Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

How Far Does Entropy Go? Duplicates, Confounding, Drift and Padding in Entropy-Based Detection of Malicious PE Files

Niccolò Rinaldi

niccolo.rinaldi@unifi.it

University of Florence

Giulio Bello

University of Pisa

Chiara Tani

University of Florence

Research Article

Keywords: malware detection, static analysis, Shannon entropy, packing, dataset bias, adversarial evasion

Posted Date: November 9th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11309969/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

How Far Does Entropy Go? Duplicates, Confounding, Drift and Padding in Entropy-Based Detection of Malicious PE Files

Abstract

Shannon entropy is a common static indicator of packed or encrypted Windows executables. We measure how much the entropy of the first PE section, alone or with the size fields of the same header, tells about maliciousness, on 43,139 PE section headers and 799,920 labelled EMBER 2018 samples. In both datasets 64% of malicious files share their first section with other files, so random cross-validation largely measures memorisation: grouping identical sections lowers the AUC of an entropy-only model from 0.87 to 0.76 and from 0.77 to 0.70. On the EMBER temporal split, 97.5% of the detections of this model at 1% false positives are sections already seen in training; on unseen sections it detects 0.9% of malware, and 7.9% with the size fields added. A strong size confound in the first dataset does not replicate on EMBER, and empty-on-disk first sections are also frequent in benign software. Using standard bounds on the entropy of a section after appending arbitrary bytes, we compute from header data alone which detections are evaded whatever the section content. On the deduplicated dataset, appends guarantee evasion for 52–92% of the samples detected by an entropy-and-size model, and for 96–98% if the attacker can also use padding of benign-looking entropy; appends that stay within the alignment slack or keep the header consistent achieve only 6–24%. Since information content, size times entropy, never decreases under appends, a monotone model is provably immune, but its AUC falls to 0.52–0.62. Adversarial training reduces evasion but does not restore detection.

Keywords: malware detection, static analysis, Shannon entropy, packing, dataset bias, adversarial evasion

1 Introduction

Static analysis inspects an executable without running it. It is the safest and cheapest stage of malware triage, and the setting in which most machine-learning (ML) malware detectors operate [1–4]. Its main weakness is obfuscation: transformations that change the bytes of a program without changing its behaviour [5, 6]. Run-time packing is one of the most common obfuscation techniques adopted by malware authors [7], it is frequent even among low-entropy malware [8], combinations of obfuscation and encryption can still evade modern antivirus software [9], and cryptography is used in malware obfuscation [10].

Shannon entropy is the most common feature used to react to packing and encryption. Compressed and encrypted data are close to uniformly distributed over the 256 byte values, so their entropy approaches the maximum of 8 bits/byte, while native code and text sit lower [11]. Entropy is cheap to compute, appears in many static feature sets [4], and continues to be proposed as a detection feature, including in more elaborate forms such as structural, multiscale and Kolmogorov entropies [12, 13]. At the same time, evasion attacks against static PE detectors are now routine: reinforcement-learning agents apply functionality-preserving modifications to PE files, among which appending bytes to the unused space at the end of sections [14], and evade a gradient-boosted tree detector in about half of the cases [15].

In this paper we ask a simple question: *how much does the entropy of the first PE section, alone or with the size fields of the same section header, separate malicious from benign executables, and how easily can that separation be removed?* We find that the answer depends less on the classifier than on four issues that are easy to overlook: duplicated sections, a size confound, temporal drift, and the ease of padding. We study

them on two datasets of different origin and size, and complement the measurements with bounds on the entropy of padded sections that hold for any section content.

Our contributions are:

- **Duplicates and leakage.** An audit of a public dataset of PE section headers [16] and of EMBER 2018 [4], showing that in both 64% of malicious files share their first section with at least one other file, and a quantification of the resulting leakage under random cross-validation, including near-duplicates and malware families.
- **Drift.** An evaluation on the temporal split of EMBER showing that almost all detections of an entropy-only model on new malware are sections already seen in training, and that performance on unseen sections is close to useless at low false positive rates.
- **Confounding and packing traces.** A size-matched design that separates the contribution of section size from that of entropy, showing that a strong size confound in one dataset does not replicate in the other; and evidence that empty-on-disk first sections are a packing trace but not a malice trace.
- **Padding, with guarantees.** An application of standard entropy bounds for mixtures [17] to a section after appending arbitrary bytes (Proposition 1), and the resulting monotonicity of its information content (Proposition 2). The bounds let us compute, from header data alone, which detections are evaded *whatever the section content*; the monotonicity yields a classifier that is provably immune to appends, whose cost we measure. We also evaluate two attacker variants, with the virtual size unchanged or growing, padding of benign-looking entropy, appends small enough to keep the section header unremarkable, and adversarial training.
- **Reproducibility.** Code, results and an extraction script for EMBER are released (see Data and code availability).

