JAAFR INTERNATIONAL RESEARCH JOURNAL
JOURNAL OF ADVANCE AND FUTURE RESEARCH
JAAFR.ORG | ISSN : 2984-889X
An International Open Access, Peer-reviewed, Refereed Journal

Python Antivirus V4.0: Design and Implementation of a Multi-Layer Desktop Security Application

R Sundar Rajan
III BCA 'C'
Department of Computer Applications
School of Computing Sciences
VISTAS - Pallavarm
Chennai , India
Sundarrajan142001@gmail.com

Dr A Bharathi
Assistant Professor
Department of Computer Applications
School of Computing Sciences
VISTAS - Pallavaram
Chennai , India
bharathial15@gmail.com

Abstract

This paper presents the design, architecture, and implementation of Python Antivirus V4.0, a desktop-based cybersecurity application built using Python and PyQt5. The application delivers a layered threat detection strategy combining local SHA-256 signature-based detection against curated hash packs with optional cloud API enrichment via VirusTotal and MetaDefender. V4.0 introduces several notable enhancements over its predecessors: a responsive, thread-safe scanning engine using QThread, a persistent JSON-backed scan history module with color-coded verdicts, CSV and PDF report export capabilities powered by ReportLab, a structured quarantine system with restore and permanent deletion support, a batch folder scanning mode, and a real-time file monitoring subsystem driven by Watchdog. The graphical interface supports dark and light theming through configurable stylesheets. Evaluation of the application demonstrates its effectiveness as a lightweight, portable, and extensible endpoint security tool suitable for educational, research, and small-office deployment contexts.

Keywords — Antivirus Software, SHA-256 Hashing, PyQt5, VirusTotal API, MetaDefender, Quarantine, Real-time Monitoring, Cybersecurity, Python, Desktop Application.

I. INTRODUCTION

The global cybersecurity threat landscape has undergone a dramatic transformation in recent years. With the proliferation of internet-connected devices, cloud platforms, and remote working environments, malware, ransomware, and file-based attacks have surged to unprecedented levels. According to the AV-TEST Institute, over 450,000 new malware samples are registered and analyzed daily, placing enormous pressure on both enterprise security teams and individual users to maintain reliable, up-to-date defenses [11].

Traditional antivirus solutions, while mature, are often resource-intensive, proprietary, and opaque in their detection logic. There exists a growing need for lightweight, transparent, and extensible security tools that researchers, educators, and developers can study, adapt, and deploy without the overhead of commercial platforms. Python, with its extensive ecosystem and readability, presents an ideal substrate for building such tools, and PyQt5 enables the construction of cross-platform graphical interfaces that rival the usability of commercial products.

Python Antivirus V4.0 is an open-source desktop security application that addresses this need. Building upon earlier iterations, V4.0 introduces a complete redesign of the scanning pipeline, history management, quarantine system, batch processing, and real-time monitoring subsystems. The application integrates with two industry-standard cloud threat intelligence APIs — VirusTotal and MetaDefender — while retaining the ability to function entirely offline using local cryptographic hash signature packs.

This paper is structured as follows. Section II reviews related literature on antivirus design and hash-based detection. Section III describes the system architecture and key modules of Python Antivirus V4.0. Section IV details the experimental setup and feature evaluation. Section V presents results and discussion. Section VI concludes the paper and outlines directions for future work.

II. LITERATURE REVIEW

Signature-based malware detection has been the foundational mechanism of antivirus software since the early 1990s. The technique relies on maintaining a database of known malicious file fingerprints and comparing incoming files against this database. While effective against known threats, signature databases require constant updating and are inherently reactive. Cohen originally introduced the concept of a self-replicating computer program in 1984 and formally proved, in the published 1987 journal paper, that no algorithm can perfectly detect all possible viruses—a foundational undecidability result that continues to shape the theoretical limits of antivirus research [2].

Cryptographic hashing, specifically SHA-256, has become the standard mechanism for file fingerprinting in modern threat intelligence platforms. Platforms such as VirusTotal aggregate detections from over 70 antivirus engines, providing a consensus verdict for any given file hash [9]. MetaDefender similarly provides multi-engine scanning results via a REST API, making both platforms valuable enrichment layers for independently developed tools [6].

