WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

DEVICESHIELD

Krishvanth K R[1], Rajkumar S[2], Nikhil Dev P J[3] ,Mrs. N. G. Dharaniya[4]

Assistant Professor, Department of Information Technology,
Sri Shakthi Institute of Engineering and Technology, Coimbatore-641062, Tamil Nadu, India.
Email ID: krishvanth345@gmail.com

ABSTRACT :

Device owners rarely hold a record of the original hardware configuration of their laptops, so after servicing it is difficult to check whether components such as RAM, storage, GPU or motherboard have been replaced or changed, and manual checking is slow and unreliable. This paper presents DeviceShield AI, a web-based hardware integrity monitoring and comparison system for Windows computers. The system scans the processor, motherboard, BIOS, memory modules, storage drives, graphics adapter, battery and network adapters through Windows Management Instrumentation, stores each scan as a hardware profile, and compares a later profile with an earlier one. A rule-based module checks the identity of each component, counts the differences and assigns a LOW, MEDIUM or HIGH risk level with recommendations; it is not a trained machine-learning model. The back end is written in Python with Flask and keeps user accounts, devices, hardware profiles and comparison reports in a SQLite database, and a browser dashboard presents the results. In the hosted mode each user has a private account, and a small loopback agent on the user's own computer performs the scan. The system was run on a Lenovo laptop with an AMD Ryzen 5 7235HS processor, where the saved scan history reached scan number 115 and a comparison of two consecutive scans reported no component changes. Ten constructed scenarios and twenty-one API checks were executed. The scenarios confirmed the behaviour of the rules and the API checks confirmed per-account privacy, while the tests also exposed limitations such as a failure for computers without a battery and false alarms from virtual network adapters. The reported differences require manual verification and do not, by themselves, prove unauthorized hardware replacement.

Keywords: Hardware Integrity, Hardware Inventory, Windows Management Instrumentation, Baseline Comparison, Rule-Based Risk Classification, Servicing Verification, Flask, SQLite.

1. INTRODUCTION

A laptop that is handed over for servicing may return with a new keyboard, a cleaned fan or a replaced battery, and in most cases the work is exactly what the customer agreed to. The difficulty is that the owner rarely has a written record of the original hardware configuration. Most people know the approximate size of their memory and storage, but not the module manufacturer, the serial numbers or the exact graphics adapter. After the service, an owner who wishes to confirm that the RAM, the storage drive, the graphics hardware or the motherboard are still the original parts has nothing to compare against, and a legitimate repair and an unrecorded replacement look the same.

DeviceShield addresses this gap with a simple idea: record the hardware identity before the service, record it again afterwards, and let the software compare the two records. The application reads the information through Windows Management Instrumentation (WMI) [7], keeps each scan in a SQLite database, and presents the profile and the comparison in a browser dashboard. Fig. 1 shows the stages of a servicing cycle and marks the three stages that the system supports: the baseline scan, the later scan and the comparison.

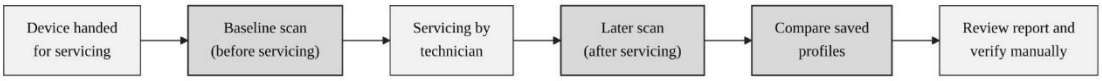


Fig. 1. Stages of a servicing cycle and the stages supported by DeviceShield.

1.1 Existing challenges in verifying hardware after servicing

Manual verification means opening the Windows settings pages or the Device Manager, reading model names by eye and comparing them with memory or with an old specification sheet. Serial numbers and MAC addresses are especially hard to check by hand. Built-in tools and third-party hardware viewers answer the question of what is in the computer now, but, to the knowledge of the authors, the common desktop tools do not by default keep a private history of scans, compare two scans of the same device, or summarise the differences with a risk level. Enterprise asset-management products keep inventories, but they are designed for organisations that administer fleets of devices, not for an individual who wants to check a single laptop after a repair.

