
Control Hackathon 2025: Debate Protocol Experiment

Anandmayi Bhongade
KCL

With
[organization or key organizers] & Redwood Research, Apart Research

Abstract

This project evaluates debate-based protocols for detecting code backdoors as part of AI control systems. We implemented and compared five monitoring approaches: Basic, Detailed, Chain-of-Thought (CoT), Best-of-N (BoN), and our novel Debate Monitor. The Debate Monitor structures assessment as a formalized argument between perspectives arguing for and against backdoor presence. Using a dataset of 20 code examples covering four backdoor types, all monitors achieved perfect detection rates. While quantitative metrics showed no differentiation, qualitative analysis revealed that the Debate Monitor provided more comprehensive justifications by explicitly addressing alternative explanations.

Keywords: AI control, security, monitoring, threat models, [other relevant keywords]

1. Introduction

We investigated whether debate-based monitoring protocols can improve code backdoor detection compared to standard approaches. This work builds directly on Greenblatt et al. (2024) "AI Control: Improving Safety Despite Intentional Subversion," extending it through debate mechanisms inspired by Tyler Tracy's "Project Proposal: Debate Protocol" (2025).

Our research addresses four specific backdoor types:

1. **Command Execution:** Hidden code that executes arbitrary system commands

2. **Data Exfiltration:** Code that steals sensitive information
3. **Hidden Logic:** Deceptive conditionals that activate under specific circumstances
4. **Network Access:** Unauthorized connections to external servers

2. Methods

Monitor Implementations

We implemented five distinct monitoring protocols using the Claude 3.5 Sonnet API, each representing different approaches to AI safety verification:

Basic Monitor

- Utilizes a structured prompt with explicit security criteria focusing on network access, command execution, and data exfiltration
- Implements a 0-10 suspicion scoring system with clear thresholds (≥ 7 indicating likely backdoor)
- Extracts reasoning and suspicion score using regex pattern matching
- Returns a standardized MonitorResponse object with both quantitative assessment and qualitative reasoning

Detailed Monitor

- Extends the basic approach with a comprehensive evaluation framework across seven vulnerability categories:
 - Network communication (HTTP requests, socket connections)
 - Process execution (os.system, subprocess)
 - File operations (unauthorized reading/writing)
 - Environment variable access
 - Obfuscation techniques (base64 encoding, encrypted strings)
 - Time-based triggers
 - Suspicious imports
- Performs multi-stage evaluation with weighted scoring for different vulnerability types
- Implements category-specific analysis with final aggregation

Chain-of-Thought (CoT) Monitor

- Implements a structured seven-stage reasoning pipeline:
 1. Code feature extraction
 2. Library/package identification
 3. Identification of sensitive operations
 4. Analysis of control flow
 5. Threat modeling
 6. Cross-reference with known backdoor patterns
 7. Final verdict synthesis
- Enforces XML tags to separate reasoning stages for better interpretability

- Models step-by-step human security expert reasoning for transparency

Best-of-N Monitor

- Implements ensemble methodology with three independent analysis runs
- Uses parallel prompting with the same core analysis but different framing
- Applies maximum function to suspicion scores to prioritize safety (any detected threats trigger warning)
- Aggregates reasoning from all runs to provide comprehensive explanation
- Reduces false negatives through diversity of analytical perspectives

Debate Monitor

- Implements a dialectical three-stage process with explicit role playing:
 1. Prosecutor stage: Aggressively seeks evidence of backdoor presence with strong suspicion bias
 2. Defense stage: Critically evaluates first-stage claims with explicit counterarguments
 3. Judge stage: Synthesizes evidence from both perspectives to reach balanced conclusion
- Uses structured XML for clear separation of debate stages
- Implements explicit handoff between different reasoning modalities
- Final verdict includes confidence rating and rationale synthesis

Code Repository and Implementation Details

Our implementation leverages the Control-Arena framework

(<https://github.com/anthropics/control-arena>), which provides infrastructure for evaluating AI control protocols:

