

DEVADARA: an offline, privacy-first antivirus for Windows

Project case study, by Chavindu Jayawardana

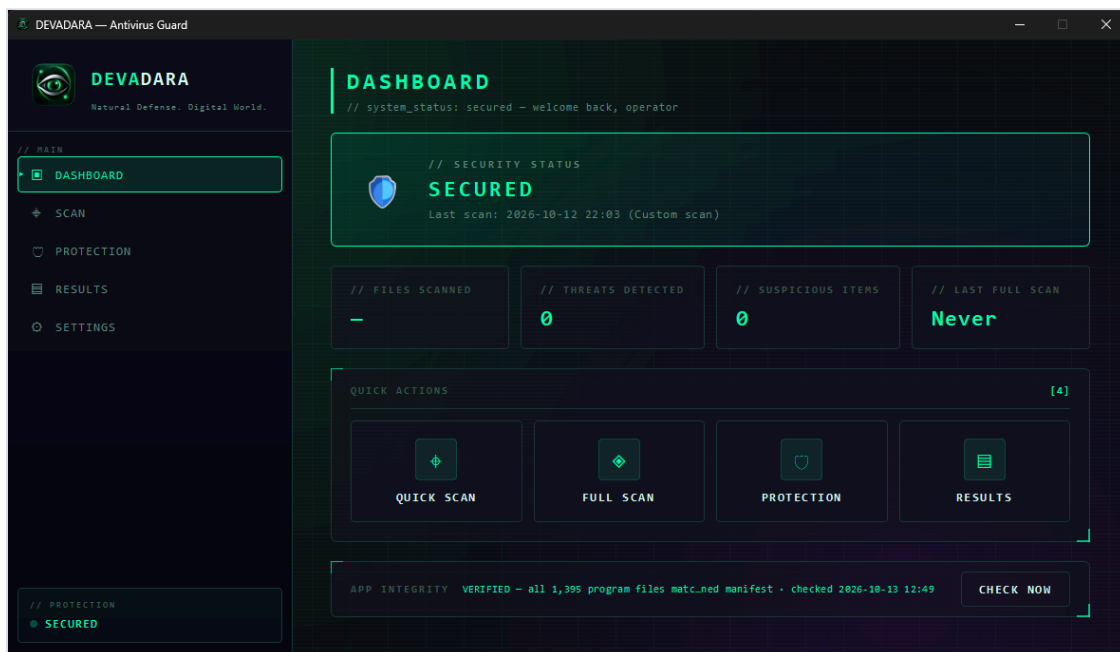
October 2026

1. Introduction

DEVADARA is a desktop antivirus for Windows 10/11 (64-bit) that runs fully on the user's own PC. It checks files with four layers: a SHA256 hash check, YARA rules, a Windows digital signature check and a small machine-learning model. It can quarantine files in an encrypted, restorable way, watch the Downloads folder and USB drives, and warn about ransomware-like bursts of file activity. It has no cloud, no account and no telemetry, and it never deletes or quarantines anything without the user's explicit click.

It is a learning project and a portfolio piece. It is not meant to replace Microsoft Defender or any full antivirus product. It is for people who are curious about how an antivirus works inside, and for anyone reviewing my work as a developer.

- **Current version:** v1.0.0 (defined in `devadara/version.py`)
- **Project page:** <https://www.chavindujayawardana.com.lk/devadara>
- **Download and release notes:** <https://github.com/chavindujayawardhana-pixel/devadara-releases>



The Dashboard: overall protection status, last scan, counters, quick actions, and the app's own integrity check ("all 1,395 program files match the signed manifest").

2. Why I built it

I am an IT student from Sri Lanka. I do not have a formal coding background. I learned by building, working with Claude Code as my pair programmer.

I built DEVADARA for three reasons:

1. **I needed a portfolio project.** I wanted one real, complete piece of software that I could show and explain from top to bottom.
2. **I wanted to learn how antivirus tools really work.** Not just "it finds viruses", but what actually happens to a file when it is scanned, why false alarms happen, and how a security tool protects itself.
3. **Privacy first, offline by design.** I wanted a tool that never uploads anyone's files. Everything, including the ML model, runs on the local machine.

3. The core idea: an immune system for files

I designed DEVADARA around the human immune system. The idea also shows in the code: the module comments call the ML layer the "instinct" layer, quarantine the "isolation ward", the Downloads watcher the "mouth" layer, and the USB watcher the "skin and gut" layer.