Stronger assurance exists at the firmware level. NIST guidelines describe the protection of the system BIOS and platform firmware resiliency [1], [2], and the TPM specification defines a hardware root of trust that can measure the boot process for attestation [3]. These mechanisms need platform support, administrative set-up and a verifier, and they do not give an ordinary owner a simple list of which memory module or drive was replaced. Any tool at the level of an ordinary owner can only estimate changes from the identifiers that the computer reports, so its output should be treated as guidance and not as a guarantee.

1.2 Baseline comparison and rule-based risk classification

Keeping a trusted baseline and comparing later observations with it is well established in system security. File-integrity checkers such as Tripwire store a database of file attributes and report any file whose attributes differ from the stored values [4]. DeviceShield applies the same idea to hardware identity instead of file contents: the first scan acts as the baseline, a later scan acts as the observation, and the difference between them is the finding. Research on counterfeit integrated circuits [5] and hardware Trojans [6] shows that physical access to a device is a real threat model, but those works operate at the silicon level and require laboratory methods. DeviceShield works at a much simpler level, because it only observes the identity that each component reports to the operating system.

The comparison is performed by fixed rules and not by a trained machine-learning model. The program checks the identity field of each component, counts how many components differ, and maps the count to a risk level: no difference is LOW, one or two differences are MEDIUM and three or more are HIGH. The word "AI" in the project title refers to the intention of automating the analysis and the recommendation, and the work does not claim any statistical learning. A difference between two scans does not prove that a part was replaced without permission, because drivers, virtual network adapters, a legitimate repair or a change in how Windows reports a field can all produce differences.

1.3 Problem statement and objectives

In many computer servicing situations, device owners may not have a proper record of their laptop's original hardware configuration, and manual verification after servicing is time-consuming and may not be accurate. There is therefore a need for a lightweight tool that scans and records the available hardware details, stores the hardware profile, compares it with a later scan, and generates a report with a risk level and recommendations, while stating clearly that a reported difference is a prompt for verification and not proof of tampering.

The main objective of this work is to develop a web-based Hardware Integrity Monitoring and Comparison System for Windows computers. The specific objectives are:

i. To scan a Windows computer and collect the CPU, motherboard, BIOS, RAM, storage, GPU, battery and network adapter details through WMI.
ii. To store every scan as a hardware profile in a database so that a history of the device is available.
iii. To compare any two saved profiles of the same device and list the components whose identity has changed.
iv. To assign a rule-based risk level (LOW, MEDIUM or HIGH) and generate recommendations for each comparison.
v. To provide a browser dashboard that shows the current profile, the scan history and the comparison report.
vi. To support user accounts so that every user sees only their own devices, scans and comparisons.
vii. To allow a hosted dashboard to scan a user's own computer through a loopback-only local agent.
viii. To evaluate the system through scenario testing of the comparison rules and API testing of the back end, and to document its limitations.

2. MATERIALS AND METHODS

2.1 Development approach

The project was developed in an iterative, incremental manner, following the incremental development model described in standard software engineering texts [9]. A working scanner was written first, then the database, then the API, then the comparison rules, and finally the dashboard and the hosted mode with accounts. Each increment was run on a real Windows laptop before the next was started, which allowed errors in the data returned by WMI to be found early. The scope covers Windows 10 and Windows 11 computers, eight hardware categories, a SQLite database with four tables, a Flask REST API, a single-page dashboard, user registration and sign-in, a loopback scanner agent and a deployment configuration for a small cloud service.

2.2 System architecture

The system is organised in five layers (Fig. 2). The presentation layer is the browser dashboard written in HTML, CSS and JavaScript. The application layer is the Flask API, which handles authentication, scan storage, history and comparison. The analysis layer is the rule-based comparison module. The data layer is the SQLite database. The acquisition layer runs on the computer being scanned and consists of the loopback agent and the WMI scanner.

