J-DA
JOURNAL
DEKSTOP APPLICATION

Received: 18 Oct 2025 | Revised: 4 Nov 2025 | Accepted: 1 Des 2025 | Published: 12 Des 2025 E-ISSN: 2961-9688, P-ISSN: 2961-9696

DOI: 10.59431/jda.v4i2.665

RESEARCH ARTICLE

Simple Antivirus Scanner: A Signature-Based Malware Detection System Using Delphi

Antonov Alin [1*]

[1] SourceCodester Community, Romania.

Correspondence

[1*] SourceCodester Community, Romania.
Email: antonov.alin@sourcecodester.com

Funding information
SourceCodester Community, Romania.

Abstract

The *Simple Antivirus Scanner* was developed as an instructional model to demonstrate the core mechanisms of signature-based malware detection. Implemented using Delphi, the system integrates MD5 hashing, database-driven signature comparison, and asynchronous scanning through the TBackgroundWorker component, providing both functionality and responsiveness in a Windows environment. The project's architecture includes recursive file traversal, efficient hash computation, and a structured virus definition database that enables accurate identification of known malware. The inclusion of a harmless test virus allows for safe experimentation and validation of the detection process, reinforcing user understanding of hash-based recognition. Results show that the system performs effectively in detecting catalogued threats, offering a predictable and transparent learning experience. However, it lacks the capabilities of modern antivirus systems such as heuristic analysis, real-time protection, and automated signature updates. As a pedagogical platform, it serves as a bridge between theory and practice—illustrating file system operations, cryptographic applications, and data-driven threat identification. Future development may expand the scanner into a more comprehensive framework incorporating multi-layered detection, cloud-based updates, and AI-based classification. Ultimately, the project emphasizes clarity and accessibility, making it a valuable resource for students, educators, and cybersecurity enthusiasts seeking practical insight into antivirus design principles and malware detection logic.

Keywords
Antivirus; Malware Detection; Md5 Hashing; Delphi Programming; Cyber Security Education.

1 | INTRODUCTION

The evolution of malicious software since the advent of personal computing underscores that defensive mechanisms must continuously advance alongside offensive innovation. Early antivirus programs based on straightforward signature matching established the foundation for modern security architectures that now integrate heuristic analysis, behavioral monitoring, machine learning, and cloud-based threat intelligence (Szor, 2005; Aycock, 2006; Sikorski & Honig, 2012). Despite significant technological progress, signature-based detection remains a practical and widely adopted method for identifying known malware due to its speed, deterministic outcomes, and low false-positive rate. Its effectiveness,

however, decreases when adversaries employ packing, polymorphism, or targeted code alterations that modify a binary's fingerprint and bypass traditional matching (Bonfante *et al*., 2008; Gandotra *et al*., 2014; Gaudesi *et al*., 2015). The Simple Antivirus Scanner project demonstrates these foundational detection principles through an educational implementation emphasizing file traversal, hash computation, and database comparison. Such a framework enables students and researchers to observe the internal workings of antivirus engines and to recognize the limits of static detection in real-world environments (Du *et al*., 2019; Surendran & Thomas, 2022; Mat *et al*., 2021). Integrating dynamic analysis—where file behavior is monitored during execution—further enhances detection by collecting API traces and identifying behavioral anomalies, although this extension introduces greater system complexity and higher potential for false positives (Darem *et al*., 2021; Wagener *et al*., 2007; Santos *et al*., 2011).

The choice of hashing algorithms, including MD5, SHA-1, SHA-256, and TLSH, affects both performance and resistance to collision attacks. Existing studies highlight trade-offs between computational efficiency and cryptographic strength while documenting vulnerabilities in outdated digest methods (RFC 1321; Ali & Farhan, 2020; Boyanov, 2024; Kessler, 2016; Wang *et al*., 2010; Nkouankou *et al*., 2022; Oliver *et al*., 2013). From an implementation standpoint, using programming environments capable of producing native code and supporting asynchronous operations ensures responsive and scalable scanning across large file systems (Delphi Documentation; TBackgroundWorker Component Documentation). Although not intended as a production-grade security solution comparable to commercial suites that employ extensive signature repositories, cloud updates, sandboxing, and telemetry, the Simple Antivirus Scanner remains pedagogically valuable. It allows learners to examine the algorithmic decisions, architectural trade-offs, and performance considerations that underpin contemporary antivirus software, bridging theoretical understanding with practical application (Signature-Based Detection in Antivirus Software, 2015; Souri & Hosseini, 2018; Yerima *et al*., 2015; Kim *et al*., 2018; Seifi & Parsa, 2018).