Immune system	DEVADARA	Where in the code
Memory cells recognise threats seen before	Exact SHA256 hash matching against known-bad files, and Authenticode signature checks that recognise trusted software	<code>hash_scanner.py</code> , <code>signature_scanner.py</code>
Pattern recognition reacts to something new that looks dangerous	A local Random Forest model judges unknown executables by their structure	<code>ml_scanner.py</code>
Adaptive memory gets faster at recognising a threat after meeting it once	Each threat the user confirms by quarantining it is added to a local hash list, so the same file is recognised instantly next time	<code>memory.py</code> , <code>data/memory_hashes.json</code>

4. What it does

Every statement in this section was checked against the source code.

4.1 The four checks

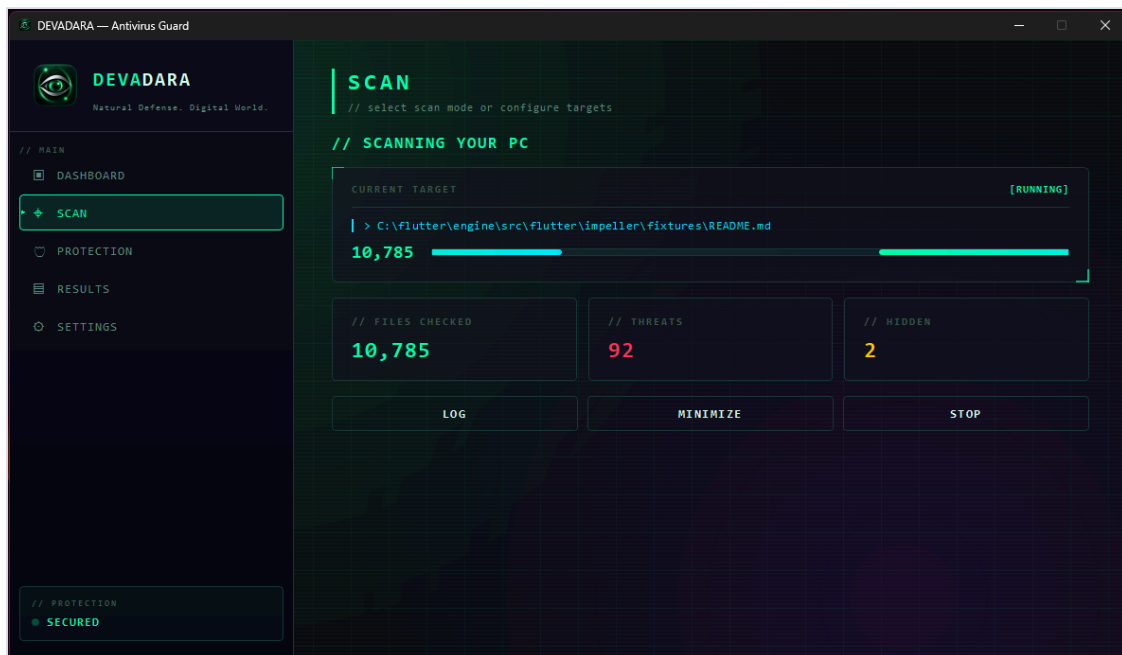
All scanning entry points (manual scans, Downloads and USB auto-scan, the startup sweep, the command-line tools) go through one function: `scan_engine.evaluate_file()` . The checks run in a fixed order. Each one runs only if the previous one did not already decide.

1. **SHA256 hash check.** The file's hash is compared with the bundled known-bad list plus the local "memory" list. An exact match is decisive. The bundled list is deliberately tiny: it only contains the harmless EICAR test file.
2. **YARA rules.** A bundled `yara64.exe` runs five starter rule files (EICAR test, packers, process-injection hints, ransomware-note text, suspicious PowerShell) against **every** file type. This is the only layer that looks at scripts and documents. A rule marked `severity="high"` is decisive. A rule marked `severity="suspicious"` sets a **floor**: later layers can raise the verdict but can never lower it below "suspicious", not even a valid signature.
3. **Authenticode signature check.** For executable-type files only (`.exe` , `.dll` , `.sys` , `.msi` and similar), Windows' own `Get-AuthenticodeSignature` is called through PowerShell. A full, chain-trusted, non-expired signature ends the check as `trusted_signed` and the ML step is skipped. A signature that

does *not* validate but whose claimed publisher is on a small local list can only lower an ML "malware" verdict to "suspicious". It can never mark a file safe.