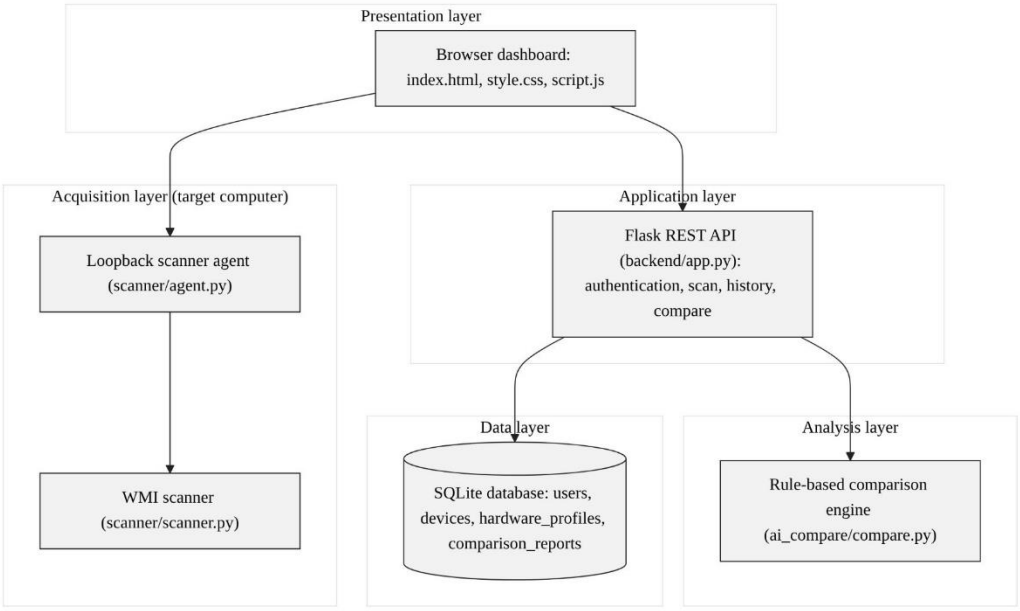


Fig. 2. Layered architecture of DeviceShield.

In local mode, the Flask application runs on the computer to be scanned and starts the scanner as a subprocess. In public mode, the Flask application runs on a server and never scans its own host; the browser obtains the profile from the agent on the user's computer and then posts it to the server. This separation is the central design decision of the project, because a hosted server can neither read the hardware of the user's computer nor should it be able to. A web page cannot read such information itself, so the agent listens only on the loopback address 127.0.0.1 (port 8765), accepts requests only from the configured site origin, and sets the Private Network Access response header. Browser rules for cross-origin requests [10] and the Private Network Access checks introduced in Chrome [11] apply to this connection, and the user is asked to allow it.

2.3 Technology stack and development environment

Hardware information is read from Windows through WMI with the Python wmi module [14]. The back end uses Python 3.10 or newer with Flask [12], Flask-Cors and Werkzeug, and stores data in SQLite [13] through the sqlite3 module, so that no separate database server is needed. The API follows the REST style [15] and exchanges JSON. The dashboard is a single page made of index.html, style.css and script.js and uses no framework. For an optional public deployment the project includes a render.yaml file for the Render service with gunicorn and a 1 GB persistent disk that holds the SQLite file. The reference computer used for development and testing was a Lenovo laptop with an AMD Ryzen 5 7235HS processor, 24 GB of RAM, a 476 GB SSD and an NVIDIA GeForce RTX 3050 6GB Laptop GPU. The implementation consists of about 1,500 lines of Python, JavaScript, HTML and CSS.

2.4 Data collection and processing

The scanner creates a WMI connection and runs one function for each of the eight categories, storing the selected fields in a dictionary. RAM, storage and network adapters are lists, because a computer can have several modules, drives and adapters. Only network adapters that report a MAC address are kept, and the battery is stored as null if no battery is present. Much of the board, BIOS and memory information originates from the SMBIOS tables supplied by the firmware [8], so the reliability of the values depends on the manufacturer: some systems report blank or placeholder serial numbers. The result is a JSON profile with one key for each category. The back end accepts a profile only if it is a JSON object whose cpu entry, when present, is also an object, and it truncates the device name to 120 characters. Table 1 lists the categories, the WMI class that supplies each of them, the field that the comparison treats as the identity of the component, and the comparison rule.