2 | SYSTEM ARCHITECTURE AND DESIGN

The Simple Antivirus Scanner, developed using *Delphi*, is built upon a robust architectural foundation designed to demonstrate the principles of signature-based malware detection. Delphi, a rapid application development (RAD) environment optimized for Windows desktop software, provides extensive access to native Windows APIs, a rich component library for user interface construction, and reliable tools for file system operations. Its ability to produce compiled native code contributes significantly to performance efficiency, enabling fast file scanning, direct communication with system-level resources, and effective memory management. These characteristics make Delphi particularly suitable for antivirus software development, where speed and stability are critical. The system also benefits from Delphi's integrated cryptographic libraries, which simplify the implementation of hash-based verification essential for detecting malicious files. A central aspect of the Simple Antivirus Scanner's architecture is the use of the TBackgroundWorker component from DelphiArea. This component enables asynchronous scanning operations that prevent the application interface from freezing during extensive scans, ensuring a smooth user experience. It supports scan progress reporting, cancellation of ongoing tasks, and interface updates in real time. The scanner employs the MD5 hashing algorithm, implemented through Delphi's cryptographic tools or compatible third-party libraries, to generate unique digital fingerprints for scanned files. To manage and store virus definitions, the system utilizes a lightweight file-based or embedded database optimized for quick lookups and simple updates, ensuring that new malware signatures can be added with minimal overhead.

Like most signature-based systems, the scanner relies on a continuously updated signature database to maintain accuracy and relevance. Its primary advantage lies in its ability to detect known threats quickly and reliably through pattern matching. However, it remains vulnerable to new or modified malware capable of altering their digital signatures to avoid detection. This weakness has been widely acknowledged in malware research, emphasizing the necessity of complementary detection methods (Gandotra *et al*., 2014; Gaudesi *et al*., 2015; Seifi & Parsa, 2018). Empirical studies show that while signature-based approaches are efficient for catalogued malware, they perform poorly when dealing with emerging, previously unseen samples (Yerima *et al*., 2015; Souri & Hosseini, 2018). As a result, commercial antivirus solutions typically integrate heuristic and behavioral analysis to detect anomalies beyond known signature patterns (Souri & Hosseini, 2018). The Simple Antivirus Scanner, while intentionally educational in scope, illustrates this balance between simplicity and functionality and reinforces the importance of advancing detection methodologies in response to the continual evolution of malware strategies (Santos *et al*., 2011). By combining Delphi's technical capabilities with a structured database approach, the scanner demonstrates how accessible tools can effectively be used to construct an instructional model of malware detection. The system's internal structure consists of several interconnected components designed to work harmoniously. At its core is the Main Scanner Engine, responsible for recursively traversing directories, filtering files by their extensions, calculating MD5 hash values, determining file sizes, comparing those results to the database of known malware, and flagging infected files. This structured workflow ensures efficient and accurate analysis

of diverse file types across multiple directories. The choice of MD5 as the hashing function—despite its well-documented weaknesses—reflects its continuing practicality for fast identification in controlled or educational environments (Boyanov, 2024). Although MD5 is vulnerable to collision attacks and should not be used in high-security contexts, it remains effective for integrity checks where cryptographic strength is not the primary concern (Kessler, 2016; Oliver *et al*., 2013). Nevertheless, any production-level antivirus system should employ stronger algorithms and additional validation layers to mitigate these risks (Wang *et al*., 2010; Nkouankou *et al*., 2022).

To support database creation and management, the system incorporates a Database Maker Utility, which allows users to create new virus definition files, add or modify existing signatures, and verify database consistency. This module simplifies the process of maintaining up-to-date virus definitions and supports importing and exporting databases for collaborative or educational use (Ali & Farhan, 2020). The application also includes a well-designed user interface intended to provide clear control over the scanning process. Users can select files or folders, configure filtering parameters such as file extensions or scan depth, and observe real-time feedback on scanning progress. Detected threats are displayed immediately, allowing users to interpret results without technical difficulty. This attention to interface design enhances usability and ensures that the scanner is accessible to users with varying technical backgrounds (Wang & Yu, 2005; Wagener *et al*., 2007). An additional component of the system is the Test Virus, a harmless console application that displays the text "This is a fake virus." This demonstration file enables users to test the scanner's detection capability safely, providing a controlled means of verifying functionality without exposing the system to actual threats (Wagener *et al*., 2007). Collectively, these components create a cohesive and functional environment that blends theoretical instruction with practical experimentation. By aligning technical functionality with educational clarity, the Simple Antivirus Scanner functions as both a learning platform and a working prototype for understanding malware detection mechanisms.