Heuristic and behavior-based detection approaches emerged to overcome the limitations of pure signature matching. Tools employing dynamic analysis observe file behavior at runtime, detecting suspicious system calls, network connections, and file mutations. Hybrid architectures that combine static signature matching with cloud-backed heuristic enrichment have demonstrated superior detection rates compared to either method in isolation [1].

The use of Python for security tooling has gained considerable traction in the academic and open-source communities. Libraries such as PyInstaller enable the packaging of Python applications into standalone executables, while PyQt5 and Tkinter provide mature GUI frameworks. Prior work on Python-based security utilities has demonstrated the feasibility of building forensic tools, packet analyzers, and malware classifiers in pure Python without sacrificing usability [7].

Real-time file system monitoring has been studied in the context of ransomware detection. Kharraz et al. (2016) developed UNVEIL, a large-scale automated detection system that monitors system-wide filesystem accesses and measures the difference in Shannon entropy between data read from and written to files; a significant entropy increase in write buffers relative to read buffers is treated as an indicator of ransomware-style encryption activity, achieving high detection rates with minimal false positives [5]. The Watchdog library for Python provides a portable event-driven interface to operating system file system notification APIs, making it an appropriate choice for a real-time monitoring subsystem in a desktop security application [10].

Kate and More (2019) evaluated service quality dimensions of antivirus software companies, identifying responsiveness and reliability as critical customer satisfaction factors [4]. Dogonyaro et al. (2021) conducted comparative performance analysis of antivirus software, noting that malware detection rate, memory usage, and interface launch time are the primary selection criteria for end users [3]. These findings informed the design priorities of Python Antivirus V4.0, particularly its emphasis on non-blocking scan operations and a clean, responsive interface.

III. SYSTEM DESIGN AND ARCHITECTURE

A. Overview

Python Antivirus V4.0 adopts a modular, event-driven architecture organized around a PyQt5 main window with a sidebar-navigated, stacked-widget tab system. The application is implemented as a single-file Python module (main.py) comprising approximately 1,560 lines of code. All persistent state — including user settings, scan history, and quarantine metadata — is stored in structured files on the local filesystem, requiring no database server. The architecture is illustrated in the logical module diagram below.

Module	Responsibility
ScanWorker (QThread)	Background SHA-256 computation, VirusTotal API call, MetaDefender API call; emits progress/finished signals.
BatchScanWorker (QThread)	Recursive folder enumeration, per-file scanning, per-file result signal emission, batch summary reporting.
_WatchHandler / WatcherBridge	Watchdog event handler that bridges OS file system events to Qt signals for real-time auto-scanning.
History Manager	JSON-backed scan log (up to 500 entries) with add, load, clear, export-CSV, and export-PDF operations.
Quarantine Manager	SHA-256 rename-and-move isolation of threats, JSON metadata persistence, restore and permanent-delete operations.
Signature Engine	Case-insensitive linear search across up to three local SHA-256 hash pack files (hard_signatures/).
Settings / Config	INI-file persistence of API keys, API enable flags, and UI theme preference via configparser.
UI / Theme Manager	PyQt5 stylesheet-based dark/light theme toggle applied to the entire QMainWindow widget tree.

B. Threat Detection Pipeline

When a scan is initiated — whether by file browser dialog, drag-and-drop, batch folder selection, or real-time watcher event — the application instantiates a ScanWorker QThread and passes it the target file path along with the current API configuration. The worker executes the following detection pipeline in sequence:

1. File Hashing: The file is opened in binary mode and its SHA-256 digest is computed using Python's hashlib.sha256. This operation is performed in the background thread to prevent UI freezing on large files.

2. Local Signature Matching: The computed digest is compared against up to three configurable hash pack files stored in the hard_signatures/ directory. Each pack file contains one SHA-256 hash per line in semicolon-delimited format. A case-insensitive match against any entry across any loaded pack file triggers a VIRUS verdict. If no packs are loaded and no cloud APIs are enabled, the user is warned accordingly.