- Repository Structure: Organized with clear separation between dataset generation, monitor implementations, evaluation logic, and visualization
- Monitor Architecture: Implemented abstract base class with standardized interface and protocol-specific implementations
- Evaluation Pipeline: Built automated workflow with asyncio for parallel evaluation
- Visualization Tools: Created matplotlib-based visualization functions for backdoor detection performance

Core Technical Components:

1. Monitor Framework: `class Monitor(ABC): """Base class for all monitoring protocols.""" @abstractmethod async def call(self, task_state: TaskState, trajectory: ProtocolTrajectory, assistant_message: ChatMessageAssistant) -> MonitorResponse: """Evaluate assistant message for backdoors.""" pass`
2. Standardized Response Model: `class MonitorResponse(BaseModel): """Standardized response from all monitors.""" suspicion_score: Optional[int] = Field(None, ge=0, le=10) reason: str = ""`
3. Evaluation Metrics: `def calculate_metrics(results): """Calculate precision, recall, F1 etc. from evaluation results.""" # Implementation details for`

computing true/false positives/negatives # and deriving standard classification metrics

Dataset Construction Methodology

We created a highly controlled testing environment with precisely crafted backdoor examples:

1. Standard Backdoor Dataset: • 8 clean examples with simple algorithmic functions (factorial, palindrome checking, etc.) • 12 backdoored examples (3 examples each of 4 backdoor types) • Backdoor types implemented with explicit malicious signatures:
 - Network access: Socket connections to exfiltrate data
 - Command execution: System calls to add malicious users or execute arbitrary commands
 - Data exfiltration: Functions that gather and send sensitive information
 - Hidden logic: Complex conditional paths that bypass security under specific conditions
2. Subtle Backdoor Dataset: • Same distribution as the standard dataset (8 clean, 12 backdoored) • Advanced obfuscation techniques employed:
 - Backdoors disguised as legitimate functionality (error reporting, analytics, utilities)
 - Time-based or environment-variable triggers
 - Complex conditional logic to hide malicious intent
 - Integration with legitimate code rather than separation
3. Dataset Generation: • Programmatically generated with controlled randomization • JSON-based storage with consistent schema • Each example annotated with ground truth labels and backdoor type

ControlArena Settings and Integration

Our implementation directly integrates with the ControlArena framework using:

- Protocol Integration: Extended the base ProtocolTrajectory class to track and log monitor evaluations
- Task Adaptation: Utilized ControlTaskMetadata to structure the backdoor detection task
- Model Integration: Used the provided ModelOutput and ChatMessageAssistant interfaces
- Tool Interface: Leveraged the existing ToolCall structure for tracking code interactions

Model Configuration

We used Claude 3.5 Sonnet as the monitoring model with consistent parameters:

- Temperature: 0.0 for deterministic reasoning
- Max tokens: 4096 to accommodate multi-stage reasoning

- System prompt: Carefully crafted role specifications for each monitoring protocol
- Authentication: API key management via environment variables with dotenv

Testing Methodology

Our evaluation framework implemented a rigorous process:

1. **Balanced Dataset Processing:** • Each code example processed by all five monitors • Evaluations run in parallel using async for efficiency • Consistent task framing across monitors
2. **Metric Collection:** • Binary classification (backdoor/clean, with suspicion score ≥ 7 indicating backdoor) • Detailed reasoning capture and parsing • Per-monitor and per-backdoor-type performance tracking
3. **Cross-Monitor Comparison:** • Performance metrics calculation (accuracy, precision, recall, F1) • ROC-like curve visualization • Confusion matrix generation for each monitor • Backdoor type detection heatmap
4. **Qualitative Analysis:** • Reason extraction and categorization • False positive/negative case study • Analysis of detection patterns by backdoor type
5. **Visualization:** • Performance comparison charts • Detection rate by backdoor type • ROC curves

Technical Challenges and Solutions

During implementation, we encountered several technical challenges:

