International Journal of Creative and Open Research in Engineering and Management
ISSN: 3108-1754 (Online)
Volume 02 Issue 11 November-2026 | Impact Factor: 3.5



"Ghost Process Detection Tool": Analysing and Detecting Techniques of Files

Harini R
Computer Science Engineering
Akshaya Institute of Technology
harini.achar10@gmail.com

Lalithya R
Computer Science Engineering
Akshaya Institute of Technology
lalithyakeerthy7@gmail.com

Kaveri TO
Computer Science Engineering
Akshaya Institute of Technology
kaverito2004@gmail.com

Madhu MA
Computer Science Engineering
Akshaya Institute of Technology
madhumadhu63640@gmail.com

How to Cite this Article:
R, H., R, L., TO, K. & MA, M. (2026). "Ghost Process Detection Tool": Analysing and Detecting Techniques of Files. International Journal of Creative and Open Research in Engineering and Management, 02(11), 1-9. https://doi.org/10.55041/ijcope.v2i11.169

License:
This article is published under the terms of the Creative Commons Attribution 4.0 International License (CC BY 4.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original author(s) and the source are credited.
© The Author(s). Published by International Journal of Creative and Open Research in Engineering and Management.





https://doi.org/10.55041/ijcope.v2i11.169

***Abstract*—The Ghost Process Detection Tool (GhostGuard) is a Python/Streamlit security application that collects running-process metadata, applies transparent heuristic checks, and uses an unsupervised Isolation Forest model to flag unusual process behaviour. The system records the process ID, parent ID, CPU and memory usage, thread count, status, executable path and owner, and converts these into an explainable risk score with High, Medium and Low bands. It also supports search, filtering, a process detail panel and CSV export, and is strictly read-only: it never terminates processes or changes system settings. The risk score is a triage aid, not a probability of malware, and the tool cannot by itself prove kernel-level ghost processes or rootkits.**

***Index Terms*—ghost process, anomaly detection, Isolation Forest, risk scoring, process monitoring**

I. Introduction

Modern malware often hides inside legitimate-looking pro-cesses, runs from temporary directories, or consumes resources in ways that traditional signature-based antivirus does not capture. Analysts therefore need a quick, explainable way to triage which running processes deserve closer inspection.

This paper makes three contributions. The *front end* is a Streamlit interface that lets users inspect, search and filter processes. The *back end* is a Python layer that collects process metadata with `psutil`, applies rule-based checks and trains an anomaly model. The *integration* layer joins both into a single workflow that exports results as CSV for audit. The prototype runs on Windows 10/11, Linux and macOS, although some process fields may be unavailable depending on permissions.

II. Background

A. Problem and Motivation

A ghost process is one that is hidden, masquerading or otherwise difficult to attribute to a trusted program. Detecting such behaviour requires combining several weak signals, such as unusual paths, abnormal resource use and atypical parent–child relations, rather than relying on a single indicator.

B. Detecting and Related Work

Anomaly detection with Isolation Forest isolates outliers by random partitioning and needs no labelled malware data [4]. Research on process-memory and injection analysis [2], [3] shows that behavioural features complement static inspection. GhostGuard follows this direction but keeps a deliberately lightweight, read-only design.

III. Detection Techniques

A. Process Snapshot Collection

The tool takes a snapshot of all running processes and records PID, parent PID, CPU percentage, memory usage, thread count, status, executable path and user where permissions allow. A blank path can simply mean access was denied and is not treated as proof of a hidden process.

B. Rule-Based Indicators

Explainable rules add points to a process score, for example high CPU or memory use, or an executable located in a temporary or public directory. Because each rule is visible to the user, every flag can be justified.

C. Isolation Forest Anomaly Detection

When at least ten processes are available, an Isolation Forest learns a relative baseline from the current scan and assigns each process an anomaly score. A process can be unusual yet legitimate, so the model output is combined with, not substituted for, the rules.

D. Risk Scoring and Flagging

Rule points and the anomaly score are merged into a single *Risk Score* and mapped to High, Medium or Low bands. The score is a triage value, not a probability that the process is malware. CPU usage is a short snapshot and can be noisy, so several scans should be run before drawing conclusions.