Category	**WMI class**	**Identity compared**	**Rule**
CPU	Win32_Processor	name	Reported if the name differs
Motherboard	Win32_BaseBoard	serial number	Reported if the serial number differs
BIOS	Win32_BIOS	serial number	Reported if the serial number differs
RAM	Win32_PhysicalMemory	serial number of each module	Reported if the set of serial numbers differs; otherwise the capacity of each module is compared
Storage	Win32_DiskDrive	serial number of each	Reported if the set of serial numbers differs

Category	WMI class	Identity compared	Rule
		drive	
GPU	Win32_VideoController	name	Reported if the name differs
Battery	Win32_Battery	device ID	Reported if the device ID differs
Network	Win32_NetworkAdapter	MAC address of each adapter	Reported if the set of MAC addresses differs

Table 1. Hardware categories, WMI classes, identity fields and comparison rules.

2.5 Database design

The database has four related tables. The users table holds the account name (unique and case-insensitive) and the password hash. A device belongs to a user and is identified within that user by its device name. A hardware profile belongs to a device and stores the eight component categories as JSON text together with the scan date. A comparison report belongs to a device and stores the risk level, the verdict, the report text and the creation time. Foreign keys are enabled for each connection, and every query that returns scans or comparisons filters by the identifier of the signed-in user. In the hosted mode the SQLite file is placed on persistent storage so that the history survives restarts.

2.6 Comparison algorithm and risk classification

The comparison is implemented in a separate module (compare.py) that takes two profile files as input and has three steps. First, the single-identity categories are compared by one field each. Second, the multi-component categories are converted to maps from the identity value to the component, and the key sets are compared; for RAM, when the keys are equal, the capacity of each module is compared as well. When a component has no identity value, for example a blank serial number, a position-based key such as item-0 is used, so that two such lists of the same length compare as equal. Third, the number of recorded changes decides the risk level, as shown in Table 2.

Changes	Risk level	Verdict	Recommendation in the report
0	LOW	NO CHANGES DETECTED	No identity changes found; keep the baseline scan for future checks
1 or 2	MEDIUM	REVIEW CHANGES	Verify each change against the service receipt or technician notes; if a replacement was not authorised, contact the repair provider before using the device for sensitive work
3 or more	HIGH	HIGH RISK	Several identities changed; have the device inspected by a trusted technician; avoid entering sensitive credentials until the changes are verified

Table 2. Risk levels, verdicts and recommendations.

The thresholds are deliberately simple. They treat every category equally, so the replacement of a battery counts as much as the replacement of a motherboard. This simplicity makes the rule easy to explain and inspect. Every report states that the method is rule-based component identity checking and not a trained AI model.

2.7 Scan and comparison workflow

When the user selects Scan device, the dashboard disables the button and chooses the path by mode. In local mode it calls GET /api/scan on the Flask server, which runs the scanner and saves the result. In public mode it calls the local agent, which runs the scanner in a subprocess with a 90-second time limit and returns the profile; the browser then posts the profile to POST /api/scan, and the server saves it under the signed-in account. To compare, the user selects a baseline scan and a later scan of the same device. The server checks that both scan identifiers belong to the user and to the same device, runs the comparison, stores the report and returns it. Errors, such as an agent that is not running, are turned into a clear message on the dashboard.

2.8 Security and privacy

A hardware profile contains serial numbers and MAC addresses that identify a particular machine, so the design treats it as private data. Scans are stored only after the user signs in, each account sees only its own devices and scans, and the component details view of the dashboard hides serial numbers and MAC addresses. Passwords are stored only as salted scrypt hashes produced with the Werkzeug helper [17], following the recommendation of the OWASP Password Storage Cheat Sheet [16]. The session cookie is HttpOnly and SameSite=Lax and, in public mode, Secure. Registration is limited to five attempts per hour and sign-in to ten attempts per minute for each client address. All database statements use bound parameters, the request body is limited to 2 MB, and in public mode the server sends a Content-Security-Policy and other security headers. Broken access control is listed among the most

common web application weaknesses [18], which is why the per-account filtering of every query was tested explicitly.

2.9 Testing and evaluation

Testing combined three approaches. The scanner and the dashboard were exercised on the reference laptop in local mode. The comparison module was tested with ten defined scenarios, each built by changing one property in a copy of a baseline profile that mirrors the reference laptop; the serial numbers in this profile are simulated values. The Flask API was tested with a scripted client against a fresh database in public mode, using twenty-one test cases. These checks ran in a Linux test container where the WMI scanner cannot run, so the constructed profile was used, and the cross-origin extension was replaced by a stub because the package could not be installed there. The repository does not include an automated test suite, and the loopback agent was checked by code review and design but was not exercised in the API tests.

