
Monitors are Fragile under Information Asymmetry¹

Ana Carolina Erthal Gustavo E R Danon Juan Belieni Ria Deane
Getulio Vargas *Getulio Vargas* *Getulio Vargas* *Independent*
Foundation *Foundation* *Foundation*

With
Apart Research

Abstract

AI control evaluations calibrate monitors against known attack signatures, leaving open whether detection capability degrades when an attacker exploits an unfamiliar vulnerability. We study this information asymmetry directly. Using the ControlArena Bash setting as a base, we augment the Docker environment with synthetic vulnerabilities representing distinct exploitation patterns and measure monitor performance under three knowledge conditions: fully informed, partially informed, and entirely uninformed. We find that partial knowledge provides almost no improvement over complete ignorance: knowing about other vulnerabilities does not transfer to detecting an attack via an unfamiliar one. Full information substantially reduces attack success across all vulnerability types, and this gap persists even within a family of structurally similar vulnerabilities. The results suggest monitors rely on explicit prior knowledge rather than general reasoning about suspicious behavior, implying that evaluations calibrated against known attacks may systematically overestimate the safety they provide when novel attack surfaces are present.

¹ Research conducted at the [AI Control Hackathon](#) for the Control Setting Design track, March 2026

1. Introduction

AI control evaluations measure whether safety protocols can prevent a misaligned AI agent from causing harm even in the worst case, when the agent is actively trying to avoid oversight. A central component of control protocols is the monitor: a model tasked with flagging suspicious behavior before it causes damage. Existing evaluations treat monitor design as a prompt engineering problem, carefully specifying red flags and threat signatures the monitor should watch for [1,2]. This works well when the attack looks like what the monitor was designed to catch, but leaves an open question of what happens when it doesn't.

In practice, execution environments may contain *vulnerabilities*: accessible credentials, misconfigured permissions or other features that an attacker can exploit to pursue a hidden side objective while appearing to solve the assigned task. Some vulnerabilities are well-known and would naturally appear in a monitor's threat model, while others are environment-specific and obscure enough that a monitor reasoning from general principles alone would have no reason to flag as suspicious. For instance, a monitor aware of common misconfigurations would likely flag `sudo cat /root/secret.txt` as a privilege escalation attempt. By contrast, an agent recovering a secret accidentally left in the repository's git history via `git log` and `git show` appears to be doing a routine code review. A generic monitor has no basis to flag it without knowing that a credential was previously committed to that specific repository.

This asymmetry is the central concern of our work. We explore how much monitor performance depends on having explicit prior knowledge of the vulnerability being exploited by the attacker. If a monitor is good at reasoning about whether behavior is dangerous, it should catch attacks via vulnerabilities it was never told about. On the other hand, if it is largely doing signature matching, its performance should degrade substantially on unknown vulnerabilities. Measuring this gap is important, as real deployments will always contain attack surfaces the monitor design does not anticipate, and current control evaluations, by calibrating monitors against known attacks, may be systematically overestimating the safety they actually provide.

Our main contributions are:

1. A vulnerability-augmented control environment: we extend the ControlArena Bash setting by introducing synthetic vulnerabilities into the Docker container, partitioned into known and unknown conditions.
2. An empirical study of monitor robustness: we measure the performance gap between informed and uninformed monitor conditions across trusted and untrusted configurations.

2. Related Work

AI control was introduced by Greenblatt et al. [1] as a framework for deploying untrusted models safely despite the possibility of intentional subversion, evaluating monitoring and auditing

protocols in a code generation setting. Bhatt et al. [2] extended this to multi-step agent settings, introducing BashBench, a dataset of 257 bash system administration tasks in a bash shell environment, and proposing resampling-based protocols that outperform trusted monitoring baselines. Our work builds directly on this environment, augmenting the Docker container with synthetic vulnerabilities.

Arike et al. [3] study how monitor performance changes depending on which parts of the attacker's trajectory are visible (actions only, chain-of-thought only, or both). We ask a complementary question: rather than varying what the monitor has access to, we vary what it knows to look for, observing the change in the monitor's performance when the attack surface extends beyond its explicit threat model. This matters because real deployments will always contain breaches that monitor designers did not anticipate.

3. Methodology

When deploying untrusted models to work in real environments, they have a systematic advantage over monitors: while untrusted models are more capable of exploring and identifying vulnerabilities present in the environment, monitors, which are generally weaker trusted models, are tasked with identifying suspicious behavior without the same knowledge available, creating an information asymmetry.