At the heart of the scanner's operation lies the virus definition database, which stores essential attributes for identifying malicious files. Each record contains a unique identifier or name, the file's MD5 hash value, file size, optional extension, threat category (such as virus, trojan, or worm), severity level, and date of creation or detection. The database structure is designed for fast access through indexing of the hash field and uses file size as a preliminary filter to reduce unnecessary comparisons. This minimalist yet efficient structure supports quick updates, easy maintenance, and low storage overhead, making it suitable for instructional purposes. Although simplified compared to enterprise-grade antivirus systems, this schema captures the fundamental logic of signature-based detection used in professional cybersecurity tools, where additional metadata—such as hex patterns, behavioral cues, and distribution data—may also be included. The scanner's algorithmic workflow follows a sequential and deterministic process. During initialization, the program loads the virus definition database into memory, applies user configurations such as target directories and file extensions, and initializes counters for progress tracking. The directory traversal phase then systematically scans all subfolders, handling access permissions, hidden files, and potential errors without interrupting the overall process. For each accessible file, the algorithm calculates its MD5 hash and records the file size before comparing these attributes against entries in the virus database. A matching pair of hash and size values indicates infection, prompting the system to record details such as file path, detected signature, and timestamp. Once the scan is complete, the system compiles results, logs detected threats, generates summary statistics, and releases allocated resources. This organized procedure ensures predictable performance, transparency in results, and reliable detection outcomes—achieving the project's goal of demonstrating core antivirus functionalities through a clear and effective software model.

3 | CORE FUNCTIONALITIES

The Simple Antivirus Scanner integrates several core functions that work together to perform malware detection based on signature comparison. These core components include the file system scanning mechanism, MD5 hash computation, signature matching process, database management tools, and results reporting features. Collectively, these functions form a cohesive workflow that demonstrates how traditional antivirus systems identify and manage potential threats in a controlled, educational environment. The file system scanning module implements a recursive traversal algorithm that begins from a user-selected directory or drive and systematically scans all subdirectories and files. Using Delphi's native functions—*FindFirst*, *FindNext*, and *FindClose*—the scanner enumerates each directory's contents, processes files, and ensures that symbolic links and junction points are handled correctly to prevent infinite recursion. Robust error handling mechanisms manage common file access issues, such as permission restrictions, locked files, or excessive path lengths, without interrupting the overall scanning process. To maintain responsiveness during lengthy scans, the system employs asynchronous execution through the *TBackgroundWorker* component, which allows the user interface to remain active while scans run in the background. Additional optimizations include buffered file reading for hash calculation, optional parallel directory processing for multi-core systems, and early scan termination upon user request. During the scanning process, the application provides real-time feedback on the currently scanned file, the number of files processed and remaining, completion percentage, estimated time left, and total threats detected. This level of transparency builds user trust and provides a clear understanding of scanning progress and performance. The MD5

hash calculation module serves as the backbone of the signature-based detection process. The Message Digest Algorithm 5 (MD5) generates a 128-bit hash value, typically represented as a 32-character hexadecimal string. Although MD5 is no longer considered secure for cryptographic use due to known collision vulnerabilities, it remains appropriate for file identification in antivirus systems, where the primary goal is pattern recognition rather than encryption security. The hash calculation operates by opening each file in read-only mode, reading its contents in small, fixed-size blocks (commonly 64 KB) to conserve memory, and processing each block incrementally before generating the final hash value. This streaming-based approach allows the scanner to handle files of any size efficiently without loading them entirely into memory. For educational purposes, MD5 provides an ideal balance between performance, accuracy, and simplicity. However, other algorithms such as SHA-1 or SHA-256 could be substituted to improve resistance to collisions at the cost of additional processing time.