3. RESULTS AND DISCUSSION

3.1 Implemented system functionality

The application brings the main steps of hardware verification into one interface. On the reference laptop, a scan recorded the profile shown in Fig. 3 and Fig. 4: AMD Ryzen 5 7235HS (4 cores, 8 logical processors), NVIDIA GeForce RTX 3050 6GB Laptop GPU with driver 32.0.15.9166, 24 GB of memory in two modules, one 476 GB Samsung SSD, a battery with status OK, and nine network adapters with a MAC address. The profile coverage reported six of six categories (100 percent); the dashboard states that coverage shows reported fields only and is not a hardware health or security score.

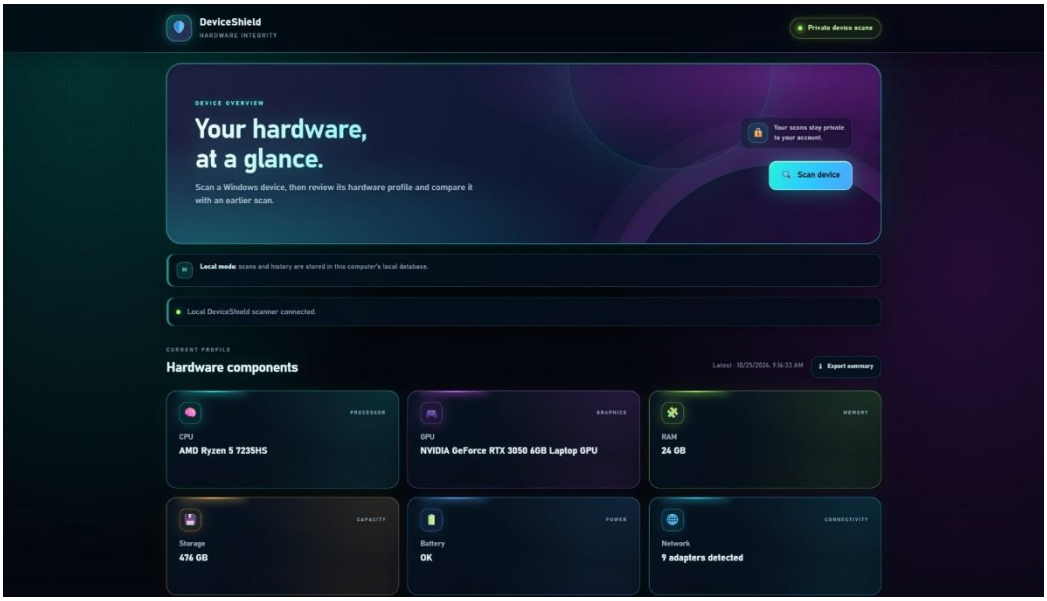


Fig. 3. Dashboard home screen with the hero section, mode notices and hardware cards.

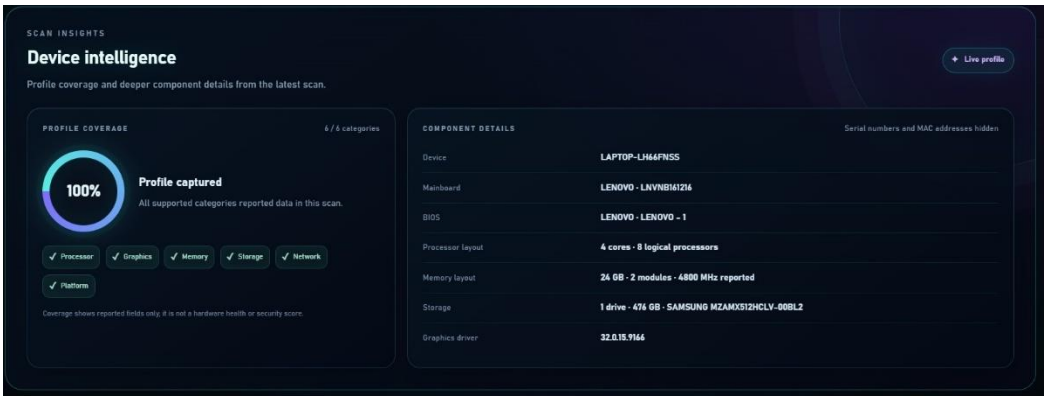


Fig. 4. Device intelligence: profile coverage and component details.