Section 2 reviews related work and Section 3 presents the entropy results and the threat model. Section 4 describes and audits the datasets and Section 5 the experimental setup. Section 6 reports the experiments, Section 7 discusses limitations and Section 8 concludes.

2 Related work

2.1 Static ML malware detection

Data-mining approaches to the static detection of unknown executables date back to Schultz et al. [1] and Kolter and Maloof [2]. Later work moved to larger feature sets and deep models [3], and to public benchmarks such as EMBER [4], whose feature vector includes byte histograms, byte-entropy histograms and section-level information. Several detectors rely on PE header fields: Belaoued and Mazouzi [18] selected Optional Header fields with a chi-square test, and Singh et al. [19] studied feature representation and selection for PE-header detectors. Entropy-based features range from the block entropy of Lyda and Hamrock [11] to structural entropy [12] and multiscale and Kolmogorov entropies [13]. We study the simplest member of this family, the entropy of one section, because it is the building block of the others and because its limits can be stated exactly.

2.2 Entropy, packing and packer detection

Lyda and Hamrock [11] computed entropy over 256-byte blocks and showed that the average entropy of plain text, native executables, packed executables and encrypted executables increases in that order (4.347, 5.099, 6.801 and 7.175 bits/byte respectively). They proposed thresholds on the average and highest block entropy (6.677 and 7.199 bits/byte) to flag packed or encrypted malware, and noted that the method can produce both false positives and false negatives. Ugarte-Pedrero et al. [7] carried out a longitudinal study of the complexity of run-time packers, Liţă et al. [20] surveyed the anti-emulation techniques of modern packers, and Kancherla et al. [21] addressed packer identification using byte plots and Markov plots. Bernardinetti et al. [22] built a packer designed for automated evasion of antivirus products. Mantovani et al. [8] questioned the assumption behind entropy-based packing detection: analysing 50,000 low-entropy Windows malware samples, they found that over 30% of them use some form of run-time packing, and that even a state-of-the-art classifier based on static features cannot reliably tell whether a low-entropy sample is packed. We look at entropy from the opposite

direction: rather than detecting packing, we measure how much first-section entropy tells about maliciousness, and how easily that signal can be removed.

2.3 Evasion of static detectors

Attacks against ML malware detectors are usually grouped by the attacker's knowledge of the model. Gradient-based attacks require a differentiable model with known parameters [23]. Score-based black-box attacks exploit the classifier's output score, for example through genetic programming [24]. Attacks against binary-output detectors can train a surrogate model and attack it instead [25], or learn a policy of file modifications by reinforcement learning, as in Anderson et al. [14], whose action set includes appending bytes to the unused space at the end of sections. Kozák et al. [15] trained such agents with proximal policy optimisation and reached an evasion rate of 53.84% against a gradient-boosted decision tree detector, and Louthanová et al. [26] compared gradient-based, evolutionary and reinforcement-learning generators of adversarial PE malware experimentally. On the defensive side, Crasta and Kumar [27] studied how to make deep-learning malware detectors more robust to adversarial attacks. These works search the space of modifications empirically. Our padding analysis studies one modification of this kind, appending bytes to a section, but characterises its effect on the entropy feature exactly, which lets us state which detections are evaded for every possible section content.

2.4 Evaluation bias

Aghakhani et al. [28] showed that static-feature classifiers can separate packed benign from packed malicious samples when packers are evenly distributed across the two classes, but that this ability does not generalise to unseen packers, with false positive rates above 40% in some settings. Arp et al. [29] list *sampling bias* and *spurious correlations*, i.e. artefacts unrelated to the security problem that create shortcuts for separating the classes, among ten common pitfalls of machine learning in security, and Pendlebury et al. [30] showed that spatial and temporal biases in the experimental setup can inflate the reported performance of malware classifiers. Earlier, Rossow et al. [31] reviewed 36 malware papers and proposed guidelines for the correctness, transparency, realism and safety of malware experiments and datasets. That sampling and temporal biases can inflate results is therefore well known [29, 30]; our contribution is to measure how much they do for one widely used feature, with a duplicate audit, a size-matched design and a temporal evaluation.