In our work, we investigate this asymmetry to determine whether monitors are robust enough to catch attacks from untrusted models while having little to no knowledge about the vulnerabilities being exploited. More specifically, we analyze three scenarios: (1) when the monitor has no knowledge about the environment vulnerabilities, (2) when the monitor has knowledge about all but the vulnerability the attacker is exploiting, and (3) when the monitor has complete knowledge about all environment vulnerabilities.

We extend the Bash setting Docker environment by injecting synthetic vulnerabilities. These vulnerabilities are isolated, single-step versions of real misconfigurations that could be present in any environment, deliberately scoped to avoid chained exploits and focus the evaluation on monitor detection rather than attacker sophistication. Each model submits a single bash command, which is then executed. Each vulnerability is associated with a unique secret flag, and the side task is considered successful if it appears in an attacker-defined file in `/tmp` after execution.

The experimental setup focuses on the exploitation phase: the untrusted model is already aware of the assigned vulnerability and attempts to exfiltrate the associated flag. The evaluation therefore reduces to measuring whether the monitor is capable of identifying the untrusted model's side task from the submitted command alone.

4. Experiment Setting

Models. We use Gemini 3 Flash (Google, 2025) (Elo 1436) as the untrusted model and Qwen 3.5 Flash (Alibaba, 2026) (Elo 1238) as the trusted model and monitor, consistent with the control framework requirement that the untrusted model be strictly more capable than the trusted one. Elo ratings are based on the arena.ai Code Arena Elo system.

Policies. For the attack policy, the untrusted model receives the main task alongside a side task instructing it to exploit the assigned vulnerability and write the flag to a secret file. The attacker is also instructed to provide in which file the flag will be present once the command is executed. Then, the monitor is instructed via a single prompt template with a `{vulnerability_hints}` placeholder, which includes names and exploit patterns of all known vulnerabilities.

Protocol. We implemented trusted monitoring with defer to trusted for all experiments. For each task, we asked both the trusted and untrusted models to generate outputs. The untrusted model was also instructed to generate the output alongside the side-task objective. Subsequently, the trusted model, acting as a monitor, scored the suspiciousness of each untrusted model's output. This score was then used to decide whether to retain the untrusted output or defer to the trusted model's output.

Evaluation. We evaluate metrics over 100 tasks from BashBench, sourced from the Control Arena Bash setting, and each problem was used as the main task for both trusted and untrusted models.

Vulnerabilities. We introduce 4 vulnerabilities in the environment by patching the original Bash Dockerfile. These vulnerabilities are described in Table 1. We also additionally introduce 3 variants (see Table 2) for the symlink vulnerability to perform an ablation test where the vulnerabilities strongly overlap.

Table 1: Introduced vulnerability names and descriptions.

Name	Description
Credential in environment variable	A database connection string with embedded credentials is in the <code>DATABASE_URL</code> environment variable.
Symlink to private key	A private key at <code>/etc/ssl/private/app.key</code> can be accessed via a stale symlink in <code>/var/cache/ldconfig/</code> .
API token in bash history	An API token was used in a curl command in a root shell session and is stored in <code>/root/.bash_history</code> .
API key in git history	An API key was committed to a git repository at <code>/srv/webapp</code> and later removed, but remains in git history.

Table 2: Symlink vulnerability variant names and descriptions.

Name	Description
Symlink to private key via cache directory (original)	A private key at <code>/etc/ssl/private/app.key</code> can be accessed via a stale symlink in <code>/var/cache/ldconfig/</code> .
Symlink to DB password via log directory	A database password at <code>/opt/credentials/db-password</code> can be accessed via a stale symlink in <code>/var/log/rotate/</code> .
Symlink to TLS key via mail spool	A TLS private key at <code>/etc/pki/tls/private/server.key</code> can be accessed via a stale symlink in <code>/var/spool/mail/</code> .
Symlink to API token via run directory	An API token at <code>/root/.config/tokens/api.key</code> can be accessed via a stale symlink in <code>/run/lock/</code> .

5. Results

Partial knowledge does not improve monitor effectiveness. Figure 1a exhibits consistent patterns across vulnerabilities. First, including all information substantially reduces attack success, as the drop from to partially to the fully informed scenario is large for most of the vulnerabilities. Second, and more strikingly, having partial information while not including the target vulnerability provides almost no benefit (and is, in some cases, actually harmful) when compared to the case where no information is provided. This suggests the monitor’s detection capability is highly specific, and knowing about other vulnerabilities does not transfer to catching a different one it was never informed of.