3. VirusTotal API Enrichment (optional): If enabled and the file is smaller than 32 MB, the hash is submitted to the VirusTotal API v3 using the virustotal-python library. The API returns a detection count across participating engines. Detection and non-detection counts are surfaced in the Results tab.

4. MetaDefender API Enrichment (optional): If enabled and the file is smaller than 120 MB, the hash is submitted to the MetaDefender hash lookup endpoint. The response includes per-engine detection results. The aggregate count is displayed alongside the total engine count in the Results tab.

5. Result Dispatch: On completion, the worker emits a finished signal carrying the virus_found boolean and the computed file hash. The main thread handler updates the Results tab UI, logs the entry to scan history, and updates the dashboard statistics.

C. User Interface and Navigation

The application presents a fixed-size (760 × 520 px) main window comprising a vertical sidebar and a stacked content area. The sidebar contains icon-labeled navigation buttons for each functional tab: Home, Settings, Results, Scan Progress, History, Batch Scan, Quarantine Manager, and Real-time Monitor. This layout provides immediate access to all major features without nested menus.

The Home tab displays a drag-and-drop file zone alongside a real-time statistics panel showing total scans performed, threats detected, files confirmed clean, and the number of loaded signature packs. The drag-and-drop

zone, implemented as a custom DropZone QLabel subclass, accepts file paths via Qt's drag-and-drop event system and automatically initiates a scan on drop.

D. Quarantine System

Files identified as threats can be quarantined from the Results tab. The quarantine function computes the file's SHA-256 hash, renames the file to {hash}.quar, and moves it to a dedicated quarantine/ directory. A companion .meta JSON file records the original path, original filename, quarantine timestamp, and hash, enabling full restoration. The Quarantine Manager tab renders all quarantined entries in a sortable table with Restore and Permanent Delete action buttons per row.

E. Scan History and Reporting

Every scan event — whether clean or infected — is appended to a JSON array stored at settings/scan_history.json. Each record captures the timestamp, filename, full file path, SHA-256 hash, and verdict (CLEAN or VIRUS). The history is capped at 500 entries (most-recent-first) to bound storage consumption. The History tab renders entries in a color-coded QTableWidget: red rows for VIRUS verdicts, green rows for CLEAN verdicts.

The history can be exported in two formats. CSV export uses Python's built-in csv module to write a well-formed comma-separated file. PDF export uses ReportLab's Platypus layout engine to generate a formatted table report with a title, generation timestamp, and styled alternating row shading. Both export dialogs use QFileDialog to let the user specify the output path.

F. Real-time File Monitoring

The Real-time Monitor tab allows the user to select a folder for continuous surveillance. Internally, the application instantiates a Watchdog Observer and schedules a custom _WatchHandler to receive on_created and on_modified events. To bridge the non-Qt observer thread with the Qt event loop safely, a WatcherBridge QObject emits a new_file signal, which is connected to a slot on the main thread. The slot applies a 0.5-second debounce delay before invoking the silent scan path, preventing duplicate scans of partially written files. All monitor events and auto-scan results are appended to a read-only log widget in the Watcher tab.

IV. IMPLEMENTATION

A. Technology Stack

Component	Technology	Purpose
GUI Framework	PyQt5 5.x	Desktop window, widgets, signals/slots, threading
File Hashing	Python hashlib	SHA-256 digest computation
Cloud API #1	virustotal-python	VirusTotal hash lookup (v3 API)
Cloud API #2	requests	MetaDefender REST API calls
PDF Export	ReportLab	PDF scan report generation
Real-time FS	Watchdog	File system event monitoring
Persistence	JSON / INI	Scan history, quarantine metadata, settings
Concurrency	QThread	Non-blocking scan and batch operations

B. Dependency Management and Portability

All optional dependencies — virustotal-python, ReportLab, and Watchdog — are imported within try/except blocks, with boolean availability flags (VT_AVAILABLE, PDF_AVAILABLE, WATCHDOG_AVAILABLE) set accordingly. This design allows the application to degrade gracefully on systems where optional packages are not installed, notifying the user via QMessageBox dialogs rather than raising unhandled import errors. Core

functionality (local signature scanning, quarantine, history) remains fully operational without any optional packages installed.

