# EQTY Lab Cupcake

## Security Assessment (Summary Report)

**September 30, 2025**

*Prepared for:*
**EQTY Lab**

*Prepared by:* **Michael Brown, Boyan Milanov, and Will Vandevanter**

# Table of Contents

# Project Summary

## Contact Information

The following project manager was associated with this project:

**Mary O'Brien**, Project Manager
mary.obrien@trailofbits.com

The following engineering director was associated with this project:

**Keith Hoodlet**, Engineering Director, AI/ML and Application Security
keith.hoodlet@trailofbits.com

The following consultants were associated with this project:

**Michael Brown**, Consultant
michael.brown@trailofbits.com

**Boyan Milanov**, Consultant
boyan.milanov@trailofbits.com

**Will Vandevanter**, Consultant
will.vandevanter@trailofbits.com

## Project Timeline

The significant events and milestones of the project are listed below.

| Date | Event |
|---|---|
| **September 3, 2025** | Pre-project kickoff call |
| **September 19, 2025** | Delivery of report draft |
| **September 19, 2025** | Report readout meeting |
| **September 30, 2025** | Delivery of final summary report |

# Project Targets

The engagement involved reviewing and testing the following target.

**cupcake**

| | |
|---|---|
| Repository | https://github.com/eqtylab/cupcake/tree/v0.1.0 |
| Version | `0652015b613903bb75ec283bad5f1c566d85e081` |
| Type | Rust |
| Platform | macOS (Apple Silicon) |

# Executive Summary

## Engagement Overview

EQTY Lab engaged Trail of Bits to review the security of Cupcake (`0652015b613903bb75ec283bad5f1c566d85e081`).

Cupcake is a policy engine for AI coding agents. It works by intercepting tool calls from AI coding agents and evaluating them against user-defined policies written in Open Policy Agent (OPA) Rego, returning Allow, Block, or Warn decisions. The system integrates with Claude Code through a hooks mechanism that captures actions like shell commands or file edits before execution. It compiles policies to WebAssembly (Wasm) for fast evaluation in a sandboxed environment. Cupcake stores its configuration and trust data in a `.cupcake` directory and uses signals to gather contextual information such as Git branch status or file contents during policy evaluation. Users write policies that can block specific commands, protect directories, enforce workflow requirements, or inject behavioral guidance prompts back to the agent.

A team of three consultants conducted the review from September 3 to September 18, 2025, for a total of four engineer-weeks of effort. With full access to source code and documentation, we performed static and dynamic testing of Cupcake, using automated and manual processes.

## Observations and Impact

This audit focused on the initial release of the Cupcake tool (v0.1.0) as of September 3, 2025 (`0652015b613903bb75ec283bad5f1c566d85e081`). Our analysis focused on a few key areas and specifically on answering the following questions:

- Given access to a deployed instance of Cupcake with Claude Code, what level of effort is required to bypass policies and otherwise circumvent restrictions?
    - Our threat model was largely focused on access from the Claude Code client only. We believe this perspective applies well to other usages of Cupcake such as in a MCP server or in a deployed chat agent. We also felt this gave us the strongest opportunity to review Cupcake dynamically.
- Does the documentation meet the claims of the tool, and what guidance can we provide with regard to ideal market fit?
- What vulnerabilities, potential bypasses, and violations of security best practices can be identified through source code review?

Overall, the Cupcake system reduces the likelihood of an unintentionally misaligned agent executing unsafe actions and could be applied as an early alert system on adversarial behavior. Additionally, Cupcake has readable plain language policies, operates quickly due

to Rust-to-Wasm communication, and includes enterprise auditing and logging. However, we did not find Cupcake protections to be sufficient for providing a level of security guarantee. There are a few high-level findings that contribute to this statement:

- The ability to protect against bypasses using unaccounted for tools would require large and complex allowlists that are likely unmanageable for the end user (see TOB-QTY-LAB-CUPCAKE-2). The impact is that most policies could likely be bypassed by a dedicated adversary. However, as stated above, the process to reach that bypass would likely result in alerts due to denied actions. Note that it is likely impossible to reach a state where a set of policies could protect against all adversary actions and be deemed "safe" while excluding all "unsafe" ones.

- Implementing regex analysis for tool inputs (TOB-QTY-LAB-CUPCAKE-3) represents an improvement over current string comparison checks but remains insufficient as a comprehensive security control. While a regex-based prevention policy for the Bash tool could better detect variations of dangerous commands like `rm -rf` compared to simple string matching, it cannot reliably identify all bypass techniques and should not be considered a security guarantee.

- The signals feature allows policies to be bypassed in the current state (TOB-EQTY-LAB-CUPCAKE-4, TOB-EQTY-LAB-CUPCAKE-5) and could enable remote code execution by a dedicated adversary (TOB-EQTY-LAB-CUPCAKE-10). Additionally, the Cupcake integrity validations meant to protect the signals can be bypassed from inside the agent (TOB-EQTY-LAB-CUPCAKE-6). These issues are all fixable, but they contribute to the current interpretation of the tool.

- Cupcake can load an additional configuration, including policies, actions, and signals, via the `CUPCAKE_GLOBAL_CONFIG` environment variable if it is set. Attackers or malicious agents can exploit this variable to load a malicious Cupcake configuration (TOB-EQTY-LAB-CUPCAKE-11).

Additionally, we found two issues that, when remediated, would improve the existing security posture: a weakness in the logging design (TOB-EQTY-LAB-CUPCAKE-9) and an inconsistent application of keys (TOB-QTY-LAB-CUPCAKE-10).

## Recommendations

Trail of Bits recommends that EQTY Lab take the following recommendations under consideration. Although the tool is still under active development, early implementation of these practices is crucial to building a solid security foundation.