Scans are kept between sessions: the scan numbers on the reference laptop reached #115 and the history spans 24 and 25 October 2026. The comparison of scan #114 (24 October, 4:05:55 PM) with scan #115 (25 October, 9:16:33 AM) reported NO CHANGES DETECTED with risk level LOW and recommended keeping the baseline scan (Fig. 5), which is the expected result for two scans taken a day apart without servicing. The history endpoint returns at most the latest 100 scans, and the badge on the dashboard shows that count.

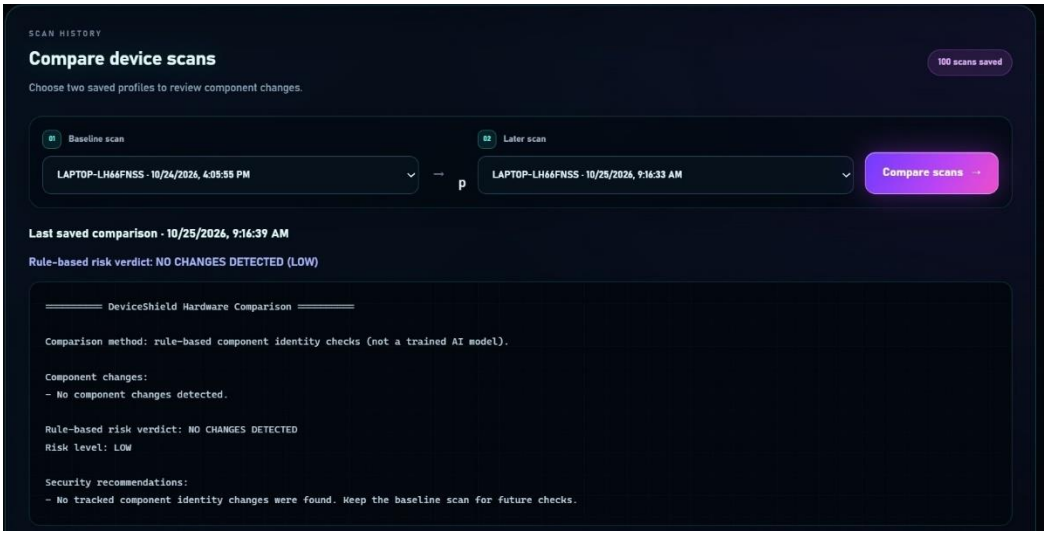


Fig. 5. Compare device scans panel with the last saved comparison.

3.2 Results of the comparison scenarios

Table 3 shows the output of the comparison module for the ten scenarios. Scenarios S1 to S5 show that the risk level follows the number of changes as designed. Scenario S6 shows a false positive, because a VPN adapter is not a hardware replacement. Scenario S8 shows that a driver update is invisible to the rules, which is acceptable for a tool about component identity but should be known by the user. Scenario S10 exposed a defect: when the battery value of the later profile is null, as for a computer without a battery, the comparison script stops with an AttributeError.

ID	Change in the later profile	Changes reported	Risk level	Result
S1	None (identical profile)	0	LOW	Pass
S2	GPU name changed to RTX 3060	1 (GPU)	MEDIUM	Pass
S3	Serial number of one RAM module changed	1 (RAM list)	MEDIUM	Pass
S4	One RAM serial and the SSD serial changed	2 (RAM, storage)	MEDIUM	Pass
S5	Motherboard, BIOS and SSD serials changed	3	HIGH	Pass
S6	A virtual (VPN) network adapter added	1 (network list)	MEDIUM	Pass; innocent change flagged
S7	RAM capacity differs for the same serial number	1 (capacity)	MEDIUM	Pass
S8	Only the GPU driver version updated	0	LOW	Pass by design; not tracked
S9	CPU name changed	1 (CPU)	MEDIUM	Pass
S10	Battery value is null	–	–	Fail; script error

Table 3. Output of the comparison module for ten scenarios.