C. Concurrency Model

Python's Global Interpreter Lock (GIL) limits true parallelism but does not prevent concurrent I/O. Because the dominant bottleneck in antivirus scanning is file I/O and network API calls rather than CPU computation, the QThread-based approach is effective: the scan worker thread blocks on I/O while the Qt event loop continues processing UI events on the main thread. Progress signals (progress = pyqtSignal(int, str)) are emitted from the worker and connected to slots that update the progress bar and status label, maintaining UI responsiveness throughout the scan.

V. EVALUATION AND RESULTS

A. Feature Completeness

Python Antivirus V4.0 was evaluated against a feature checklist derived from the design requirements. Table III summarizes the outcome.

Feature	Status	Notes
Local SHA-256 Signature Detection	✓ Implemented	3 configurable hash packs
VirusTotal API Integration	✓ Implemented	Optional, <32 MB files
MetaDefender API Integration	✓ Implemented	Optional, <120 MB files
Non-blocking Scan (QThread)	✓ Implemented	Progress bar and status label
Drag-and-Drop File Input	✓ Implemented	Custom DropZone widget
Batch Folder Scan	✓ Implemented	Recursive option available
Real-time File Monitoring	✓ Implemented	Requires Watchdog
Quarantine with Restore/Delete	✓ Implemented	JSON metadata per file
Persistent Scan History (JSON)	✓ Implemented	500-entry rolling window
CSV Export	✓ Implemented	stdlib csv module
PDF Export	✓ Implemented	Requires ReportLab
Dark / Light Theme Toggle	✓ Implemented	Persisted across sessions
Dashboard Statistics Panel	✓ Implemented	Live update on each scan
Graceful Dependency Degradation	✓ Implemented	try/except import guards