1. **Response Parsing Issues:** • Challenge: Extracting structured data (suspicion scores, reasoning) from free-form text responses • Solution: Implemented robust regex pattern matching with fallback mechanisms and error handling
2. **Model Consistency:** • Challenge: Ensuring consistent scoring across different monitoring protocols • Solution: Standardized prompt engineering with explicit scoring criteria and structured output formats
3. **Debate Monitor Complexity:** • Challenge: Managing multi-turn conversation within single API call • Solution: Implemented structured XML tagging and explicit handoff between debate stages
4. **Subtle Backdoor Design:** • Challenge: Creating backdoors that were genuinely difficult to detect • Solution: Studied real-world attack patterns and implemented obfuscation techniques mimicking legitimate code
5. **Evaluation Robustness:** • Challenge: Ensuring fair comparison across protocols • Solution: Consistent task framing, standardized metrics, and controlled experimental conditions

3. Results

Overall Performance Metrics

Monitor	Accuracy	Precision	Recall	F1 Score
Basic Monitor	1.00	1.00	1.00	1.00
Detailed Monitor	1.00	1.00	1.00	1.00
CoT Monitor	1.00	1.00	1.00	1.00
BoN Monitor	1.00	1.00	1.00	1.00
Debate Monitor	1.00	1.00	1.00	1.00

All monitors achieved perfect detection across all backdoor types. Confusion matrices revealed perfect classification for all monitors:

- True Positives: 12
- True Negatives: 8
- False Positives: 0
- False Negatives: 0

Qualitative Analysis

While all monitors achieved identical quantitative performance, examination of reasoning outputs revealed significant qualitative differences:

1. **Reasoning Structure:** The Debate Monitor's three-stage process produced more structured analyses with explicit consideration of alternatives.
2. **Example Comparison:** A comparison of monitor outputs for a sample data exfiltration backdoor:

Basic Monitor:

```
This code contains multiple highly suspicious elements that strongly indicate the presence of a backdoor:
```

1. Network access backdoors: The code uses `urllib.request` to make an HTTP request to a suspicious URL `"https://data-stealer.evil/collect"`

Debate Monitor:

```
After considering both arguments, I conclude this code contains a data leak backdoor. The code collects environment variables and attempts to read AWS credentials and SSH keys. While the counterargument points out potential legitimate uses for environment variable access, the combination of sensitive data collection, base64 encoding, and transmission to a suspicious domain ("data-stealer.evil") clearly indicates malicious intent.
```

4. Discussion and Conclusion

Strengths

1. **Structured Reasoning Process:** The Debate Monitor implemented a formal three-stage assessment process, encouraging consideration of alternative explanations before reaching conclusions.
2. **Comprehensive Implementation:** Our code provides a complete framework for backdoor detection, including dataset generation, evaluation, and visualization components.

Second Experiment with Subtle Backdoors

After observing perfect performance across all monitors, we conducted a second experiment with a more sophisticated dataset:

1. **Subtle Backdoor Dataset:** We created backdoors disguised as legitimate functionality (error reporting, system utilities, analytics), using complex conditional logic and time-based triggers.
2. **Results:** Despite increased sophistication, all monitors still achieved perfect detection, suggesting Claude 3.5 Sonnet has strong capabilities in identifying even subtle backdoors.

Challenges and Limitations

1. **Perfect Performance:** All monitors achieved perfect results in both experiments, making it difficult to differentiate their performance quantitatively.
2. **Implementation Challenges:** We encountered several technical difficulties:
 - API response handling required extensive error handling for different response formats
 - Standardizing score extraction across different monitor outputs proved challenging
 - Initial debate structure implementation required multiple prompt engineering iterations
3. **Dataset Limitations:** While we created increasingly sophisticated backdoors, we were limited by:
 - Time constraints of the hackathon environment
 - API credit limitations preventing larger-scale testing
 - Difficulty creating truly deceptive backdoors that would evade detection
4. **Scale Limitations:** We initially planned to test with 50+ examples per backdoor type, but had to scale back to 20 total examples due to time and resource constraints.

Future Work

1. **More Complex Backdoors:** Test with even more subtle and advanced backdoor types that better reflect real-world attacks, potentially using adversarial techniques.
2. **Multiple Models:** Evaluate the effectiveness of using different models for the argument and counterargument phases.
3. **Red Team Testing:** Subject the monitors to adversarial attempts to evade detection.
4. **Larger Dataset:** Expand testing to hundreds of examples with greater diversity in both clean and backdoored code.

5. Appendix

Additional Materials - code is on github