The signature matching process is the central operation of malware detection in the Simple Antivirus Scanner. Once a file's MD5 hash and size have been calculated, the system compares them against entries in the virus definition database. The matching algorithm uses indexed search methods to quickly identify database records with identical hash values, achieving logarithmic or constant-time performance depending on database structure. When a potential match is found, a secondary check verifies that the file size corresponds with the stored signature data to minimize false positives. If both conditions are satisfied, the system flags the file as infected, logs the detection details, and displays the threat information to the user through the results interface. Files that do not match any known signatures are marked as clean, allowing the scan to continue seamlessly. The detection process can be extended through advanced verification techniques such as file extension validation, hexadecimal pattern comparison, file header inspection, or metadata correlation. These methods can further reduce false positives caused by coincidental hash matches but generally increase computational complexity and scanning duration. The database management utility functions as an essential administrative tool for maintaining the virus definition repository. Through this module, users can create new signature databases or modify existing ones by adding malware entries derived from known samples. Each entry records critical attributes including the file's MD5 hash, size, classification, and severity level. The utility supports both automatic and manual entry modes, enabling users to import data directly from analyzed files or enter information manually from external reports. Database maintenance operations include editing or deleting outdated records, searching and filtering signatures by name or type, and verifying database consistency. Import and export features allow the integration of multiple databases or the backup of existing ones in formats such as CSV or XML. Depending on user needs, the system supports several database structures—text-based for readability and manual editing, binary for compactness and speed, or embedded formats like SQLite or Firebird for more complex queries. The design prioritizes portability, ease of maintenance, and adaptability to future extensions. Finally, the results reporting module provides comprehensive feedback once the scanning process concludes. The application compiles a summary including the total number of files scanned, directories traversed, threats detected, clean files, total scan duration, and average scanning speed measured in files per second. A detailed list of identified threats is presented, containing the file path, detected malware name, file size, MD5 hash, detection timestamp, and suggested user action. The interface enables sorting and filtering of results by various criteria such as threat name, file path, or severity, as well as keyword-based searching for specific entries. Results can be exported to text or CSV files for record-keeping or analysis. In addition, the user can view file properties, locate files directly within their directories, and manually delete or quarantine infected files if supported. This thorough reporting process enhances user understanding of detection outcomes and reinforces the educational objective of the system by illustrating each stage of the antivirus workflow—from scanning to detection and remediation. In summary, the Simple Antivirus Scanner integrates these core functionalities into a coherent and instructive system that accurately models the operational logic of real antivirus engines. By combining practical efficiency with pedagogical clarity, the application not only demonstrates the fundamentals of signature-based detection but also provides an accessible platform for studying and extending malware analysis techniques.

4 | RESULTS AND DISCUSSION

4.1 Results

4.1.1 Implementation Details

The Simple Antivirus Scanner was developed using Delphi, a rapid application development (RAD) environment designed for Windows desktop applications. Delphi provides high performance for system-level operations, direct access to Windows APIs, a rich component library for user interface design, and efficient file system handling. Its ability to produce compiled native code enables rapid file scanning and precise control over memory and cryptographic processes, making it suitable for antivirus applications. The user interface was designed to be simple and functional, consistent with the project's educational purpose. The main window includes a scan configuration panel that allows users to select drives or folders through a browse dialog, specify file extension filters, and define scan depth options—either scanning the current directory only or including subdirectories.

Control buttons such as *Start Scan* and *Stop/Cancel* manage scanning operations, while a progress bar visually indicates the scan's completion status. A results area lists scanned files, highlights detected threats using red text or warning icons, and displays key details such as file path, size, hash value, and scan status. A real-time status bar presents ongoing statistics, including the total number of files scanned, threats detected, and elapsed time. A separate section manages the virus definition database, allowing users to open, edit, or update signatures and perform import or export operations. The interface prioritizes clarity, usability, and accessibility, requiring no advanced technical knowledge to operate effectively.

