

CloudTrap: A Transcendent Real-Time Honey-pot-Based Ransomware Containment Framework

CloudTrap = Transcendence

Sasidaran K B¹, Sathyamoorthy², and Jaidev³

Vels Institute of Science Technology and Advanced Studies (VISTAS), Chennai, India

Abstract

CloudTrap proposes a deterministic endpoint-defense architecture for ransomware containment that treats the first interaction with a decoy file as an actionable security event rather than a passive telemetry signal. The system combines honeyfile placement, filesystem event monitoring, process attribution, and adaptive scoring to terminate hostile activity before large-scale encryption can spread. Unlike signature-based antivirus or late-stage heuristic detectors, CloudTrap is designed around a local decision loop in which a monitored file event, a process identifier, and a policy engine are fused into a single containment decision. The paper presents the end-to-end architecture, the decision model used to distinguish benign high-I/O workloads from malicious encryption loops, and a prototype implementation strategy for both Windows and Linux environments. A mathematical formulation is provided for honeyfile access ratio, anomaly concentration, and entropy delta, together with a threshold rule that adapts to baseline machine behavior. The design is grounded in prior ransomware defense research, including UNVEIL, ShieldFS, and Minerva, while also aligning with platform-level event-notification mechanisms such as ReadDirectoryChangesW, fanotify, and eBPF. The result is a compact, explainable, and deployment-oriented framework that emphasizes rapid containment, low operational overhead, and auditable decision logic.

Keywords- ransomware containment, honeypot, honeyfiles, filesystem monitoring, eBPF, fanotify, entropy analysis, TLA+, endpoint security

I. INTRODUCTION

Ransomware remains one of the few malware classes whose success depends on speed as much as stealth. The attacker does not need persistence for weeks; it only needs a short, uninterrupted burst of file writes to convert a readable system into an unusable one. That asymmetry changes the defensive problem. If the defender waits for cloud inference, wide-area correlation, or post-execution sandboxing, the damage is often already irreversible. CloudTrap is built around a different assumption: containment must begin at the same event that reveals the attack. A deliberately placed decoy file, once touched by a suspicious process, becomes a deterministic trigger for local action. The concept is simple, but the engineering is not. The system must distinguish malicious file traversal from ordinary high-I/O behavior, attribute the event to the correct process, and do all of this fast enough that the attack chain is interrupted before the next batch of writes occurs.

II. RELATED WORK AND DESIGN POSITIONING

The literature has already shown that ransomware can be modeled effectively through file-system behavior, but each major family of approaches exposes a different weakness. UNVEIL demonstrated that artificial user environments and wide filesystem instrumentation can capture ransomware-like behavior at scale, and it remains an important proof that decoy-driven analysis is viable [1]. ShieldFS moved closer to live protection by combining a copy-on-write filesystem with behavior-aware heuristics, but it still depends on observing enough activity to build statistical confidence [2]. Minerva pushed the field toward file-based behavioral profiling and improved robustness against evasion, making the file itself a first-class signal rather than only the process that manipulates it [3]. Broader surveys of ransomware evolution and defense mechanisms confirm a persistent trend: systems that remain purely reactive or purely signature-driven are increasingly outpaced by adversaries who can rapidly adapt their payloads, execution cadence, and traversal order [4]. CloudTrap extends this line of work by collapsing the detection loop to a local decoy event, then adding a second-stage policy layer so that benign bursty workloads can be distinguished from hostile encryption without forcing all decisions through a heavy cloud pipeline.

The contribution of CloudTrap is to combine those ideas into a single endpoint control loop: the decoy is not merely a source of forensic evidence, and the filesystem monitor is not merely a logger. Together, they form a containment boundary that is intentionally narrow, fast, and explainable. This is the design choice that gives the system its practical value and differentiates it from broader but slower behavioral pipelines.