3 Entropy of padded sections and threat model

3.1 Entropy and packing

Let X be a sequence of bytes and $p(i)$ the relative frequency of byte value $i \in \{0, \dots, 255\}$. The Shannon entropy [32] of X is

$$H(p) = -\sum_{i=0}^{255} p(i)\, \log_2 p(i), \tag{1}$$

which ranges from 0 (a single repeated byte) to 8 bits/byte (uniform distribution).

Packers such as UPX compress the original code and data and add a small decompression stub that restores them in memory at run time. This leaves two traces in the file. First, compressed or encrypted regions have entropy close to 8. Second, the section layout changes: UPX, for example, produces a first section with no bytes on disk and a large virtual size, which the stub fills at run time. Entropy captures only the first trace; the second is visible in the section header.

3.2 Appending bytes to a section

Suppose k bytes with byte distribution r are appended to a section of n bytes with byte distribution p. The new distribution is the mixture

$$q = \lambda p + (1 - \lambda)\, r, \qquad \lambda = \frac{n}{n + k}. \tag{2}$$

Let $h(\lambda) = -\lambda \log_2 \lambda - (1 - \lambda) \log_2 (1 - \lambda)$ be the binary entropy function.

The following bounds are standard: they are the elementary inequalities $H(Y \mid Z) \le H(Y) \le H(Y, Z)$ [17, Ch. 2] applied to the mixture q. We state them in the form we use because, together with the header fields, they determine

the entropies an attacker can reach without any knowledge of the section content.

Proposition 1 (Entropy bounds) *For every p and r,*

$$\begin{aligned} H(q) &\geq \lambda H(p) + (1-\lambda)H(r), \\ H(q) &\leq \lambda H(p) + (1-\lambda)H(r) + h(\lambda). \end{aligned} \tag{3}$$

In particular, for padding with a single byte value ($H(r) = 0$),

$$\lambda H(p) \leq H(q) \leq \lambda H(p) + h(\lambda), \tag{4}$$

and the upper bound is attained when the padding byte does not occur in the section.

Proof We include the argument for completeness. Draw a byte uniformly from the padded section and let $Z \in \{0, 1\}$ indicate whether it comes from the original section ($Z = 1$, probability λ) or from the appended bytes. The byte Y has distribution q, and conditionally on Z it has distribution p or r. Since conditioning does not increase entropy, $H(q) = H(Y) \geq H(Y \mid Z) = \lambda H(p) + (1-\lambda)H(r)$. Moreover $H(Y) \leq H(Y, Z) = H(Z) + H(Y \mid Z) = h(\lambda) + \lambda H(p) + (1-\lambda)H(r)$, with equality if and only if Z is a function of Y, i.e. if no byte value occurs both in the section and in the appended bytes. □

Equation (4) depends only on $H(p)$, n and k, which are recorded in the section header data. It therefore lets us study padding without access to the binaries: if a classifier assigns a benign score to every entropy value in the interval, the attack succeeds whatever the content of the section.

Let $B = nH(p)/8$ be the *information content* of the section, in bytes: the size of an ideal order-0 compression of its content. The lower bound of Proposition 1 has an immediate consequence for B.

Proposition 2 (Information content cannot decrease) *For every p, r and $k \geq 0$,*

$$(n+k)\,H(q) \;\geq\; n\,H(p) + k\,H(r) \;\geq\; n\,H(p). \tag{5}$$

Proof Multiply the first inequality of Equation (3) by $n+k$ and use $\lambda(n+k) = n$ and $(1-\lambda)(n+k) = k$. □

Appending bytes can therefore lower the entropy of a section but never its information content, nor its raw size. This gives a simple sufficient condition for robustness.

Corollary 1 (Append-robust classifier) *Let f be a classifier score that is non-decreasing in each of B, the raw size n and the virtual size v. If an attacker can only append bytes to the section, so that n does not decrease and v does not decrease, then f cannot be lowered, and no malicious section detected at a threshold τ can be moved below it.*

The corollary is a sufficient condition, not a characterisation: the set of reachable feature values is smaller than the orthant it implies, because of the upper bound in Equation (3). We use it in Section 6.5 to measure what provable robustness to appends costs in detection performance.