4. **Local Random Forest model.** For unsigned executables that are still undecided, `pefile` reads **52 raw PE header fields** (DOS header, file header and optional header values). A scikit-learn `RandomForestClassifier` (200 trees, trained once offline and bundled) gives a probability. It is reported in tiers: **malware** at 85% or more, **suspicious** from 60% to 85% (needs a human look), otherwise **safe**.

If one layer fails (a locked file, a missing tool, a timeout), it counts as "found nothing" and the rest of the scan continues.



A scan in progress: the current file, a live count of files checked, and flagged and hidden counts. This scan was over a developer folder, which is where most false alarms came from (see section 6.1).

4.2 Quarantine

Quarantine encrypts the flagged file with Fernet (from the `cryptography` library) using a new random key for each file, and moves it into DEVADARA's data folder. The original file is removed only **after** the encrypted copy has been read back and checked to decrypt to exactly the same bytes. Restore decrypts the file, checks its hash and puts it back. It refuses to overwrite a file that now sits at the original location.

To be honest about the limits: the per-file keys are stored in the quarantine index on the same machine. The point of the encryption is to make the file harmless where it sits, not to keep it secret from someone who already has access to the PC.

4.3 Downloads and USB auto-scan

- **Downloads.** A `watchdog` watcher ignores browser temporary names (`.crdownload` , `.part`) and reacts when the download is renamed to its final name. It also waits until the file size stops changing before it scans.
- **USB.** The app polls for new removable drives (every 2 seconds by default) and scans a new drive when it appears. Some external hard drives report themselves as "fixed" disks, and those are not caught.

Background scans only **log and notify**. They never act on a finding.

4.4 Behavior monitoring

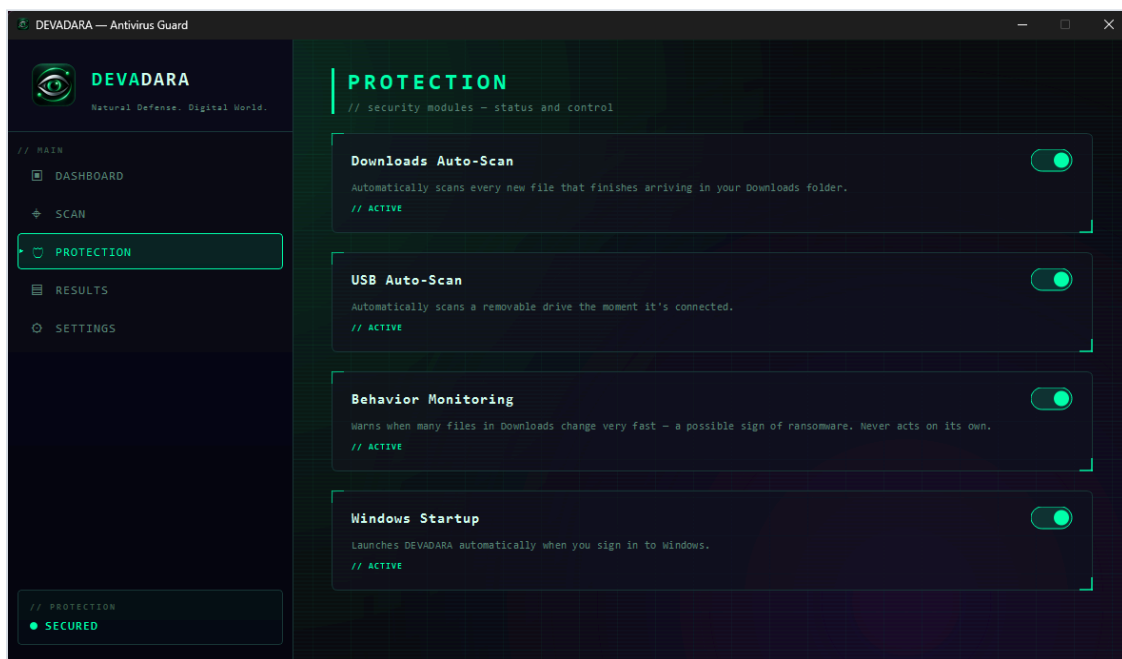
This watches the **Downloads folder only**. If 20 or more file events happen within 5 seconds, it shows a "possible ransomware-like activity" warning. It then waits 15 seconds before it can warn again. It makes a best guess at which program has files open there (this list is often empty on a busy PC). It **never acts on its own**. The user can suspend, resume or kill a listed program, but only through a confirmation dialog. The process guard refuses Windows' own processes, anything under `%SystemRoot%`, DEVADARA itself, and process IDs that have been reused since the alert.