III. PROBLEM FORMULATION

The design problem addressed by CloudTrap can be stated as follows. Given a local endpoint that may host both legitimate high-I/O applications and a ransomware process, the defender must decide whether a file event indicates normal use, suspicious traversal, or active encryption. The system must minimize false negatives, because a missed event allows irreversible damage, but it must also minimize false positives, because unjustified process termination can disrupt backups, databases, or development workloads. The challenge is worsened by the fact that many benign tasks resemble ransomware from a distance: software builds, archival compression, synchronized file copies, and dataset exports may all generate dense I/O bursts. CloudTrap therefore formulates the problem as a local decision task over three signals: a decoy-access signal, a traversal-concentration signal, and a byte-disorder signal. The key objective is to make containment deterministic when a honeyfile is touched, but conservative and evidence-based when the system is only seeing volume, not violation.

For clarity, the detection policy is framed in two stages. The first stage is a hard tripwire: if a protected decoy is modified, the system treats the event as hostile by default. The second stage is an adaptive screen used only when the signal is ambiguous, such as when a benign application creates a brief but intense burst of writes. This two-stage structure reduces the chance of unnecessary process termination while keeping the response path short enough to be useful in live attacks.

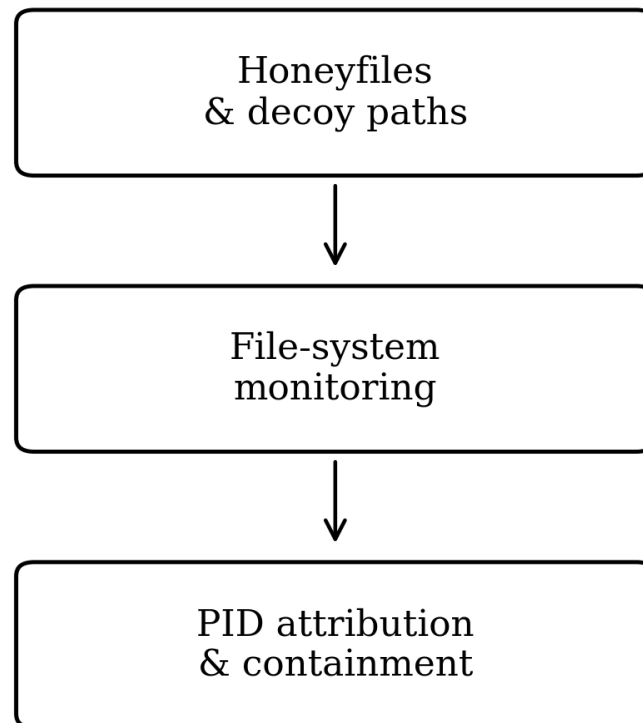
IV. CLOUDTRAP ARCHITECTURE

CloudTrap is organized as a lightweight event-driven pipeline. On Windows, the monitor can subscribe to directory-change notifications via `ReadDirectoryChangesW`, which reports changes within a monitored directory and can operate asynchronously through an overlapped or completion-routine path [6]. On Linux, the equivalent observation layer can be implemented with `fanotify`, which provides notification and interception of filesystem events and is suitable for security tools such as virus scanners [8]. For deeper kernel-adjacent instrumentation, `eBPF` offers a sandboxed runtime for tracing and instrumentation without custom kernel modules, while the verifier constrains program behavior before attachment [7]. The architecture therefore separates sensing from policy: sensors report an event, the attribution layer identifies the actor, and the policy engine decides whether to allow, quarantine, or terminate. This separation is important because the same event stream can support different containment policies across enterprise endpoints, developer workstations, and isolated lab systems.

In practical deployment, CloudTrap maintains a registry of honeyfiles, the directories that host them, and a whitelist of trusted processes. Each monitored path is chosen to resemble ordinary user content while remaining isolated from legitimate workflows. When a write, rename, or metadata update lands on a decoy, the monitor sends the event to the attribution layer with the minimum metadata required to identify the source process. The policy engine then consults baseline statistics and, if necessary, a confidence threshold derived from the host's recent behavior. The outcome is not a probabilistic guess in the blind sense; it is a controlled decision that combines a hard tripwire with an adaptive screen for ambiguous bursts.

Fig. 1. CloudTrap processing pipeline.