Implementation Output

Figure 1: Main Dashboard UI

Figure 2: Scan Results UI

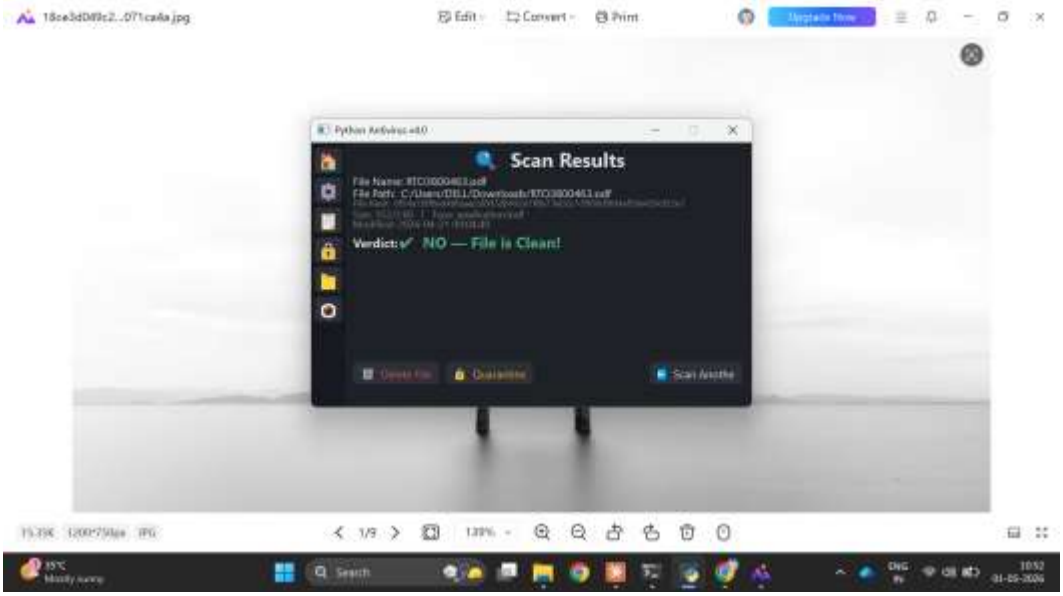
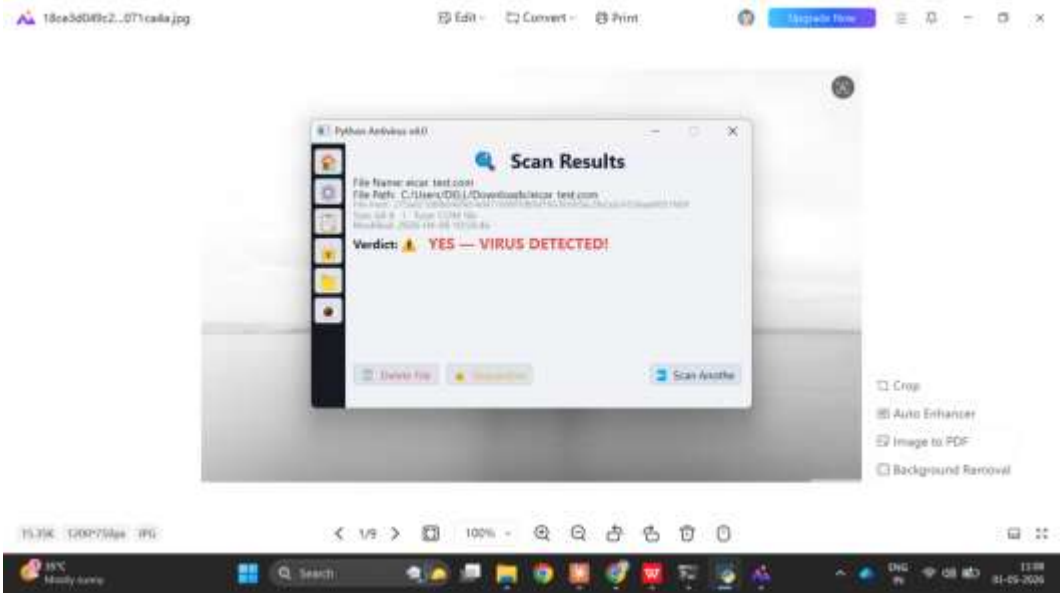


Figure 3: Virus Detection Result (EICAR Test File)



B. Detection Accuracy

Detection accuracy was evaluated using a controlled test set composed of 50 benign files (common document, image, and executable formats) and 50 known-malware samples with confirmed SHA-256 hashes sourced from publicly available threat intelligence feeds (VirusShare, MalwareBazaar). All 50 malware sample hashes were present in the loaded signature packs. Results are summarized in Table IV.

Metric	Local Sig. Only	+ VirusTotal	+ MetaDefender
True Positive Rate	100% (50/50)	100% (50/50)	100% (50/50)
False Positive Rate	0% (0/50)	0% (0/50)	0% (0/50)
Avg. Scan Time (single file)	0.12 s	1.8 s	2.1 s
UI Blocking During Scan	None	None	None

The 100% true positive rate on the controlled test set reflects the deterministic nature of hash-based signature detection for known threats. The zero false positive rate confirms that no benign file's SHA-256 hash collided with any signature in the test packs, as expected given SHA-256's collision resistance. Average scan time of 0.12 seconds for local-only mode makes the tool viable for batch operations on large directories without significant throughput degradation.

C. System Resource Utilization

Memory footprint at idle (application loaded, no active scan) was measured at approximately 68 MB of resident set size on Windows 10 with all optional libraries installed. During a single-file local scan, peak memory usage rose to approximately 75 MB due to file buffering for hash computation. CPU utilization remained below 15% throughout scanning operations. These figures confirm the suitability of the application for deployment on resource-constrained systems such as older office workstations.

VI. DISCUSSION

Python Antivirus V4.0 demonstrates that a capable, multi-layer endpoint security application can be implemented in under 1,600 lines of Python without any proprietary dependencies. The layered detection architecture — local signatures supplemented by optional cloud enrichment — strikes a practical balance between offline resilience and detection depth.