To address the risks associated with the threat paths we identified, we recommend the following measures:

- Consider implementing the following high-level recommendations to protect the integrity of the `.cupcake` directory. Its integrity is critical to any claims with respect

to agent alignment and guardrails. Specifically, an agent that can modify the signals, actions, or policies is detrimental to Cupcake operation.

    - Apply restrictive Unix permissions to the `.cupcake` directory and consider running the agent under a different user account to ensure the directory remains immutable to the agent.
    - The `.cupcake` directory location should be immutable to the agent and validated during runtime operation to make sure it has not been tampered with. The use of environment variables is not recommended for this value (see also the recommendation related to Cupcake's dependency on environment variables).

- Make the Cupcake trust integrity feature non-optional for any Cupcake implementations that involve signals or actions.
- Create comprehensive unit tests to validate that the trust integrity feature is operating correctly. This feature provides a level of protection against signal tampering.
- Review the requirement stated in the documentation that the key used in the signature for trust verification must be reproducible. As is, this creates a signature tampering risk if an attacker (1) knows the machine ID, Cupcake binary path, username, and project directory to derive the key, and (2) overwrites the trust file. Adequate protections of the `.cupcake` directory and the trust file (see previous findings) prevent an overwrite of the file. However, this approach remains vulnerable to key prediction by attackers with knowledge of the implementation. Consider adding a nonce per installation or `.cupcake` directory such that the key is reproducible but not predictable if reproducibility is required.
- Implement a regex to validate the input from policy tools to limit the opportunity for bypass policies.
- Remove Cupcake's dependency on environment variables. Since Cupcake operates in the same local environment as the AI agents it is applying guardrails to, the environment cannot be trusted. Hence, Cupcake configuration should not be modifiable through environment variables since a malicious agent could easily tamper with them. Consider using command-line arguments instead of environment variables.
- Make the Cupcake binary fail-proof and resilient to errors. It should catch all errors and crashes and handle them by returning by either default-allowing or default-denying the action that could not be evaluated against security policies. For maximal security, we recommend a default-deny policy.

# Summary of Findings

The table below summarizes the findings of the review, including details on type and severity.

| ID | Title | Type | Severity |
|---|---|---|---|
| 1 | Cupcake can be bypassed by user input by setting CUPCAKE_WASM_MAX_MEMORY to zero | Access Controls | Medium |
| 2 | Security policies could be bypassed using tools not accounted for in a policy | Data Validation | High |
| 3 | Bash command policy can be bypassed through various means | Data Validation | High |
| 4 | rulebook_security_guardrails can be bypassed using a symbolic link | Data validation | High |
| 5 | Signal tampering enables policy bypasses | Access Controls | High |
| 6 | Cupcake integrity verification can be disabled by removing trust manifest | Access Controls | High |
| 7 | .claude/settings.json is not protected and attackers can remove Cupcake hooks | Access Controls | Low |
| 8 | Cupcake's logs can be read by a malicious agent | Auditing and Logging | Low |
| 9 | Event keys might not be consistent | Configuration | Low |
| 10 | Allowing alternative configuration via environment variable is unsafe | Access Controls | High |
| 11 | HMAC signing key is predictable with limited knowledge | Cryptography | Medium |

# A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

| Vulnerability Categories | |
|---|---|
| **Category** | **Description** |
| **Access Controls** | Insufficient authorization or assessment of rights |
| **Auditing and Logging** | Insufficient auditing of actions or logging of problems |
| **Authentication** | Improper identification of users |
| **Configuration** | Misconfigured servers, devices, or software components |
| **Cryptography** | A breach of system confidentiality or integrity |
| **Data Exposure** | Exposure of sensitive information |
| **Data Validation** | Improper reliance on the structure or values of data |
| **Denial of Service** | A system failure with an availability impact |
| **Error Reporting** | Insecure or insufficient reporting of error conditions |
| **Patching** | Use of an outdated software package or library |
| **Session Management** | Improper identification of authenticated users |
| **Testing** | Insufficient test methodology or test coverage |
| **Timing** | Race conditions or other order-of-operations flaws |
| **Undefined Behavior** | Undefined behavior triggered within the system |

| Severity Levels | |
|---|---|
| **Severity** | **Description** |
| **Informational** | The issue does not pose an immediate risk but is relevant to security best practices. |
| **Undetermined** | The extent of the risk was not determined during this engagement. |
| **Low** | The risk is small or is not one the client has indicated is important. |
| **Medium** | User information is at risk; exploitation could pose reputational, legal, or moderate financial risks. |
| **High** | The flaw could affect numerous users and have serious reputational, legal, or financial implications. |

| Difficulty Levels | |
|---|---|
| **Difficulty** | **Description** |
| **Undetermined** | The difficulty of exploitation was not determined during this engagement. |
| **Low** | The flaw is well known; public tools for its exploitation exist or can be scripted. |
| **Medium** | An attacker must write an exploit or will need in-depth knowledge of the system. |
| **High** | An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue. |

# About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review assessments, supporting client organizations in the technology, defense, blockchain, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, Uniswap, Solana, Ethereum Foundation, Linux Foundation, and Zoom.

To keep up with our latest news and announcements, please follow @trailofbits on X or LinkedIn and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact or email us at info@trailofbits.com.

**Trail of Bits, Inc.**
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

# Notices and Remarks

## Copyright and Distribution



## Test Coverage Disclaimer

Trail of Bits performed all activities associated with this project in accordance with a statement of work and an agreed-upon project plan.

Security assessment projects are time-boxed and often rely on information provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test software controls and security properties. These techniques augment our manual security review work, but each has its limitations. For example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. A project's time and resource constraints also limit their use.