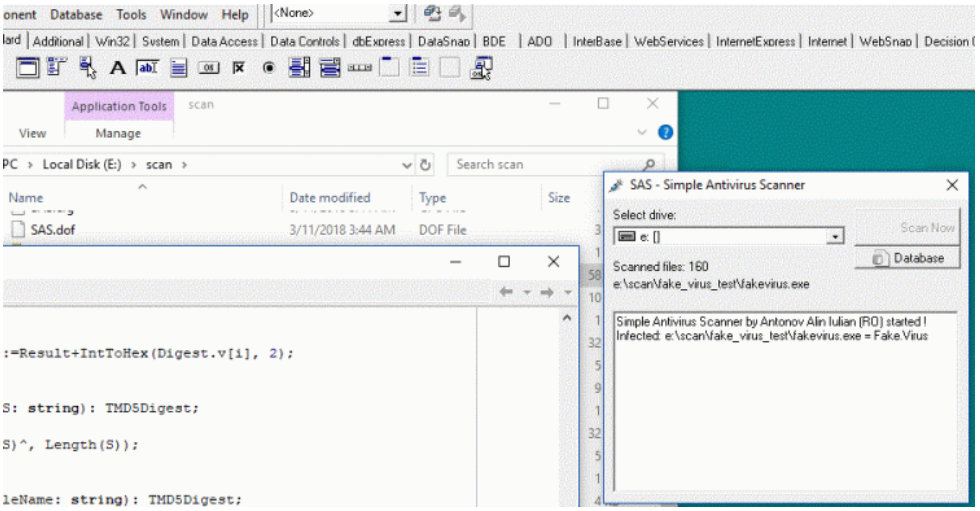

Figure 1. Application Results Interface

To ensure responsive operation during long scanning processes, the application implements asynchronous execution using Delphi's TBackgroundWorker component. This component handles background scanning in a separate thread, allowing the user interface to remain interactive. The scanning logic is executed in the *DoWork* event, progress updates are handled in *ProgressChanged*, and completion or cancellation is processed in *RunWorkerCompleted*. This architecture prevents the application from freezing and ensures safe communication between threads. Error handling in the background thread captures exceptions and reports them through the main interface, preventing system crashes and maintaining stability throughout the operation. The MD5 hash calculation in Delphi can be implemented through several options: the *TIdHashMessageDigest5* class from the Indy components, the *DCPcrypt* library, or the *THashMD5* class from the *System.Hash* unit in newer versions of Delphi. The implementation involves creating an MD5 object, opening the target file in read-only mode, reading its content in 64 KB chunks, updating the hash with each block, and converting the final hash into a 32-character hexadecimal string. This process enables efficient handling of large files without excessive memory usage while providing reliable identification of known malware signatures. Although MD5 is no longer considered cryptographically secure due to potential collisions, it remains practical for matching known malware within an educational context.

The virus definition database may be implemented using several approaches depending on performance and scalability needs. Simple text-based formats such as CSV allow easy editing but limited search efficiency. XML formats provide structured and validated data but result in larger file sizes and slower parsing. INI files offer simplicity but limited structure, while embedded databases such as SQLite or Firebird provide faster indexed lookups, transactional integrity, and scalability at the cost of complexity. For this project, a text or XML format provides an appropriate balance between simplicity and functionality. The test virus component serves as a safe, educational tool for verifying the scanner's performance. This harmless console program displays the text *"This is a fake virus"* when executed. It has no malicious functionality, system impact, or persistence mechanisms, ensuring a safe testing environment. Its MD5 hash and file size are included in the virus definition database so that the scanner can detect it during scans. This component provides valuable educational insight. It allows users to observe how signature matching identifies files, validate that detection functions correctly, and experiment safely with modifying the file to see how even small changes alter the signature. Students can thus understand the necessity of frequent signature updates and how detection relies on static file properties.

4.1.2 Advantages and Limitations

The Simple Antivirus Scanner has notable advantages for teaching and demonstration. Its codebase clearly illustrates signature-based detection concepts, cryptographic hashing, file traversal, and database comparison. It operates efficiently with minimal system resources, delivers accurate results for known threats, and offers

predictable and reproducible outcomes. Its simplicity and transparency make it ideal for instructional use, allowing students to easily follow and modify the logic. However, the system also presents significant limitations. It cannot detect unknown or zero-day malware, lacks heuristic and behavioral analysis, and does not provide real-time protection or automated updates. The database must be manually maintained, and the system offers no quarantine or removal capabilities. Performance may degrade when scanning large drives sequentially, and security measures such as tamper protection or secure database verification are absent. These constraints are acceptable for an educational prototype but highlight the complexity of professional antivirus systems that incorporate multi-layered detection, cloud updates, and machine learning.

4.2 Discussion

The Simple Antivirus Scanner presents several possibilities for enhancement that could substantially extend its functionality and educational value. One potential improvement is the inclusion of additional hash algorithms such as SHA-1 or SHA-256, which would strengthen detection accuracy and reduce the likelihood of false positives resulting from MD5 collisions. Incorporating multiple hashing methods would provide greater assurance of uniqueness across files, particularly as malware samples grow more sophisticated in evasion techniques. Another enhancement involves implementing hex pattern matching to identify malware through specific byte sequences or patterns within a file. This feature would enable the detection of polymorphic or metamorphic malware variants that maintain core code similarities while altering surface characteristics to evade simple hash-based detection. Furthermore, integrating heuristic analysis—both static and dynamic—would improve the scanner's capability to identify suspicious files or behaviors even when a direct signature match is unavailable. Static heuristics could evaluate file structure and embedded API calls, while dynamic heuristics could observe runtime behaviors in a sandboxed environment, providing more adaptive and resilient detection. Beyond these core improvements, the scanner could evolve further through the addition of real-time protection and automatic signature updates. Real-time protection would involve the integration of a file system filter driver capable of monitoring and intercepting file operations as they occur, thereby preventing the execution or modification of infected files. Meanwhile, an automatic update mechanism could be developed to connect securely to an online repository for retrieving the latest signature definitions. Together, these enhancements would transform the Simple Antivirus Scanner from a static educational tool into a more dynamic and responsive security solution. However, these advancements would also introduce increased development complexity, higher system overhead, and greater testing requirements to ensure stability and compatibility with modern operating systems.