3.3 Results of the API checks

All twenty-one API test cases returned the expected status codes and bodies (Table 4). Requests without a signed-in session were rejected with HTTP 401, invalid input was rejected with HTTP 400, and a duplicate user name in different letter case was rejected with HTTP 409. Two users registered the same device name and each saw only their own scans (2 and 1), and an attempt to compare a scan of one user with a scan of the other returned HTTP 400, which supports the per-account privacy objective. Six further registrations within an hour were refused with HTTP 429, and the stored password was confirmed to be a scrypt hash and not the password. After the checks, the database contained two users, two devices, three hardware profiles and one comparison report, which agrees with the actions performed.

Group	Test IDs	Expected outcome	Result
Access without sign-in	A1, A2	401	2 of 2 passed
Input validation	A3, A4, A10	400	3 of 3 passed
Registration, sign-in, session and sign-out	A5–A9, A19	201, 409, 401, 200	6 of 6 passed
Per-account isolation	A11, A12, A14, A15	Own data only; cross-account compare 400	4 of 4 passed
Comparison of own scans	A13	200, MEDIUM	1 of 1 passed
Mode, headers and static files	A16–A18	405, 200 with headers, 404	3 of 3 passed
Rate limiting and password storage	A20, A21	429; hash stored	2 of 2 passed

Table 4. Summary of the API test cases.

3.4 Comparison with conventional approaches

Table 5 compares the approaches that an owner could use to verify hardware after a service. DeviceShield does not replace firmware-level attestation, which gives stronger evidence of boot state, and it should not be considered more accurate than other tools without comparative testing. Its contribution is practical: it is free, scans in one click, keeps a private history, and compares two scans in plain language.

Approach	Strengths	Limitations
Manual inspection	No tools needed	Slow and error-prone; serial numbers are hard to read; no record
Windows System Information or Device Manager	Built in; shows the current configuration	No history or comparison; each field is read manually
Third-party hardware viewers	Detailed current report	Usually no private scan history or automatic comparison
Enterprise asset management	Fleet inventory and policies	Needs an administrator and infrastructure; not aimed at individuals
TPM or firmware attestation	Strong cryptographic evidence of boot state	Needs platform support and a verifier; does not list replaced parts
DeviceShield	Free; one-click scan; stores history; compares two scans; risk level and recommendations	Windows only; relies on identifiers reported through WMI; rule-based; results need manual verification

Table 5. Approaches to verifying hardware after servicing.

3.5 Discussion and limitations

The results show that the application achieves its purpose in the situations tested. It records the profile of a real Windows laptop, keeps a history, compares two scans and reports the differences with a risk level and recommendations. The scenario tests show that the rules detect a change in any tracked identifier, scale the risk with the number of changes and are transparent. They also show the price of simplicity. The tests and the reading of the source code revealed the following limitations, which do not stop the application from working in normal use but affect the accuracy of the reports:

- A null battery value stops the comparison script, so the compare request fails for computers without a battery.
- The arrow in the change messages is stored with an incorrect character encoding and appears as odd characters in the printed report.
- The scanner stores one CPU, GPU and battery, so a laptop with integrated and dedicated graphics reports only one GPU.
- Risk depends only on the number of changes, so a battery change counts as much as a motherboard change.
- Virtual network adapters are compared like physical ones, which can raise a MEDIUM result without any hardware replacement.
- Driver and firmware versions are not part of the identity, and a replacement part with a blank serial number is not detected.
- The history endpoint returns at most 100 scans, and the rate-limit counters are kept per process, count failed attempts and reset on restart.
- There is no password recovery or email verification, SQLite suits a single instance, and the repository has no automated tests.

The most important point for the user is the wording of the output. A MEDIUM or HIGH verdict means that the reported identities differ and should be

verified against the service receipt or the technician's notes. It does not state that a part was stolen or replaced without permission, and the report says so. The comparison also depends on a trusted baseline: a baseline taken after tampering would hide the change, which is why the owner is encouraged to scan before the device leaves their hands.