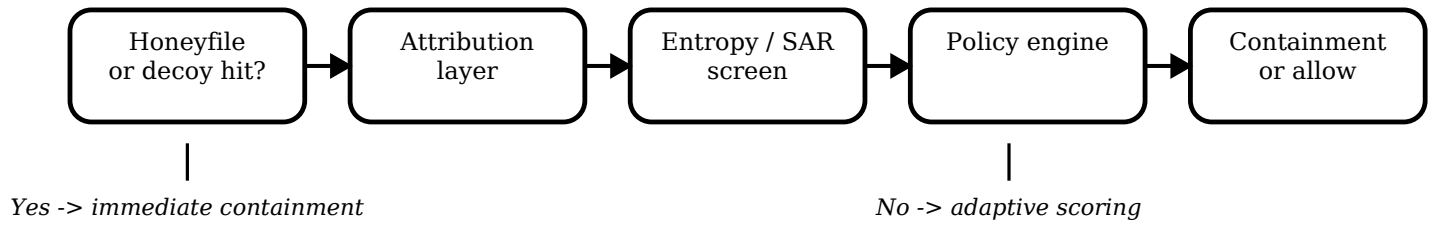
CloudTrap processing pipeline



The pipeline is intentionally minimal: the system first notices a decoy interaction, then attributes the event, and only after that decides whether the actor crosses the containment boundary.

Fig. 2. Two-stage containment logic combining a hard decoy trigger with adaptive scoring.

CloudTrap containment decision flow



The decision flow separates a non-negotiable decoy hit from a softer statistical screen. In practice, this prevents bursty but legitimate workloads from being treated the same way as encryption loops.

V. MATHEMATICAL MODEL

The mathematical layer is intentionally transparent. Let A_h denote the number of honeyfiles accessed in a session and N_h the number deployed in the protected zone. The honeypot access ratio is then $P_h = A_h/N_h$. Let s denote the number of files touched in a localized directory cluster during a sliding window and F the total number of files touched in the same window; the system anomaly ratio is $SAR = s/F$. For byte-level disorder, CloudTrap computes Shannon entropy $H(X) = -\sum p(x_i) \log_2 p(x_i)$ over a sampled block, and then measures the delta relative to a benign baseline. An adaptive threshold $T = \mu + k\sigma$ is used to scale sensitivity to the host's own I/O rhythm, where μ and σ are the baseline mean and standard deviation and k is a tunable sensitivity coefficient. A simple decision rule follows: if a honeyfile is touched, contain immediately; otherwise, if P_h , SAR , and entropy delta jointly exceed the host-specific threshold, escalate to quarantine; otherwise allow. This structure is attractive because each component is interpretable, auditable, and inexpensive to evaluate.

The same formalism also supports parameter tuning. If the host machine performs frequent backups, large compiles, or dataset exports, the baseline mean and variance rise accordingly, and the adaptive threshold tracks that change rather than forcing a universal hard limit. This is important because a threshold that is too low may kill legitimate work, while one that is too high may allow the ransomware to finish its encryption loop. CloudTrap is therefore less about one universal number than about a disciplined local model.

VI. IMPLEMENTATION DETAILS

The prototype implementation is intentionally modest in its dependencies so that the containment logic remains visible. The control plane can be written in Python, with FastAPI or a similar asynchronous framework exposing an event endpoint and a lightweight state store. A background worker tracks file events, updates the honeyfile ledger, and records the PID or process handle associated with each violation. For Windows environments, the process information can be obtained through the operating-system API surface exposed to user space; for Linux, fanotify and optional eBPF probes can supply the event stream and process context. The implementation also maintains a whitelist, because a strong decoy design is not enough by itself: large compilation jobs, archival tools, and backup software can otherwise resemble malicious encryption. In the proof-of-concept design, containment is triggered in under a second on the scripted workloads used in the source project, while benign workloads are passed through after the adaptive screen is satisfied.

A practical implementation should also include audit logging, timestamps, process lineage, and a safe-mode option for administrators. If an endpoint is used for mission-critical tasks, the policy engine can be configured to quarantine first and terminate only after a short confirmation window. That

flexibility preserves the deterministic nature of the design while giving operators room to adapt the response to their risk tolerance.