3.3 Threat model

The attacker controls a malicious PE file that a static detector flags, and wants it classified as benign. The detector uses the features of the first section header described in Section 5, and the attacker can evaluate it as often as needed. The attacker may only append bytes to the raw data of the first section, updating the raw size, and may choose the number and value of the appended bytes but not the content of the section. We consider two variants, which differ in how they treat the virtual size.

In the *fixed* variant the virtual size is unchanged. The PE format allows the raw size of a section to exceed its virtual size [33], and bytes beyond the virtual size are not mapped into memory. Enlarging the raw data of the first section shifts the file offsets of the following sections but not their relative virtual addresses, so the memory image of the program does not change. It preserves functionality for any amount of padding. A large excess of raw over virtual size is, however, rare: raw size exceeds virtual size by more than 4 KiB in at most 0.1% of the files of either dataset, so a detector that checks the consistency of the two fields could flag it. We therefore also consider the fixed variant restricted to an excess of at most 4 KiB.

In the *growing* variant the virtual size grows by k as well, as when the appended bytes are mapped into memory. Without further changes this is possible only while the enlarged section still ends before the next section begins in memory, i.e. within the slack left by the section alignment; the section-padding action of Anderson

et al. [14] appends bytes to this unused space. Larger appends would require moving the following sections in memory and updating every reference to them, which does not preserve functionality in general. Neither dataset records the address of the second section, so we estimate the slack as $\lceil v/4096 \rceil \cdot 4096 - v$, assuming a section alignment of 4 KiB and contiguous sections. The slack is a small fraction of the section size: for the malicious sections detected in the first dataset its median is 0.3–0.4% of the raw size. We report the growing variant both for all padding fractions, a scenario that is not always reachable in a real binary, and restricted to this slack.

In both variants the attacker chooses the padding content. Besides a single repeated byte, which minimises the entropy of the padding, we consider padding of entropy $H(r) \in \{6.0, 6.25, 6.5\}$ bits/byte, the range where benign first sections concentrate (Figures 1 and 2); such padding cannot be removed by discarding runs of identical bytes.

We work in feature space: we do not modify binaries (see Section 7).

4 Datasets and audit

4.1 PE Section Headers

The *PE Section Headers* dataset by Oliveira [16] describes each PE file through its MD5 hash and one section header: raw size on disk, virtual address, virtual size and the entropy of the section's bytes. In 93.2% of the records the section starts at relative virtual address 4096, i.e. it is the first section, typically `.text`; the feature is therefore the entropy of the first section, not of the whole file. The dataset contains 43,293 records: 41,568 malicious and 1,725 benign. We removed 24 exact duplicate records (all benign) and the 5 files that appear with both labels, and kept one record per file (the one with the lowest virtual address, removing 120 records). This leaves 43,139 files: 41,444 malicious and 1,695 benign.

4.2 EMBER 2018

EMBER 2018 [4] provides raw static features for one million PE files, with malware family labels produced by AVClass [34] and the month in which each file first appeared. We extracted, for each labelled file, the name, raw size, virtual size and entropy of the first entry of the section table, together with the label, the month and the family; 80 labelled files without sections were skipped. This gives 799,920 files: 400,000 malicious and 399,920 benign. The official split assigns 600,000 files that appeared up to October 2018 to training and 200,000 files from November and December 2018 to testing. EMBER does not record the virtual address of the section.

4.3 Duplicated first sections

Different files can share the same first section, for instance variants of a family built from the same code, or files produced by the same packer or installer. We call two records identical when they have the same entropy, raw size and virtual size, i.e. the same feature vector. Table 1 shows that duplication is massive and remarkably similar in the two datasets: in both, 63.9% of malicious files share their first section with at least one other file, and the number of distinct first sections is only 39.3% and 39.6% of the number of malicious files. In the first dataset the largest group contains 6,046 files (14.6% of the malicious class); in EMBER it contains 34,943 malicious files with entropy 7.96 bits/byte, almost all from a single family (*xtrat*), and is visible as a spike in Figure 2. Benign files are more diverse. Some first sections occur in both classes: 19 sections covering 1,796 files in the first dataset, and 2,735 covering 118,072 files in EMBER.

With random cross-validation, copies of the same section fall in both the training and the test folds, which inflates the measured performance (Section 6.1). For the first dataset we therefore also build a *section-level* dataset with one record per distinct first section, removing the sections that occur in both classes, whose label is not determined by the features. It contains 17,605 sections: 16,274 malicious and 1,331 benign.