4. CONCLUSION

DeviceShield AI presents a practical approach to helping a device owner check the hardware of a Windows laptop before and after servicing. The system scans eight hardware categories through WMI, stores each scan as a profile, compares two saved profiles with transparent identity rules, and presents a report with a risk level and recommendations in a browser dashboard. A hosted mode with user accounts and a loopback scanner agent allows the dashboard to be used over the internet without exposing the hardware data of one user to another account. The tests on a real laptop, ten comparison scenarios and twenty-one API checks support the working of the rules and the privacy of accounts, and they also document the limitations of the current version.

The effectiveness of the system depends on the identifiers that manufacturers report, on a trusted baseline and on further testing, and a reported difference is never proof of unauthorized replacement. Future development can focus on fixing the null handling and the arrow encoding in the comparison module, storing GPUs and batteries as lists, weighting the risk by component type, ignoring known virtual adapters, reporting driver and firmware versions as informational lines, producing a PDF report with a time stamp, adding email verification and password recovery, moving to a managed database, adding automated tests, extending the scanner to Linux and macOS, and exploring TPM-based attestation.

5. ACKNOWLEDGMENTS

The authors thank Sri Shakthi Institute of Engineering and Technology for the academic environment and facilities that made this work possible. They are also grateful to Dr. S. Prakash, Head of the Department of Information Technology, to the project guide Mrs. N. G. Dharaniya, to the project co-ordinator Dr. T. Esther and to the faculty members for their advice and encouragement during the project.

6. REFERENCES

1. D. Cooper, W. Polk, A. Regenscheid, and M. Souppaya, "BIOS protection guidelines," NIST Special Publication 800-147, Apr. 2011.
2. A. Regenscheid, "Platform firmware resiliency guidelines," NIST Special Publication 800-193, May 2018.
3. Trusted Computing Group, "TPM 2.0 Library Specification." Available at: https://trustedcomputinggroup.org/resource/tpm-library-specification/
4. G. H. Kim and E. H. Spafford, "The design and implementation of Tripwire: A file system integrity checker," in Proc. 2nd ACM Conf. Computer and Communications Security (CCS), 1994, pp. 18–29.
5. U. Guin, K. Huang, D. DiMase, J. M. Carulli, M. Tehranipoor, and Y. Makris, "Counterfeit integrated circuits: A rising threat in the global semiconductor supply chain," Proc. IEEE, vol. 102, no. 8, pp. 1207–1228, Aug. 2014.
6. M. Tehranipoor and F. Koushanfar, "A survey of hardware Trojan taxonomy and detection," IEEE Design & Test of Computers, vol. 27, no. 1, pp. 10–25, Jan.–Feb. 2010.
7. Microsoft, "Windows Management Instrumentation." Available at: https://learn.microsoft.com/en-us/windows/win32/wmisdk/wmi-start-page
8. Distributed Management Task Force (DMTF), "System Management BIOS (SMBIOS) Reference Specification," DSP0134. Available at: https://www.dmtf.org/standards/smbios
9. I. Sommerville, Software Engineering, 10th ed. Harlow, U.K.: Pearson, 2016.
10. Mozilla, "Cross-Origin Resource Sharing (CORS)," MDN Web Docs. Available at: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS
11. Google Chrome Developers, "Private Network Access: introducing preflights." Available at: https://developer.chrome.com/blog/private-network-access-preflight
12. Pallets Projects, "Flask documentation." Available at: https://flask.palletsprojects.com/
13. SQLite Consortium, "SQLite documentation." Available at: https://www.sqlite.org/docs.html
14. T. Golden, "Python WMI module." Available at: https://github.com/tjguk/wmi
15. R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, Univ. of California, Irvine, 2000.
16. OWASP Foundation, "Password Storage Cheat Sheet." Available at: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html
17. Pallets Projects, "Werkzeug documentation: security helpers." Available at: https://werkzeug.palletsprojects.com/
18. OWASP Foundation, "OWASP Top Ten." Available at: https://owasp.org/www-project-top-ten/