4.5 Self-integrity check

After each release build, a signing tool hashes every program file into `integrity_manifest.json` and signs it with an **Ed25519** private key. Only the public key ships inside the app. On start-up and on "Check now", the app re-hashes its files and reports any that are changed, missing or extra. Genuine DEVADARA files are never quarantined and are labelled "Known Good". Any changed or unknown file in its folder is scanned like any other file.

I checked the released v1.0.0 folder with the app's own check while writing this: **all 1,395 files matched the signed manifest**.

This protects against corrupted or swapped rules, model files, DLLs and similar. It does **not** protect against someone replacing `DEVADARA.exe` itself, admin-level or kernel malware, or a stolen private key.



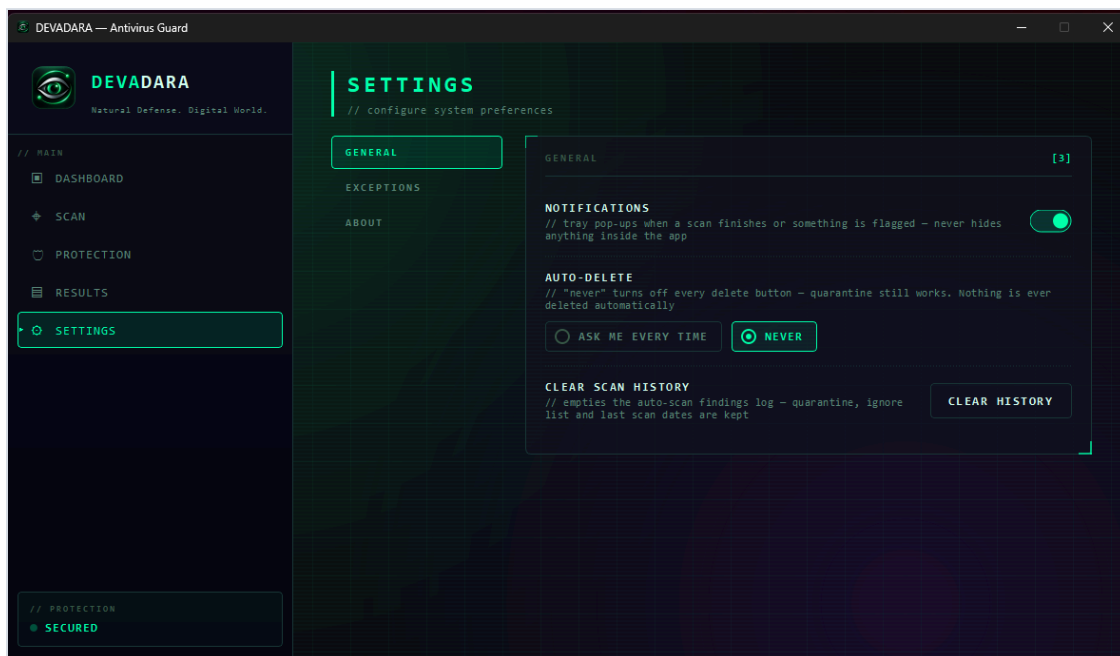
Protection: each background watcher has its own switch. The status shows SECURED, PARTIALLY PROTECTED or NOT PROTECTED, with no made-up percentages.

4.6 Safety rules

These are written down as project rules in the code base and enforced in the GUI:

- **Never auto-delete, never auto-confirm a threat.** Every action (quarantine, restore, ignore, unhide, suspend or kill a program, even "Open in Explorer") goes through one shared confirmation dialog. **Cancel is the default** button. Permanent delete also needs the word `DELETE` typed in.
- **Delete can be switched off.** Settings > Auto-delete has exactly two options, "Ask me every time" and "Never". "Never" disables every delete button. Quarantine and restore still work.