As an educational platform, the Simple Antivirus Scanner offers significant pedagogical benefits across multiple fields of computer science and cybersecurity education. It provides students with hands-on experience in applying theoretical concepts to practical implementations, covering essential areas such as file system operations, cryptographic hashing, data management, and multithreaded programming. The project bridges the gap between theory and practice by demonstrating how abstract security models are realized in functioning software systems. In programming and software development courses, the scanner serves as an accessible yet realistic project that helps students understand component integration, modular coding practices, and user interface design. Within cybersecurity curricula, it enables safe experimentation with malware detection and signature-based analysis without exposure to real threats. Similarly, in software engineering education, it can be used to teach design architecture, maintenance, error handling, and version control. For capstone or research projects, students may extend the system by implementing heuristic detection, machine learning-based classification, or cloud-assisted signature management, encouraging innovation grounded in foundational understanding. When compared to production-grade antivirus solutions, the Simple Antivirus Scanner reflects the fundamental concepts upon which modern protection systems are built. Contemporary antivirus products such as Norton, McAfee, Kaspersky, Bitdefender, and Windows Defender employ layered detection strategies that combine traditional signature matching with behavioral monitoring, sandbox-based analysis, and artificial intelligence. These systems operate on global infrastructures, utilizing cloud-based databases and automated updates to respond rapidly to emerging threats. While the Simple Antivirus Scanner focuses solely on static, signature-based detection, its simplicity underscores the underlying mechanics that still form a critical layer of modern antivirus engines.

Open-source antivirus projects such as ClamAV and Comodo Antivirus offer a closer comparison in terms of design philosophy. Like the Simple Antivirus Scanner, they rely heavily on signature-based matching but are implemented at a much larger scale, incorporating automated updates, platform independence, and integration with email and file servers. Despite its smaller scope, the Simple Antivirus Scanner effectively mirrors the basic operational logic of these open-source systems, providing an instructive model for understanding their internal structures and evolution. Historically, antivirus technology has advanced through several distinct generations—beginning with early scanners based solely on static signatures, progressing to heuristic and behavioral analysis, followed by the adoption of cloud-based intelligence, and culminating in current approaches that leverage artificial intelligence and big data analytics. Within this evolutionary framework, the Simple Antivirus Scanner represents a

first-generation model that remains essential for grasping how contemporary detection methods developed from earlier technologies.

In addition to technical considerations, security and ethical awareness are critical aspects of developing and using antivirus software. When conducting experiments or extending this project, users must adhere to strict safety procedures to prevent accidental infection or system compromise. Real malware samples should only be analyzed within isolated, sandboxed, or virtual machine environments disconnected from production systems. Backup routines should be maintained, and only verified repositories such as VirusTotal should be used for obtaining malware samples. The inclusion of the harmless test virus in this project eliminates these risks, allowing for practical experimentation without exposure to actual threats. Ethical responsibility is equally important in cybersecurity research and software development. The scanner must be used solely for legitimate and authorized purposes, with explicit permission from system owners before conducting scans. Privacy should be respected when analyzing shared or networked storage, and the dissemination of malware samples, even for research, must follow proper containment and disclosure protocols. The knowledge and technical skills gained through this project should be directed toward defensive applications, system improvement, and academic inquiry—not for offensive or malicious intent. Adherence to institutional ethics policies, responsible disclosure guidelines, and relevant cybersecurity laws ensures that research using the Simple Antivirus Scanner contributes positively to the advancement of digital safety and the integrity of the academic community.

5 | CONCLUSION AND RECOMMENDATIONS

The Simple Antivirus Scanner serves as an effective educational platform for understanding the foundational principles of malware detection and antivirus system design. By implementing signature-based scanning through MD5 hashing and database comparison, the project successfully illustrates the basic mechanisms that form the basis of all modern antivirus technologies. Its development using Delphi allows efficient execution through native Windows integration, while the use of the *TBackgroundWorker* component ensures a smooth and responsive user interface even during extensive scanning operations. The project's main strengths lie in its clarity of concept, straightforward implementation, and accessibility for learning and experimentation. The inclusion of a harmless test virus allows users to safely observe how detection processes work in practice, while the demonstration of cryptographic hash functions provides tangible understanding of how malware signatures are identified and matched. Through its structured yet simple design, the scanner encourages users to grasp the theoretical and practical aspects of malware detection before advancing to more complex approaches, such as heuristic or behavioral analysis. Despite its educational advantages, the system's limitations compared to commercial antivirus software are instructive. Its inability to detect new or unknown malware, lack of heuristic evaluation, and absence of real-time protection demonstrate why modern cybersecurity solutions depend on multiple detection layers, frequent signature updates, and advanced intelligence models. Nevertheless, these constraints emphasize the scanner's purpose as a pedagogical tool, allowing learners to appreciate both the strengths and weaknesses of signature-based detection within the broader context of antivirus technology evolution.