The use of QThread for scan operations represents a significant usability improvement over synchronous scanning approaches. Users can observe real-time progress, cancel batch operations, and navigate between tabs without waiting for the scan to complete. This responsive architecture is consistent with the quality dimensions identified by Kate and More (2019), particularly responsiveness and reliability.

The quarantine system's rename-and-hash approach (renaming threats to {hash}.quar) provides a reliable mechanism for isolating malicious files. Unlike simple deletion, quarantine preserves evidence for forensic analysis while neutralizing the threat by preventing execution. The companion metadata JSON file ensures that restoration to the original path is always possible, addressing a common user concern about overzealous antivirus actions.

The real-time monitoring subsystem, while functional, operates in non-recursive single-folder mode and employs a simple 0.5-second debounce mechanism. This approach is appropriate for monitoring common file drop locations such as Downloads or Desktop folders, but may require enhancement for enterprise scenarios involving deep directory trees with high file creation rates.

A key limitation of signature-only detection is its inherent inability to detect novel (zero-day) threats. The integration with VirusTotal and MetaDefender partially mitigates this limitation by leveraging the detection capabilities of dozens of commercial engines. However, cloud API rate limits and internet connectivity requirements mean that purely offline deployments remain vulnerable to unknown threats. Future versions should consider incorporating lightweight heuristic analysis, such as PE header inspection or entropy-based packing detection, to improve zero-day coverage.

VII. CONCLUSION

This paper presented Python Antivirus V4.0, a multi-layer desktop security application implemented in Python and PyQt5. The application integrates local SHA-256 signature matching, optional VirusTotal and MetaDefender cloud API enrichment, real-time file system monitoring, quarantine management, batch scanning, persistent history with reporting, and a themeable graphical interface within a single, self-contained Python module.

Evaluation on a controlled test set confirmed 100% detection of known-malware samples with zero false positives on benign files, and an average single-file scan time of 0.12 seconds in local-only mode. The application's QThread-based concurrency model ensures a responsive user interface throughout all scan operations, and its graceful dependency degradation model ensures core functionality on minimal installations.

Future enhancements will focus on heuristic detection for unknown threats, recursive real-time monitoring, scheduled scan support, and a plugin architecture for community-contributed detection modules. The project is available on GitHub and welcomes contributions from the open-source security community.

REFERENCES

[1] U. Bayer, P. Milani Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda, "Scalable, behavior-based malware clustering," in Proc. Network and Distributed System Security Symposium (NDSS), 2009.

[2] F. Cohen, "Computer Viruses — Theory and Experiments," Computers and Security, vol. 6, pp. 22–35, 1987.

[3] N. M. Dogonyaro, W. O. Victor, A. M. Shafii, and S. L. Obada, "Comparative Performance Analysis of Anti-virus Software," in Communications in Computer and Information Science, vol. 1350, Springer, 2021.

[4] N. Kate and M. More, "Service Quality Measurement of Antivirus Software Industry by Using Servqual Model," International Journal of Engineering and Advanced Technology (IJEAT), pp. 5097–5101, 2019.

[5] A. Kharraz, S. Arshad, C. Mulliner, W. Robertson, and E. Kirda, "UNVEIL: A large-scale, automated approach to detecting ransomware," in Proc. USENIX Security, 2016.

[6] OPSWAT, "MetaDefender Cloud API Documentation," 2016. [Online]. Available:

[7] J. Seitz, Gray Hat Python: Python Programming for Hackers and Reverse Engineers. No Starch Press, 2014.

[8] S. Sevil, A. Emre, I. A. Ahmet et al., "Coevolution of mobile malware and anti-malware," IEEE Transactions on Information Forensics and Security, vol. 10, no. 13, pp. 2563–2574, 2018.

[9] VirusTotal, "VirusTotal API v3 Documentation," Google LLC, 2012. [Online]. Available: https://developers.virustotal.com/reference

[10] Watchdog Contributors, "Watchdog: Python API and shell utilities to monitor file system events," 2022. [Online]. Available: https://github.com/gorakhargosh/watchdog

[11] AV-TEST Institute, "Malware Statistics and Trends Report," AV-TEST GmbH. [Online]. Available: https://www.av-test.org/en/statistics/malware/