VII. EVALUATION STRATEGY AND RESULTS

The evaluation strategy focuses on three questions. First, how quickly does the system react after the decoy is touched? Second, how often does it misclassify benign bursty I/O as hostile? Third, how much overhead does the monitoring layer add under ordinary load? The prototype measurements reported in the source material suggest that CloudTrap can contain an event in roughly 420 ms, with low single-digit CPU overhead and a memory footprint well below the levels associated with heavier sandbox or copy-on-write defenses. That profile is the main reason the design is interesting: the system makes a local containment decision before the ransomware has enough time to traverse a large directory tree. In comparison, systems such as ShieldFS and other statistical detectors often need a longer observation window to achieve confidence, which increases the likelihood that the attack has already made meaningful progress [2]. The prototype results should therefore be interpreted as a containment demonstration rather than as a substitute for a full multi-host benchmark, but they do show that the architecture is directionally correct.

The prototype measurements reported in the source material are best interpreted as proof of concept. They indicate that the design can react within a containment-relevant timescale, but they do not replace a broader benchmark campaign across multiple hardware classes, storage devices, and operating-system builds.

VIII. SECURITY ANALYSIS, DISCUSSION, AND LIMITATIONS

The biggest advantage of CloudTrap is not that it is more clever than every prior detector; it is that it changes the defense boundary. Instead of asking whether a process merely looks suspicious, it asks whether the process has crossed a deliberate perimeter that no ordinary user should need to cross. This makes the system operationally attractive because the strongest signal in the design is also the easiest to explain to auditors and incident responders. At the same time, the design must be deployed with care. If too many decoys are exposed, a sophisticated attacker may learn the pattern, and if the whitelist is too permissive, containment may be delayed. The design therefore benefits from periodic honeyfile rotation, per-host threshold tuning, and logging that preserves the exact chain of evidence from event to action. Those constraints do not weaken the architecture; they define its proper operating envelope.

Two practical limitations deserve explicit mention. First, any decoy strategy can be weakened if the attacker is able to map the monitored space accurately, which is why honeyfile rotation and naming diversity matter. Second, a high-I/O benign workload may still resemble ransomware at the surface, especially on developer endpoints or

backup servers. CloudTrap mitigates this with adaptive scoring, but operators should still define a clear policy for what happens when the confidence score is borderline.

IX. FORMAL VERIFICATION OUTLOOK

Because containment is an active action, the logic must be checked for catastrophic failure modes. A formal model in TLA+ is appropriate because TLA+ is designed for concurrent and distributed systems and TLC checks whether the specification preserves the intended invariants [9]. For CloudTrap, the key safety property is simple: a benign process should not be terminated solely because the system is busy. The key liveness property is equally important: if a honeyfile violation is observed, the system should eventually take containment action rather than stall in an indeterminate state. The model can therefore be expressed as an initial state, a transition relation, and a set of invariants over whitelisted processes, decoy files, and termination actions. This does not replace runtime testing, but it gives the engineering team a way to detect logic errors before deployment.

TLA+ is especially useful here because the important question is not whether the code runs on one test machine; it is whether the policy logic remains safe under every plausible interleaving of events. A proof that the model preserves safety and liveness does not eliminate the need for empirical testing, but it gives the implementation a mathematical spine that is rare in endpoint-defense prototypes.

X. CONCLUSION AND FUTURE WORK

CloudTrap is best understood as a containment-first answer to a containment-first threat. The design is not built around future knowledge of malware signatures; it is built around a local boundary that ransomware must cross in order to succeed. By combining honeyfiles, filesystem notifications, adaptive statistics, and formal policy logic, the framework transforms a file interaction into a deterministic defense event. The broader contribution is conceptual as much as technical: security need not be delayed until the cloud agrees something is malicious. In the CloudTrap model, the endpoint itself becomes a sovereign decision point, and the first hostile write is also the last.