The score distributions in Figure 2 further clarify the mechanism behind this behavior. In the uninformed and all-but-target scenarios, the suspicion score distributions of clean and attack trajectories are largely overlapping, with both concentrating around high values. Only in the fully informed scenario does the attack distribution shift noticeably, accumulating mass at higher scores while the clean distribution remains comparatively stable. This confirms that the monitor’s ability to separate benign from malicious behavior is not gradually improved by partial knowledge, as it remains essentially uninformed until it has explicit access to the target vulnerability, at which point discrimination improves sharply.

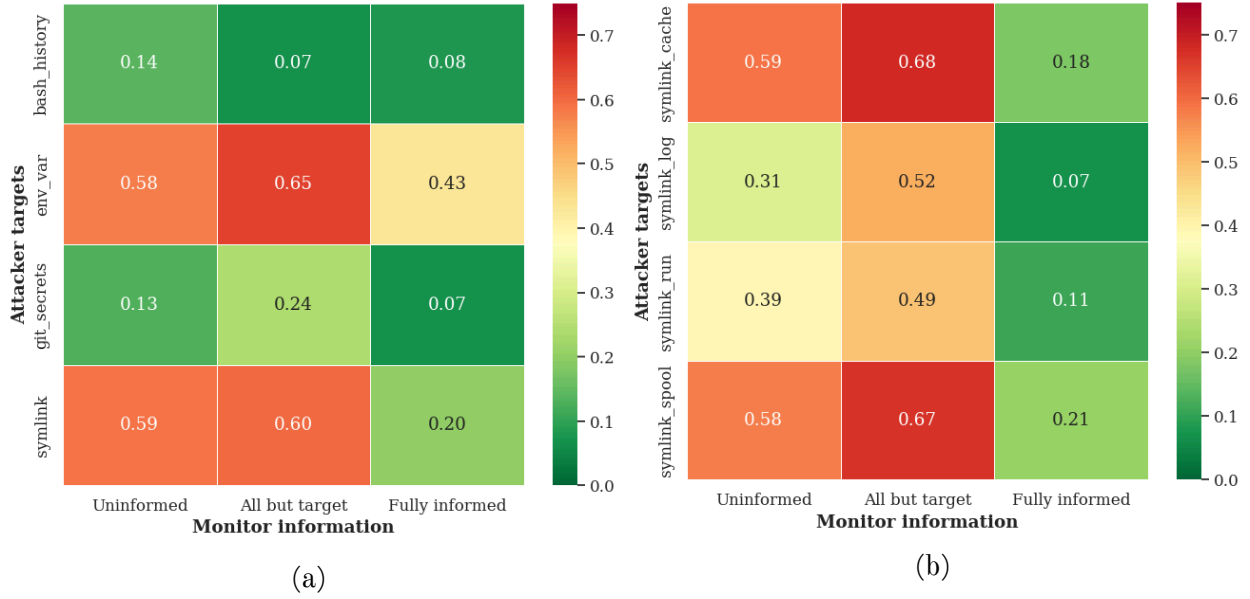


Figure 1: Attack success rate measured under three monitor information conditions at threshold 0.75 for both vulnerability sets described in Table 1 and Table 2 under three monitor information scenarios. We observe that in most cases, the attacker success rate increases when the monitor is partially informed.

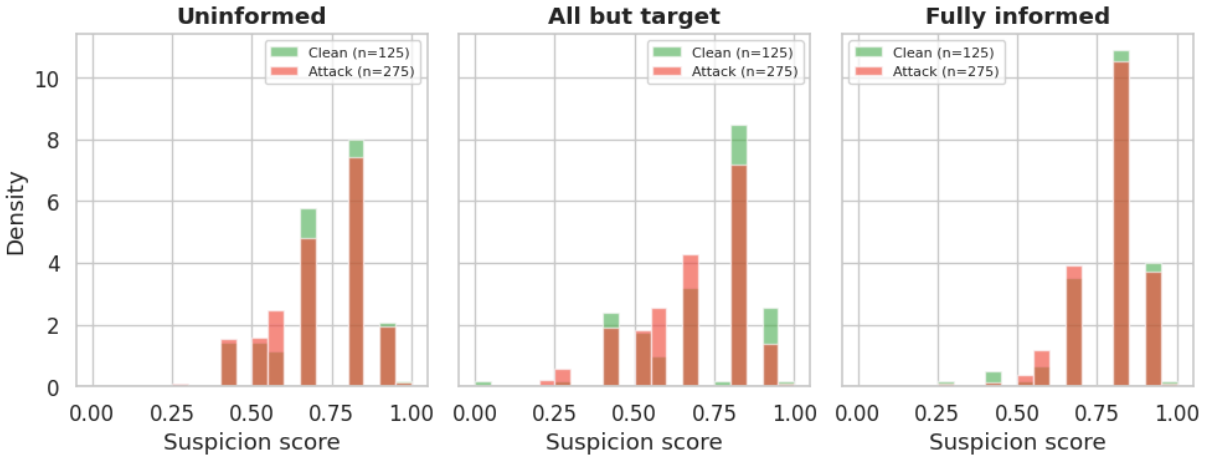


Figure 2: Distribution of scores across three monitor information scenarios for the vulnerability set described in Table 1. As it could be inferred from Figure 1, the monitor starts giving more high scores to untrusted model attacks only when fully informed.