4.4 Empty first sections

In the first dataset, 223 records have an entropy of exactly 0, all of them malicious. It is tempting to treat them as corrupted and discard them. However, 220 of them have a raw size of 0, almost all start at virtual address 4096 and they have a large virtual size (median 323 kB over the 219

Table 1 The two datasets after cleaning. Entropy in bits/byte. Two first sections are identical if they have the same entropy, raw size and virtual size.

	PE Section Headers		EMBER 2018	
	Benign	Malicious	Benign	Malicious
Files	1,695	41,444	399,920	400,000
Distinct first sections	1,350	16,293	272,126	158,492
Distinct first sections / files	79.6%	39.3%	68.0%	39.6%
Files whose first section no other file has	77.7%	36.1%	63.5%	36.1%
Files in the largest group	302	6,046	8,224	34,943
First sections with no bytes on disk	0	219	16,302	29,244
Median entropy of the first section	6.33[a]	6.53[a]	6.27	6.55
Share with entropy ≥ 7.2	0.7%[a]	16.8%[a]	4.6%	22.9%
Median raw size	90 kB[a]	190 kB[a]	80 kB	94 kB

[a]Section-level dataset (one record per distinct first section).

cleaned files): they are first sections with no bytes on disk, the layout produced by packers such as UPX. Their entropy is 0 because there is nothing to measure. At the section level they are 129 records, all malicious, and we keep them.

EMBER shows that this layout is a packing trace, not a malice trace: 45,546 files have an empty first section, and 16,302 of them are benign. The most common name of these sections is `UPX0`, in 12,057 benign and 21,625 malicious files. The fact that all empty first sections are malicious in the first dataset is therefore a property of how that dataset was collected, an instance of the sampling bias discussed by Arp et al. [29].

4.5 Distributions

Figure 1 and Figure 2 show the distributions of first-section entropy. In both datasets benign sections concentrate around 6–6.7 bits/byte, while malicious sections have a slightly higher median and heavier tails, both towards 8 bits/byte and towards 0. In the section-level data, 57.1% of benign sections lie between 6.0 and 6.5 bits/byte, and 16.8% of malicious sections lie above 7.2 bits/byte and 24.1% below 6.0. Malicious sections are larger than benign ones, by a factor of about two in the first dataset (median raw size 190 vs. 90 kB) but only by about 17% in EMBER (94 vs. 80 kB).

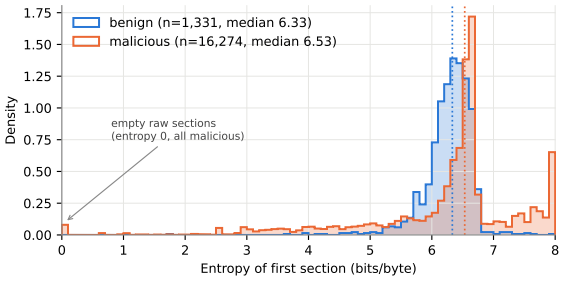


Fig. 1 Distribution of first-section entropy on the section-level PE Section Headers data (density, 0.1 bits/byte bins). Dotted lines mark the class medians. The bar at 0 consists almost entirely of empty raw sections (129 of its 130 sections), all malicious

5 Experimental setup

5.1 Features and models

We compare three feature sets: **E**, the section entropy alone; **S**, three structural features from the same header, $\log(1 + \text{raw size})$, $\log(1 + \text{virtual size})$ and their difference (the log ratio of virtual to raw size); and **E+S**, all four. For the robustness analysis we also use $\log(1 + B)$, the information content of Proposition 2. The models are a decision tree of depth 2 (DT), for interpretability; a random forest (RF, 100 trees, at least two samples per leaf); and gradient-boosted trees (XGBoost, 300 trees of depth 4, learning rate 0.05, subsampling 0.8).