For students, developers, and cybersecurity enthusiasts, this project offers hands-on experience with essential computing concepts such as file system operations, cryptography, database management, and multithreading. Its open and modifiable source code promotes curiosity and independent learning, encouraging users to expand its functionality with new algorithms, user interface improvements, or real-time protection mechanisms. To use the Simple Antivirus Scanner effectively, users should begin by downloading the source code from SourceCodester, installing a compatible version of Delphi, and obtaining the *TBackgroundWorker* component from DelphiArea. After compiling and running the project, users can start with small test directories before scanning entire drives. The included test virus serves as a safe example to verify that the detection process functions correctly. Creating custom signatures for benign files further helps users understand how database matching works and how even minor changes in file content alter hash values. For structured learning, users are encouraged to study the program's source code to understand its architecture, including the scanning algorithm, hash computation, and signature matching logic. Practical exercises may involve scanning directories with varying file counts to observe performance differences, testing the detection of modified files, and enhancing the program with features such as new hash algorithms or additional reporting functions. Safety and ethical considerations remain critical when using this tool. Users should never analyze real or potentially harmful malware on production systems. All testing should take place in isolated virtual machines or sandboxed environments to prevent accidental infection or data loss. Maintaining regular backups before conducting experiments is strongly advised.

The scanner should not be distributed or relied upon as a production-level security solution but should always be identified as an educational resource. To ensure real protection, it should be used alongside trusted commercial antivirus software. Practicing responsible and ethical use reinforces its intended purpose as a learning instrument

rather than a security product. Looking forward, the Simple Antivirus Scanner offers considerable potential for further development. Future improvements could involve expanding its architecture into a comprehensive security suite that incorporates multiple detection layers, or developing cross-platform compatibility for Linux and macOS. Cloud-based signature distribution could be implemented to allow automated updates, while the integration of machine learning models could enhance classification accuracy for detecting new and evolving malware types. Extending the project to mobile environments such as Android and iOS would further broaden its educational and research applications. Another promising direction involves transforming it into a collaborative open-source initiative, inviting contributions from global developers for continual improvement, signature database expansion, and integration with modern threat intelligence systems. Alternatively, the project could evolve into a structured educational framework featuring enhanced documentation, interactive learning modules, and assessment tools for cybersecurity training programs. Regardless of its future development path, the core concepts demonstrated by the Simple Antivirus Scanner—hash-based detection, file system scanning, and data-driven malware identification—will remain fundamental to understanding the mechanics of digital threat prevention. Its emphasis on clarity, simplicity, and accessibility ensures that it continues to serve as both a valuable educational resource and a foundation for innovation in the field of cybersecurity.

ACKNOWLEDGMENTS

Appreciation to Antonov Alin for developing and sharing this educational project, the SourceCodester community for providing a platform for knowledge sharing, DelphiArea for the TBackgroundWorker component, the Delphi community for excellent development tools and resources, and security researchers who advance malware detection technology.

REFERENCES

Ali, A., & Farhan, A. (2020). A novel improvement with an effective expansion to enhance the MD5 hash function for verification of a secure e-document. *IEEE Access, 8*, 80290–80304. https://doi.org/10.1109/ACCESS.2020.2989050.

Aycock, J. (2006). *Computer viruses and malware*. Springer.

Boyanov, P. (2024). Practical applications of hash functions MD5, SHA-1, and SHA-256 using various software tools to verify the integrity of files. *Journal Scientific and Applied Research, 27*(1), 120–137. https://doi.org/10.46687/jsar.v27i1.413.

Bonfante, G., Kaczmarek, M., & Marion, J. (2008). Architecture of a morphological malware detector. *Journal in Computer Virology, 5*(3), 263–270. https://doi.org/10.1007/s11416-008-0102-4.