- **"Open in Explorer" never runs the file.** It only highlights it in File Explorer.
- **100% offline.** There is no network code. The "Check for update" and "Report a suspicious file" items in Settings > About only say "coming in a future version".
- **Safe shell-outs.** File paths are passed to PowerShell and `yara64.exe` as separate arguments, never pasted into command text. That way a malicious file name cannot inject commands.



Settings > General with Auto-delete set to "Never": every delete button in the app is disabled.

5. Architecture and how it is built

5.1 Stack

Part	Technology
Language	Python
Desktop UI	PySide6 (Qt), custom theme
ML	scikit-learn <code>RandomForestClassifier</code> , <code>pefile</code> for PE headers, <code>joblib</code> , <code>pandas</code>
Pattern rules	YARA (bundled <code>yara64.exe</code>)
Signatures	Windows Authenticode through PowerShell
File watching / processes	<code>watchdog</code> , <code>psutil</code>
Encryption and signing	<code>cryptography</code> (Fernet for quarantine, Ed25519 for the integrity manifest)
Packaging	PyInstaller, <code>--onedir</code> , no UPX

5.2 Main modules

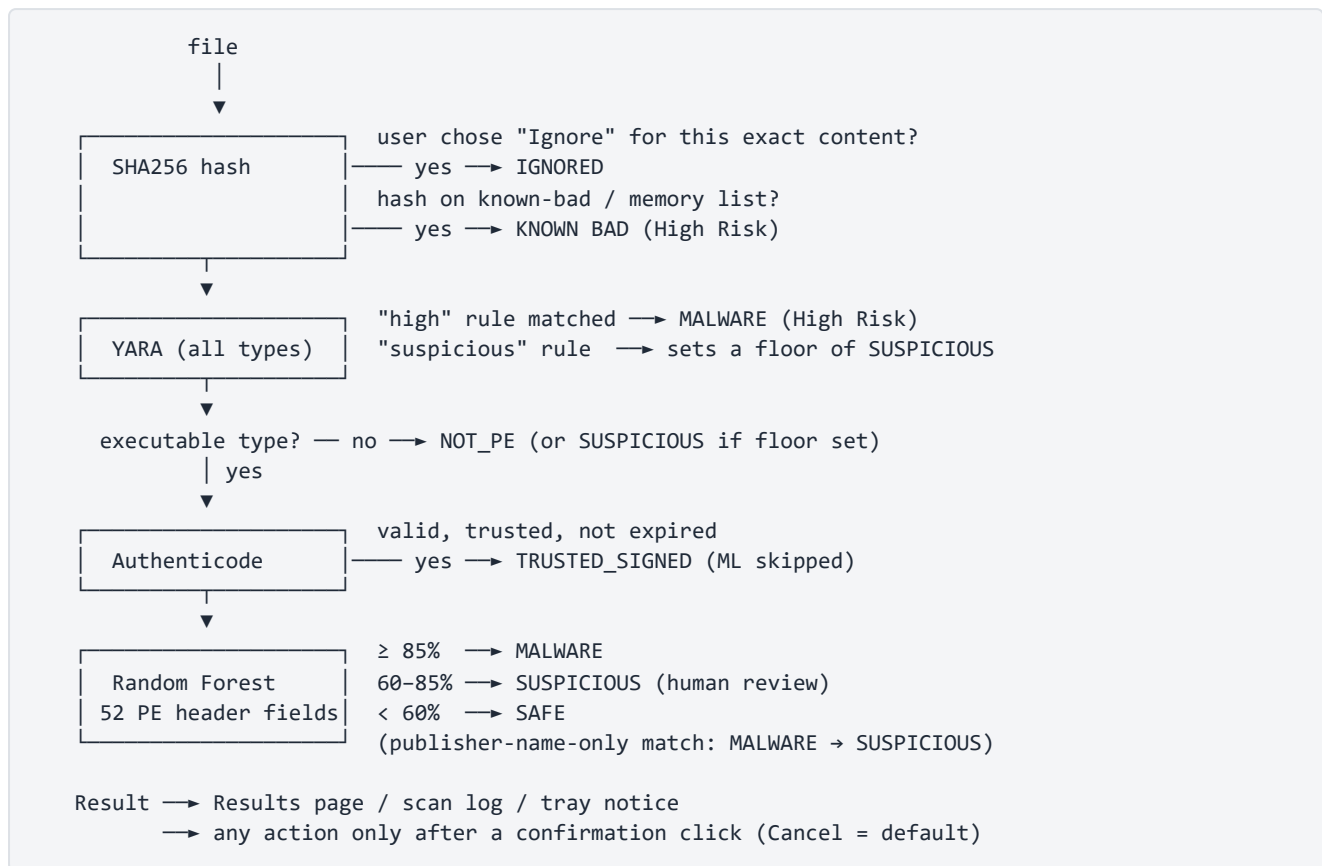
Detection back end (devadara/)