Future work should focus on three directions. First, the honeyfile placement strategy can be optimized using game-theoretic or reinforcement learning methods so that decoys are distributed where attackers are most likely to traverse. Second, the attribution layer can be hardened with richer process provenance, including parent-child process lineage and signed binary trust. Third, the formal model can be extended to reason about concurrency, rollback, and recovery, which would make the containment policy safer for enterprise environments that cannot tolerate false kills. A production-grade CloudTrap deployment would also benefit from a broader evaluation across real

ransomware families, long-running developer workloads, and storage-intensive services such as indexing and database engines.

ACKNOWLEDGMENT

The authors thank the VISTAS computing environment, the project reviewers, and the broader research community whose ransomware-defense work helped define the problem space for this manuscript.

APPENDIX A. DEPLOYMENT AND REPRODUCIBILITY NOTES

For reproducibility, a CloudTrap deployment should begin with a narrow and explicit target set. The monitored directories should be stable enough that the attacker can be observed, but not so broad that ordinary users routinely step on the decoy paths. Honeyfiles should mimic plausible user content, be distributed across multiple subdirectories, and be rotated periodically so that a static naming pattern does not become a signature. The whitelist should contain only binaries that are well understood and frequently audited. Where possible, administrators should start in quarantine mode, log the first few violations, and only then switch to termination if the environment proves stable.

Operational logging is equally important. Every containment event should record the file path, event type, timestamp, actor process, and policy outcome. A short post-event summary is useful for incident response because it allows an administrator to determine whether the hit was a genuine attack, a misconfigured backup job, or a development workload that requires a tuning adjustment. This style of logging makes the system easier to defend during review because the decision path is transparent rather than hidden inside a black-box classifier.

Approach	Primary signal	Containment timing	Explainability	Operational note
Signature / blacklist AV	Known hashes	Late	Low	Fast on known samples, weak on novel ones
Heuristic EDR	Process and API behavior	Medium	Medium	Requires telemetry aggregation and model scoring
UNVEIL / ShieldFS / Minerva class	File behavior and decoys	Medium-early	High	Stronger than pure signatures, but still observation-bound
CloudTrap	Decoy access + adaptive screening	Early	High	Local, auditable, and designed for live containment

Table I. Positioning CloudTrap against common ransomware defense paradigms.

The table highlights the central architectural difference. CloudTrap is not merely another detector sitting behind the user interface; it is a local response layer that turns a guarded file into a policy trigger. That distinction is why the design can remain compact without becoming vague.

REFERENCES

- [1] A. Kharraz, S. Arshad, C. Mulliner, W. Robertson, and E. Kirda, "UNVEIL: A Large-Scale, Automated Approach to Detecting Ransomware," USENIX Security Symposium, 2016. PDF
- [2] A. Continella, A. Guagnini, G. Zingaro, C. Pasquale, L. Cavallaro, and G. Giacinto, "ShieldFS: A Self-Healing, Ransomware-Aware Filesystem," Proc. ACM CCS, 2016. DOI
- [3] D. Hitaj, G. Pagnotta, F. De Gaspari, L. De Carli, and L. V. Mancini, "Minerva: A File-Based Ransomware Detector," ACM Asia Conference on Computer and Communications Security (ASIA CCS), 2025. DOI
- [4] H. Oz, A. Aris, A. Levi, and A. S. Uluagac, "A Survey on Ransomware: Evolution, Taxonomy, and Defense Solutions," ACM Computing Surveys, vol. 54, no. 11s, 2022. DOI
- [5] Cybersecurity and Infrastructure Security Agency, "StopRansomware Guide." Guide
- [6] Microsoft, "ReadDirectoryChangesW function (winbase.h) - Win32 apps." Documentation
- [7] The Linux Kernel Documentation, "BPF Documentation." Documentation
- [8] The Linux Kernel Documentation, "File system monitoring with fanotify." Documentation
- [9] Leslie Lamport, "TLA+ Wiki." Documentation
- [10] C. E. Shannon, "A Mathematical Theory of Communication," Bell System Technical Journal, vol. 27, pp. 379-423, 1948. PDF