5.2 Protocols

For the first dataset, experiments use the section-level data unless stated otherwise. In the *balanced*

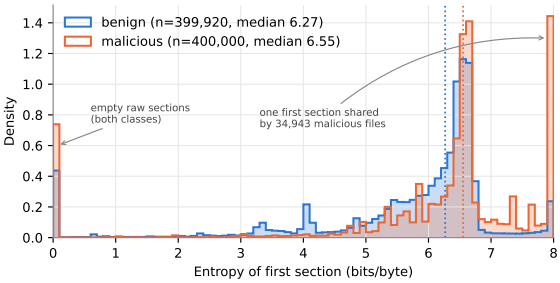


Fig. 2 Distribution of first-section entropy in EMBER 2018 (all labelled files, density, 0.1 bits/byte bins). Dotted lines mark the class medians. The malicious spike at 7.9–8.0 bits/byte is dominated by one first section shared by 34,943 files; the bar at 0 contains empty raw sections of both classes

protocol, all 1,331 benign sections are paired with an equally sized random sample of malicious sections, redrawn ten times; each draw is evaluated with stratified 10-fold cross-validation, and we report mean and standard deviation over the draws. In the *full-data* protocol, all 17,605 sections are evaluated with stratified 10-fold cross-validation, and we report only threshold-free metrics: ROC AUC and the true positive rate (TPR) at a fixed false positive rate (FPR). The TPR at 1% and 5% FPR is the most relevant metric for a triage tool, which must not flood analysts with benign files. For EMBER we build the same section-level data from the training split (329,855 distinct first sections: 122,600 malicious and 207,255 benign, after removing 1,964 sections that occur in both classes). Since these sets are large and only mildly imbalanced, we use the full-data protocol with stratified 5-fold cross-validation instead of the balanced protocol, together with the official temporal split.

5.3 Statistics

To compare E and E+S in the balanced protocol we use a paired Wilcoxon signed-rank test [35] over the ten draws; with ten pairs the smallest attainable two-sided p-value is 0.002, and since the draws share the benign sections the test should be read as indicative. In the full-data protocol we use the DeLong test for two correlated ROC curves [36], in the fast implementation of Sun and Xu [37], on the pooled out-of-fold scores. Confidence intervals (95%) for AUC and TPR are obtained by a paired bootstrap of the out-of-fold scores, stratified by class (2,000 resamples for the full-data protocol, 1,000 for the leakage check); for evasion rates we use Wilson intervals. Both the DeLong test, which treats the pooled out-of-fold scores of different fold models as a single score vector, and the bootstrap, which resamples predictions without retraining, ignore the variability due to training, so their p-values and intervals should be read as approximate. On the EMBER temporal test set (500 bootstrap resamples) the 95% intervals are at most 0.007 wide for AUC and at most 0.015 wide for TPR at 1% FPR, both on all test files and on those with unseen first sections; we report them in the code repository and omit them from the tables.

5.4 Use of large language models

In accordance with the journal's policy, we report that a large language model (Claude Sonnet 5.5, Anthropic) was used to assist with writing and debugging the analysis code, improving the language of the manuscript, and discussing and planning the experiments. The model was not used to make scientific decisions or interpret the results. All scientific decisions were made by the authors, who independently checked the code, re-ran the experiments, verified the results, and take full responsibility for the content of the manuscript.

6 Results

6.1 Duplicates inflate the measured performance

Table 2 evaluates XGBoost on all files with random stratified folds and with stratified group folds, in which records of the same group are never split between training and test. Grouping identical first sections lowers the AUC of the entropy-only model from 0.872 to 0.761 on the first dataset and from 0.775 to 0.696 on EMBER, and its TPR at 1% FPR from 36.8% to 7.3% and from 21.1% to 5.2%. The bootstrap intervals of the random and grouped estimates do not overlap. Results obtained with random splits therefore largely measure the ability to recognise sections already seen in training.

Exact deduplication is not sufficient. Grouping near-duplicates (same raw and virtual size, entropy equal to two decimals) lowers the AUC of E+S on the first dataset further, from 0.830 to

0.803, and grouping all sections of the same raw size to 0.786. On the section-level data of the first dataset, which is already free of exact duplicates, the same groupings lower the AUC of E+S only slightly (from 0.879 to 0.864 and 0.851), so its results in the following sections are not driven by residual near-duplicates; on EMBER they matter more (Section 6.2). On EMBER, grouping near-duplicates changes little with respect to exact grouping (AUC 0.772 vs. 0.769 for E+S), while grouping malicious files of the same AVClass family lowers it to 0.731, with a TPR of 12.4% at 1% FPR.