Module	What it does
<code>scan_engine.py</code>	The single four-layer pipeline. Every scanner calls it.
<code>hash_scanner.py</code> , <code>memory.py</code>	SHA256 hashing. Known-bad list plus the growing list of hashes the user confirmed.
<code>yara_scanner.py</code>	Runs <code>yara64.exe</code> and turns rule severity into a verdict or a floor.
<code>signature_scanner.py</code>	Authenticode checks, batched in groups of 40 files per PowerShell launch.
<code>ml_scanner.py</code> , <code>train_model.py</code>	Feature extraction and prediction; a one-time, manual training script.
<code>drive_scanner.py</code>	Walks folders and drives (Quick = executables only, no ML; Full = everything).
<code>persistence_scanner.py</code>	Registry Run keys, scheduled tasks and auto-start services. Their target files go through the pipeline.
<code>process_scanner.py</code>	Simple warning signs in running processes (wrong folder, user-writable location, a document app starting a shell, a missing exe).
<code>downloads_monitor.py</code> , <code>usb_monitor.py</code> , <code>behavior_monitor.py</code>	Background watchers.
<code>quarantine.py</code> , <code>ignore_list.py</code>	Encrypted quarantine; an ignore list tied to both the path and the content fingerprint.
<code>self_integrity.py</code> , <code>app_paths.py</code>	Signed-manifest check; keeps read-only program files apart from writable state (%LOCALAPPDATA%\DEVADARA).

Desktop app (devadara_gui/): `main_window.py` (sidebar, pages, tray icon), `scan_page.py` + `scan_worker.py` (scan queue on a background thread), `auto_scan_controller.py` (the watchers), `results_page.py` (five tabs: Virus, Startup, Hidden, Quarantine, Process), `settings_page.py` , `dialogs.py` (the single confirmation dialog), `process_guard.py` , `integrity_check.py` .

Owner tools (tools/): `integrity_keygen.py` and `sign_manifest.py` . These are not included in the build.

5.3 Scan flow



5.4 Timeline (from git history)

The main repository has 60 commits between 15 September and 12 October 2026.

Date (2026)	What happened
15 Sep	First commit: the hash, YARA, signature and ML pipeline. On the same day I added persistence checks (registry, scheduled tasks, services), a running-process sweep and a full-drive scan.
17–18 Sep	First PySide6 GUI. After a redesign attempt I removed the GUI completely, kept the back end, wrote a <code>CLAUDE.md</code> describing the architecture and rules, and started the GUI again: theme, app shell, Dashboard.
19 Sep	Scan page wired to a real background scan. Signature checks batched (about 8× faster on an 89-file test). Fixed a progress stall: 44.5 s with no progress on <code>System32</code> came down to a 3 s maximum.
10 Oct	Downloads/USB auto-scan, Protection page, tray icon, Windows startup, behavior monitoring in the GUI, and the Results page with the shared confirmation dialog.
11 Oct	Program files separated from writable state, Ed25519 self-integrity check and signing tools, "Known Good / Suspicious / High Risk" labels, Settings > General.
12 Oct	Settings > Exceptions and About, confirmed quarantines remembered as hashes, version 1.0.0, PyInstaller spec, public integrity key, scikit-learn bundled into the exe.
13 Oct	Public <code>devadara-releases</code> repository with the README and screenshots (from that repo's history).
13 Oct	I also published the main project to a private GitHub repository, and cleaned my repositories (see 6.6).

There are no commits between 19 September and 10 October.

6. Problems I ran into and how I solved them

6.1 ML false positives on developer tools

Problem. In my own test on my developer PC, v1.0 marked many files from development tools (for example Flutter, the Android SDK and Git) as High Risk. These were false alarms. The model only looks at 52 header numbers, so unusual but harmless executables can look "malware-shaped" to it. It was trained on a dataset of about 19,600 PE files (the source of the dataset is still to be documented).