E. Interface and Export

Users can search and filter the table, open a detail panel for any process and export the results as a CSV file. The application is read-only and makes no changes to the system.

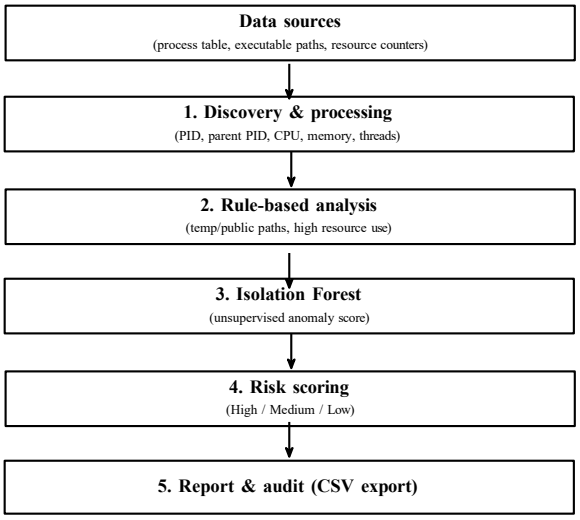

Fig. 1. Layered workflow for ghost process detection and analysis.

TABLE I
QUALITATIVE COMPARISON OF TECHNIQUES

Technique	Coverage	False Positives	Cost
Process snapshot	Structured metadata	Medium	Low
Rule-based checks	Known risky patterns	Medium	Low
Isolation Forest	Unusual behaviour	Medium to high	Medium
Combined risk score	Both signals	Low to medium	Low

IV. PROPOSED LAYERED FRAMEWORK

We combine the techniques above into five layers, shown in Fig. 1. Discovery collects the process table; rule-based analysis applies explainable checks; the Isolation Forest finds statistical outliers; risk scoring merges both signals; and a final reporting layer supports review and audit.

The layers compensate for each other's weaknesses. Rules give clear reasons but miss novel behaviour, while the anomaly model catches novelty but cannot explain itself. Together they reduce the chance that a suspicious process passes unnoticed.

V. RESULTS AND DISCUSSION

Table I compares the techniques qualitatively against the criteria most relevant to deployment. The ratings reflect general characteristics rather than measurements from a specific deployment, and the model should still be validated on a labelled, safe dataset.

The risk score acts as the analytical bridge between raw process data and actionable triage. By translating several indicators into one banded value, it lets an analyst prioritise which processes to review first. The tool does not independently prove kernel-level ghost processes or rootkits; stronger cross-view detection needs separate Windows-native inspection and careful validation. Testing should use normal applications such as a browser or text editor, never real malware.

Future work includes Authenticode signature verification, parent–child process tree visualisation, periodic polling with a user-controlled interval, persistent event history in SQLite, and model validation on labelled data.

VI. CONCLUSION

GhostGuard offers a streamlined, explainable approach to suspicious process triage by combining a live process snapshot, transparent rules, unsupervised anomaly detection and banded risk scoring in a read-only interface. It helps analysts and users quickly decide which processes deserve attention, while its documented limits make clear that the results are estimates and not proof of malicious activity.

REFERENCES

[1] E. McCallister, T. Grance and K. Scarfone, "Guide to Protecting the Confidentiality of Personally Identifiable Information (PII)," NIST Special Publication 800-122, 2010.
[2] A. Singh, R. A. Ikuesan and H. Venter, "Ransomware Detection using Process Memory," in *Proc. 17th Int. Conf. Cyber Warfare and Security (ICCWS)*, 2022, pp. 385–393.
[3] G. B. A. Lample, M. Ballesteros and S. Subramanian, "Process Injection and Memory Anomaly Detection in Host-Based Systems," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 1205–1218, 2021.
[4] M. G. Gaber, M. Ahmed and H. Janicke, "Malware Detection with Artificial Intelligence: A Systematic Literature Review," *ACM Computing Surveys*.