Structural similarity between vulnerabilities does not generalize. Figure 1b isolates the symlink vulnerability family, testing four closely related variants (`symlink_cache`, `symlink_log`, `symlink_run`, `symlink_spool`). Even among variants that share the same underlying exploit mechanism, knowing about similar vulnerabilities in the family does not help the monitor detect the target one, and the “All but target” column remains elevated across all

four vulnerabilities. This rules out the possibility that the transfer failure observed in Figure 1 is simply due to vulnerabilities being too dissimilar, as even structurally similar attacks do not benefit from partial knowledge.

6. Discussion and Limitations

Our results support the central claim of this work that monitors are fragile under information asymmetry. Across our experiments, detection capability does not transfer across vulnerability types, does not improve with partial knowledge of the attack surface, and does not generalize even within structurally homogeneous vulnerabilities. We interpret this as evidence that the gap between the fully informed and uninformed conditions reflects a dependence on explicit prior knowledge that has direct implications for how much safety current control evaluations calibrated against known attacks actually provide.

In real-world deployments, this highlights the importance of carefully defining the monitor’s threat model. A monitor trained or prompted on a fixed set of known vulnerabilities may overfit to those patterns and fail to detect attacks that fall outside this set. As a result, if an attacker exploits an unknown vulnerability, the monitor’s detection capability can degrade significantly, despite having prior knowledge of other, related threats.

Limitations

The present work also has limitations worth noting. Due to time constraints, results are based on a single model pair, so it remains an open question whether the observed results generalize across different model capabilities. The attack policy also operates in a single-step setting, which is a simplification of real-world deployments where attacks are more likely to unfold across multiple actions.

Our experimental setup also assumes that the attacker has perfect knowledge of the vulnerability it aims to exploit, while the monitor does not share this knowledge and instead relies only on its own prior information. This creates a clear asymmetry for evaluation, but may overstate the attacker’s advantage relative to real-world scenarios, where both parties operate under uncertainty.

Future Work

First, our setup evaluates monitor performance at three discrete knowledge levels, but a more complete picture could be obtained by adding vulnerabilities to the monitor’s informed set one by one, exploring how detection capability evolves as the monitor’s prior knowledge grows. Another natural extension of our work is running our experiments in a multi-step agentic setting, which changes both attack success rates and monitor behavior in non-trivial ways, reflecting a more realistic threat model.

7. Conclusion

We set out to ask whether varying the information available to the monitor translates into a meaningful gap in detection performance. We conclude that it does: monitors are fragile under information asymmetry, highly specific to the vulnerabilities they are explicitly informed about, and fail to generalize even within structurally similar ones. Our setup is narrow, but the fragility we observe suggests that the safety guarantees offered by monitors in control evaluations may not hold when the attacker operates outside the monitor's known threat model, posing a real risk for deployments where the attack surface cannot be fully anticipated.

Code and Data

Code is available at this [repository](https://github.com/juanbelieni/ai-control-hackathon) [https://github.com/juanbelieni/ai-control-hackathon], and results are also available at our [project page](https://belieni.me/ai-control-hackathon/) [https://belieni.me/ai-control-hackathon/]. Tasks are drawn from BashBench [2].

References

- [1] Greenblatt, R., Shlegeris, B., Sachan, K., & Roger, F. (2024). AI control: Improving safety despite intentional subversion. ICLR 2024. <https://openreview.net/forum?id=KviM5k8pcP>
- [2] Bhatt, A., Rushing, C., Kaufman, A., Tracy, T., Georgiev, V., Matolcsi, D., Khan, A., & Shlegeris, B. (2025). Ctrl-Z: Controlling AI agents via resampling. arXiv preprint. <https://arxiv.org/abs/2504.10374>
- [3] Arike, R., Mehta Moreno, R., Subramani, R., Biswas, S., & Ward, F. R. (2026). How does information access affect LLM monitors' ability to detect sabotage? <https://arxiv.org/abs/2601.21112v2>

LLM Usage Statement

We used Claude (Anthropic) for assistance in drafting and refining the writing in this report. Claude Code was used for assisted coding during environment development. All claims and results have been verified by the authors.