What is in place for signed files. The pipeline was designed with this risk in mind from the first commit:

- A file with a valid, trusted Authenticode signature **skips the ML model entirely**.
- The ML result is tiered (85% / 60%) instead of a yes/no answer, so borderline files go to "Suspicious, needs review" instead of "Malware".
- A publisher-name match without a valid signature can only lower "malware" to "suspicious".
- On 19 September I fixed a real bug: paths with parentheses (such as `Program Files (x86)`) were split up wrongly, so signed software was reported as "unverifiable" and could then fall through to the ML step.

What is still open. Many developer-tool binaries are not signed, so they still reach the ML step. I did not change the model for v1.0. Instead I wrote the problem down openly as a known limitation and kept the rule that nothing is ever acted on automatically.

6.2 The exe stuck on "loading detection engine..."

Problem. While testing the built exe, I saw the scan stay stuck at "loading detection engine...". That message is shown in `scan_page.py` at the point where the ML model is loaded.

Cause and fix (confirmed, commit `198e3bc`, 12 Oct). The ML model is a pickled scikit-learn `RandomForestClassifier`. The app never imports scikit-learn directly (only the training script does), so PyInstaller could not see that it was needed and left the tree and forest modules out. I added them as **hidden imports** in `DEVADARA.spec` (`sklearn.ensemble`, `sklearn.ensemble._forest`, `sklearn.tree`, `sklearn.tree._classes`). I also listed `joblib` and `pandas`, because they are imported lazily inside functions and a later refactor could silently drop them.

Lesson. A frozen build can only include what the bundler can see. Anything loaded by name at run time (pickles, plugins) has to be listed by hand.

6.3 File hosting: from Supabase to GitHub Releases

The release zip is about **115 MB** (120,163,138 bytes). Supabase Storage rejected the upload with an "object exceeded the maximum allowed size" error, so I moved the download to **GitHub Releases**. The public README now links to `releases/download/v1.0.0/DEVADARA-1.0.0-win64.zip`.

6.4 Windows SmartScreen: "Unknown publisher"

What users see. DEVADARA is not signed with a commercial code-signing certificate, so Windows SmartScreen can show "Windows protected your PC" / "Unknown publisher". SmartScreen trusts programs

based on a code-signing certificate and download reputation. A new, unsigned app from an individual has neither.

Why I have not bought a certificate yet. I decided not to pay for a certificate for now. A commercial code-signing certificate costs roughly US\$150 to 300 a year. It is on my roadmap.

How users can check the download. The README publishes the zip's SHA256 and tells users not to run the file if the hash does not match:

```
Get-FileHash .\DEVADARA-1.0.0-win64.zip -Algorithm SHA256
# expected: 53D1A6619FDA540E93E1213CC9053A81AD29C83D856D99F46BA57B09667BA865
```

I re-computed the hash of my local release zip while writing this, and it matches the published value. The README also explains that the app's Ed25519 self-check is **not** a Windows code-signing certificate: Windows does not use it to decide whether to trust the app.

6.5 Protecting the signing key and a signed release process

What the code enforces (confirmed in `tools/integrity_keygen.py` and `tools/sign_manifest.py`):

- The private key is **encrypted with a passphrase** (at least 12 characters, typed twice, never echoed or stored).
- The key tool **refuses to save the key inside the repository, a build/dist folder or a signed install**, and never overwrites an existing key. `.gitignore` also blocks `*.pem`, `integrity_private*`, `dist/` and `build/`.
- Only the **public** key is in the app (`devadara/integrity_pubkey.py`).
- The documented backup plan: two offline copies in different places, the passphrase stored separately from the key, the public-key fingerprint written down, and one test restore. If the key is lost or stolen, I make a new key and ship a new build.
- **Signing order:** build first, then sign the finished `--onedir` folder. The signing tool then **re-verifies the folder with the public key built into the app** and fails loudly if anything does not match. I chose `--onedir` because a single-file exe would have to contain its own hash, which is circular.

I backed up the key in more than one place and tested a restore. I also ran a tamper test on a copy of a signed build: after changing one file, the app reported "PROBLEMS FOUND: 1 changed".

6.6 Cleaning my GitHub repositories before making them public

What I could confirm:

- Runtime and per-machine files (settings, scan log, learned hashes, the quarantine index, which holds decryption keys, and old code dumps) were untracked and git-ignored on 10 Oct (commit `168455d`).
- The public README is written with honesty rules: it lists real limitations (including the false positives in 6.1), says plainly that it is not a replacement for a full antivirus, and lists only features that exist.

