PROJECT ABSTRACT

Malware Detection from PE File Headers using Machine Learning

A static machine-learning classifier that judges Windows executables from their PE headers alone — no execution, no sandbox. Parsed-header features in the EMBER 2,381-dimensional format feed a gradient-boosted tree ensemble that outputs a calibrated malware probability with per-feature explanations. Delivered built-to-order with a PE parser, training notebook, trained model, interactive web demo, report, PPT and viva kit.

01. Title

Malware Detection from PE File Headers using Machine Learning.

02. Introduction / Background

Signature-based antivirus misses new malware variants, while dynamic analysis — executing the sample in a sandbox — is slow, resource-heavy, and easy for malware to detect and evade. Static machine-learning detection occupies the middle ground: it classifies a Windows executable from its PE (Portable Executable) headers without ever running it. The PE format exposes strong statistical signals: packed binaries show section entropy near 8.0 bits/byte, droppers import suspiciously few APIs, and legitimate software is usually Authenticode-signed. The EMBER dataset (Elastic Malware Benchmark for Empowering Researchers, 2018) made this task reproducible by publishing 1.1 million labeled PE files — 800,000 for training, 200,000 for testing — together with a standard 2,381-feature extraction pipeline. This project builds the complete pipeline on that benchmark: a real PE parser, EMBER-style feature engineering, a gradient-boosted classifier, honest held-out evaluation, and an interactive demo that explains every verdict.

03. Problem Statement

Students learning security ML face two gaps: tutorials classify toy CSVs that teach nothing about real malware, while production AV internals are proprietary and opaque. There is no open, explainable reference build that goes from raw .exe bytes to a justified verdict. This project fills that gap with an end-to-end static detector: parse the actual headers, engineer the real feature set, train on a published benchmark, evaluate honestly on held-out data, and present each decision with the header anomalies that drove it — so the viva can probe binary formats, feature engineering and model calibration, not just API calls.

04. Objectives

- Implement a PE parser that extracts DOS, COFF and optional-header fields, the section table, entry point, subsystem, DLL characteristics and per-section entropy from real Windows executables.
- Build the EMBER-style feature pipeline: parsed-header fields, section statistics, import-table counts, byte-value histogram, byte-entropy histogram and printable-string features (2,381 dimensions).
- Train a gradient-boosted decision-tree ensemble (LightGBM-style) on the EMBER training split with validation-based threshold tuning.

- Evaluate honestly on the held-out EMBER test split: ROC AUC, precision/recall at the operating threshold, and a confusion matrix — design target ROC AUC ≥ 0.98, reported from the actual training run.
- Build an interactive web demo with a malware-probability gauge, per-feature contribution chart and batch analysis table, including real in-browser PE parsing of dropped files.
- Document PE internals, feature engineering, methodology and honest limitations in the report, PPT and viva Q&A.;

05. Proposed Methodology

The pipeline is purely static. First, the parser validates the MZ magic and PE signature, then walks the COFF file header and optional header to extract machine type, section count, entry point, image base, SizeOfImage, subsystem and DLL characteristics. Each section header is parsed and the .text section's raw bytes are sampled to compute Shannon entropy. Parsed values are encoded into the 2,381-dimensional EMBER-style vector. A gradient-boosted tree ensemble is trained on the 800,000-file EMBER training split, with early stopping on a validation slice and threshold selection on the precision/recall trade-off. A calibration step maps raw scores to probabilities, which are banded into verdicts: below 0.35 benign, 0.35–0.65 suspicious, above 0.65 malicious. Per-feature contributions are computed for every verdict to drive the demo's explainability chart. Final evaluation runs once on the 200,000-file held-out test split.

06. System Architecture / Working

Data flows in one direction: PE file → parser → feature vector → trained ensemble → calibrated probability → verdict + explanations. The Python pipeline (pefile, NumPy, LightGBM) handles training and batch scoring; the web demo re-implements the parser in JavaScript so a dropped .exe is analyzed entirely client-side with nothing uploaded. The demo's in-browser scorer uses illustrative weights on eight headline features, while the full 2,381-feature model runs in the Python pipeline — a separation the report states explicitly. The gauge, contribution chart and batch table are rendered on HTML5 canvas from the same scoring function, so the demo's behavior matches its documentation.

07. Hardware / Software Requirements

- Software: Python 3.10+, LightGBM, pefile, NumPy, pandas, scikit-learn, Jupyter Notebook; any modern browser for the demo (runs offline).
- Dataset: EMBER 2018 (public download, ~1.1M labeled PE files with fixed train/test split) — storage approximately 15–20 GB for the feature vectors.
- Hardware: training is feasible on a laptop CPU (expected several hours for the full split); inference is approximately 50 ms per file on CPU. No GPU required.
- No special hardware: the project is pure software; the demo needs no network access after download.

08. Expected Outcomes

- A trained gradient-boosted malware classifier with a design-target ROC AUC ≥ 0.98 on the held-out EMBER test split, reported honestly from the training run executed for the order.
- A real PE parser (Python and in-browser JavaScript) verified against genuine Windows executables.

- An interactive demo: drop-a-file analysis, malware-probability gauge, per-feature contribution chart and a six-sample batch table covering malicious, suspicious and benign cases.
- Complete documentation: report PDF (PE internals, feature engineering, methodology, results), PPT and viva Q&A.;
- A calibrated three-band verdict system (benign / suspicious / malicious) with documented thresholds.

09. Applications

- Teaching static malware analysis: PE format, section entropy, packing indicators and import-table signals.
- Triage aid for downloaded executables — a fast first opinion before deeper dynamic analysis.
- SOC analyst training on interpreting model explanations rather than trusting bare labels.
- Baseline for further research: retraining on EMBER2024, adding new feature families, or comparing tree ensembles with neural models.

10. Future Scope

- Retrain on EMBER2024 (64 weeks of recent files plus a challenge set of evasive samples) to cover newer malware families.
- Add emulation-assisted features (API call sequences from lightweight emulation) while keeping analysis static.
- Extend parsing to ELF and Mach-O binaries for cross-platform coverage.
- Adversarial robustness study: header-spoofing attacks and defenses such as monotonic feature constraints.
- Package the scorer as a browser extension or OS context-menu scanner for one-click file triage.