes_cipher_control with security objective: FSM, sparse encoding, fault injection

Reading file with line numbers: <path-to-ip>/aes/rtl/aes_prng_masking.sv

Running assertions checker agent on <path-to-ip>/aes/rtl/aes_prng_masking.sv for aes_prng_masking with security objective: entropy, PRNG reseed, masking bypass

Running assertion checker tool on <path-to-ip>/aes/rtl/aes_prng_masking.sv for aes_prng_masking with assertions: <assertions>

Running llm cwe checker agent on <path-to-ip>/aes/rtl/aes_prng_masking.sv for aes_prng_masking with security objective: entropy, PRNG reseed, masking bypass

Running llm cwe details retriever tool on cryptographic random number generation weakness

Running llm cwe details retriever tool on cryptographic masking bypass

Running llm cwe details retriever tool on cryptographic entropy weakness

Running similar bug agent on <path-to-ip>/aes/rtl/aes_core.sv for bug: The assertion checking that cipher_key_clear leads to key_init_sel being set to KEY_INIT_CLEAR failed.

Running similar bug tool on <path-to-ip>/aes/rtl/aes_core.sv for bug: cipher_key_clear leads to key_init_sel being set to KEY_INIT_CLEAR

Reading file with line numbers: <path-to-ip>/aes/rtl/aes_core.sv

Running similar bug tool on <path-to-ip>/aes/rtl/aes_core.sv for bug: cipher_key_clear leads to key_init_sel being set to KEY_INIT_CLEAR

Running similar bug agent on <path-to-ip>/aes/rtl/aes_core.sv for bug: The assertion verifying that the two key shares are different when masking is enabled failed.

Reading file with line numbers: <path-to-ip>/aes/rtl/aes_core.sv

Running similar bug tool on <path-to-ip>/aes/rtl/aes_core.sv for bug: key_init_cipher = key_init_q;

Running similar bug tool on <path-to-ip>/aes/rtl/aes_core.sv for bug: assign key_init_cipher = key_init_q;

Running similar bug agent on <path-to-ip>/aes/rtl/aes_core.sv for bug: The assertion checking that the state shares are different when masking is enabled failed.

Reading file with line numbers: <path-to-ip>/aes/rtl/aes_core.sv

Running similar bug tool on <path-to-ip>/aes/rtl/aes_core.sv for bug: state_init[0] = (state_in add_state_in) \$state_mask; // Masked data share

assign state_init[1] = state_mask; // Mask share

Running similar bug tool on <path-to-ip>/aes/rtl/aes_core.sv for bug: state_init[0] = (state_in add_state_in) \$state_mask;

Running similar bug agent on <path-to-ip>/aes/rtl/aes_core.sv for bug: The assertion verifying that data output is disabled during an alert condition failed.

Reading file with line numbers: <path-to-ip>/aes/rtl/aes_core.sv

Running similar bug tool on <path-to-ip>/aes/rtl/aes_core.sv for bug: data output is disabled during an alert condition