Darem, A., Ghaleb, F., Alhashmi, A., Abawajy, J., Alanazi, S., & Al-Rezami, A. (2021). An adaptive behavioral-based incremental batch learning malware variants detection model using concept drift detection and sequential deep learning. *IEEE Access, 9*, 97180–97196. https://doi.org/10.1109/ACCESS.2021.3093366

Du, D., Sun, Y., Ma, Y., & Xiao, F. (2019). A novel approach to detect malware variants based on classified behaviors. *IEEE Access, 7*, 81770–81782. https://doi.org/10.1109/ACCESS.2019.2924331

Delphi documentation. Embarcadero Technologies. (n.d.). Delphi documentation. https://docwiki.embarcadero.com/

Gandotra, E., Bansal, D., & Sofat, S. (2014). Malware analysis and classification: A survey. *Journal of Information Security, 5*(2), 56–64. https://doi.org/10.4236/jis.2014.52006

Gaudesi, M., Marcelli, A., Sánchez, E., Squillero, G., & Tonda, A. (2015). Malware obfuscation through evolutionary packers (pp. 757–758). https://doi.org/10.1145/2739482.2764940

Kessler, G. (2016). The impact of MD5 file hash collisions on digital forensic imaging. *The Journal of Digital Forensics, Security and Law.* https://doi.org/10.15394/JDFSL.2016.1431

Kim, T., Kang, B., & Im, E. (2018). Runtime detection framework for Android malware. *Mobile Information Systems, 2018*, Article 8094314. https://doi.org/10.1155/2018/8094314

Mat, S., Razak, M., Kahar, M., Arif, J., Asmara, S., & Firdaus, A. (2021). Towards a systematic description of the field using bibliometric analysis: Malware evolution. *Scientometrics, 126*(3), 2013–2055. https://doi.org/10.1007/s11192-020-03834-6

Nkouankou, A., Clarice, F., Abel, W., & Ndoundam, R. (2022). Pre-image attack of the MD5 hash function by proportional logic. *International Journal of Research and Innovation in Applied Science, 7*(8), 20–25. https://doi.org/10.51584/ijrias.2022.7802

Oliver, J., Cheng, C., & Chen, Y. (2013). TLSH — a locality sensitive hash (pp. 7–13). https://doi.org/10.1109/CTC.2013.9

Rivest, R. (1992). *RFC 1321: The MD5 message-digest algorithm*. Internet Engineering Task Force. https://www.ietf.org/rfc/rfc1321.txt

Santos, I., Brezo, F., Sanz, B., Laorden, C., & Bringas, P. (2011). Using opcode sequences in single-class learning to detect unknown malware. *IET Information Security, 5*(4), 220–227. https://doi.org/10.1049/iet-ifs.2010.0180

Sikorski, M., & Honig, A. (2012). *Practical malware analysis: The hands-on guide to dissecting malicious software* (1st ed.). No Starch Press.

Souri, A., & Hosseini, R. (2018). A state-of-the-art survey of malware detection approaches using data mining techniques. *Human-Centric Computing and Information Sciences, 8*(1). https://doi.org/10.1186/s13673-018-0125-x

Signature-based detection in antivirus software. (2015). *Journal of Computer Security. (Note: author(s), volume, and page numbers not provided in original list.)*.

Surendran, R., & Thomas, T. (2022). Detection of malware applications from centrality measures of syscall graph. *Concurrency and Computation: Practice and Experience, 34*(10). https://doi.org/10.1002/cpe.6835

TBackgroundWorker component documentation. (n.d.). DelphiArea. http://www.delphiarea.com/

Wagener, G., State, R., & Dulaunoy, A. (2007). Malware behaviour analysis. *Journal in Computer Virology, 4*(4), 279–287. https://doi.org/10.1007/s11416-007-0074-9

Wang, L., Ohta, K., Sasaki, Y., Sakiyama, K., & Kunihiro, N. (2010). Cryptanalysis of two MD5-based authentication protocols: APOP and NMAC. *IEICE Transactions on Information and Systems, E93-D*(5), 1087–1095. https://doi.org/10.1587/transinf.e93.d.1087

Wang, X., & Yu, H. (2005). How to break MD5 and other hash functions (pp. 19–35). https://doi.org/10.1007/11426639_2

Yerima, S., Sezer, S., & Muttik, I. (2015). High accuracy Android malware detection using ensemble learning. *IET Information Security, 9*(6), 313–320. https://doi.org/10.1049/iet-ifs.2014.0099

How to cite this article: Alin, A. (2025). Simple Antivirus Scanner: A Signature-Based Malware Detection System Using Delphi. *Journal Dekstop Application (JDA)*, *4*(2), 68-76. https://doi.org/10.59431/jda.v4i2.665.