Before making a repository public, I audited it. In my public web-app repository I found local tool settings containing my Windows user name, and my real email address in the commit history. I made a full backup first, then removed the settings from the whole history, replaced the email with GitHub's private noreply address, and force-pushed the cleaned history. Every file in the latest version stayed identical.

7. Security review (in progress)

An external review of DEVADARA raised four points:

1. **Quarantine ordering:** the order of steps when a file is quarantined.
2. **Safe model loading:** how the ML model file is loaded.
3. **Detection coverage:** how much real-world malware the current layers can actually catch.
4. **Silent failure states:** cases where a check fails quietly and the user is not told.

I am still checking each point. None of them is confirmed yet. Confirmed issues will be fixed in **v1.0.1**, and I will update this section with what I found and what I changed.

8. What I learned

1. **Order matters as much as the checks themselves.** Putting the signature check before the ML model, and treating a YARA "suspicious" match as a floor, changed results more than any single detector did.
2. **False positives are the hardest part.** Catching a test file is easy. Not flagging a developer's own tools is hard, and a security tool that cries wolf teaches people to click "OK" without reading.
3. **Measure before you guess.** The "hang" on 19 September was not a frozen UI. It was 109 PowerShell launches in a row with no progress shown. Timing the steps found the real cause.
4. **Packaging is its own project.** Hidden imports, `--onedir` versus `--onefile`, signing after the build, and SmartScreen all came up only once I tried to ship.
5. **Write the rules down.** Keeping the project's rules in one file (never auto-act, Cancel is the default, one pipeline) kept the design consistent while I was learning and working with an AI assistant.

9. Known limitations

(Based on the public README, plus one more point I found while writing this document.)

- In my own test on a developer PC, v1.0 marked many development-tool files (for example Flutter, Android SDK and Git files) as High Risk. These were false alarms. Please review results before you quarantine anything.
- So far I have only tested it on my own PC.
- The ML model is trained once and bundled. It does not update itself.
- The shipped list of known-bad hashes is very small. Detection relies mostly on YARA rules, signatures and the ML model.
- Behavior monitoring watches only the Downloads folder. Its guess about which program caused the changes is best-effort and can be empty.
- It is not a replacement for Microsoft Defender or other full antivirus products.
- If a check fails (for example a locked file, a missing tool or a timeout), v1.0 treats it as "found nothing" instead of showing "Scan failed". This is one of the points I am fixing in v1.0.1.

10. Roadmap

- **v1.0.1:** fixes for the confirmed points from the security review (section 7).
- **Retraining from confirmed decisions:** use the user's own confirmed quarantines and ignores to improve the model offline, while keeping the "never retrain automatically inside the app" rule.
- **A larger known-bad hash list.**
- **A results / report page** that can be saved and shared.
- **An installer** instead of a zip.
- **Code signing** with a real certificate, to remove the SmartScreen warning.
- Scanning inside archives (zip and others), which is also on the public README.

11. Appendix

Links

- Project page: <https://www.chavindujayawardana.com.lk/devadara>
- Releases and README: <https://github.com/chavindujayawardhana-pixel/devadara-releases>
- v1.0.0 download: <https://github.com/chavindujayawardhana-pixel/devadara-releases/releases/download/v1.0.0/DEVADARA-1.0.0-win64.zip>
- LinkedIn: <https://www.linkedin.com/in/chavindu-jayawardana-81a57a334>
- v1.0.0 SHA256: `53D1A6619FDA540E93E1213CC9053A81AD29C83D856D99F46BA57B09667BA865`

Release process (summary)

1. Set the version in `devadara/version.py`.
2. Build with PyInstaller from `DEVADARA.spec` (`--onedir` , no console window, no UPX, only read-only program files bundled; no runtime state, training data or owner tools).
3. Sign the finished folder with `tools/sign_manifest.py` , using the offline, passphrase-protected private key. This writes `integrity_manifest.json` and `integrity_manifest.sig` next to the exe.
4. The signing tool re-verifies the whole folder with the public key built into the app. The release stops if anything does not match.
5. Zip the signed folder and compute its SHA256.
6. Upload the zip to GitHub Releases and publish the SHA256 in the README.
7. Any rebuild means signing again.