6.2 Classifiers on distinct sections

On the section-level data of the first dataset, very high entropy is a precise but narrow signal: a threshold of 7.2 bits/byte flags 16.8% of malicious sections at an FPR of 0.68%, and 7.5 bits/byte flags 13.0% at 0.30%. The depth-2 decision tree trained on entropy labels the range between 5.2 and 6.8 bits/byte as benign and everything outside it as malicious, recovering the two tails of the malicious distribution in Figure 1.

Table 3 reports the balanced protocol. With entropy alone, the ensembles reach 69–71% accuracy and an AUC of 0.75–0.78, and XGBoost detects 22.2% of malicious sections at 1% FPR. The structural features alone reach a similar AUC (0.74), but the two sets are complementary: with E+S the random forest reaches 79.1% accuracy, an AUC of 0.876 and a TPR of 45.0% at 1% FPR, and XGBoost 78.0%, 0.865 and 36.2%. For both ensembles E+S outperforms E in all ten draws in accuracy, AUC and TPR at 1% FPR (Wilcoxon $p = 0.002$, the minimum attainable). The depth-2 tree does not benefit, because its best splits are on entropy in both cases.

Table 4 confirms these results on all 17,605 sections. With XGBoost, the AUC rises from 0.786 (E) to 0.879 (E+S) and the TPR at 1% FPR from 24.1% to 34.8%; Figure 3 shows the ROC curves. For both models the difference in AUC between E+S and E is highly significant (DeLong test, $z = 26.4$ for RF and $z = 20.7$ for XGBoost, $p < 10^{-90}$). An ablation in the balanced protocol (Table 5) shows that raw size or virtual size alone recovers most of the gain of E+S (random forest AUC 0.864 and 0.863, against 0.751 for E and 0.876 for E+S), while the size ratio, the feature most directly linked to packing, helps less (0.829).

The same pattern holds on the distinct first sections of EMBER (Table 4, Figure 3). Entropy alone reaches an AUC of 0.702 (RF) and 0.744 (XGBoost), and a TPR of 15.0% and 13.3% at 1% FPR; adding the structural features raises the AUC to 0.900 and 0.834 (DeLong test, $z = 208$ and $z = 132$, $p < 10^{-300}$). The depth-2 tree again gains little (AUC 0.689 for E, 0.693 for E+S). In the ablation, raw size or virtual size alone recovers most of the gain of E+S and the size ratio helps least, as on the first dataset (Table 5).

On EMBER the random forest is clearly ahead of XGBoost, which it is not on the first dataset, and this advantage comes from near-duplicates that exact deduplication leaves in place. When sections with the same raw and virtual size and entropy equal to two decimals are kept in the same fold, the AUC of RF with E+S falls from 0.900 to 0.863 and its TPR at 1% FPR from 52.4% to 34.8%; when all sections of the same raw size are grouped, to 0.765 and 13.0%. XGBoost is less affected (AUC 0.834, 0.822 and 0.784; TPR 24.4%, 16.0% and 12.0%), and with entropy alone both models fall to an AUC of 0.63–0.72. A flexible model can memorise exact section sizes, so on a large dataset even deduplicated results need near-duplicate grouping before they are read as generalisation.

The virtual address of the first section, available only in the first dataset, is a further structural trace: it exceeds 4096 for 2,772 malicious and 36 benign sections. Adding this binary feature to E+S raises the AUC of XGBoost from 0.865 to 0.885 in the balanced protocol and from 0.800 to 0.827 in the size-matched design of the next section. Like the size fields, it is trivial for an attacker to control.

6.3 Size confounding does not replicate

Since malicious sections are larger than benign ones, a classifier can partly separate the classes by size, and entropy may itself correlate with size. To separate these effects we match the classes on raw size: $\log_2$(raw size) is divided into bins of width 0.05 (about 3.5% in size) and, in each bin, the same number of benign and malicious sections is drawn at random. Empty raw sections

Table 2 Effect of duplicated first sections: XGBoost, stratified cross-validation with random folds or with groups of related records kept in the same fold. PE Section Headers: all 43,139 files, 10 folds, 95% bootstrap intervals in brackets. EMBER: training split (599,944 files), 5 folds

Data	Feat.	Folds	ROC AUC	TPR@1%FPR	TPR@5%FPR
PE Section Headers	E	random	0.872 [0.864, 0.879]	0.368 [0.346, 0.388]	0.525
	E	identical sections grouped	0.761 [0.749, 0.773]	0.073 [0.044, 0.104]	0.210
	E	near-duplicates grouped	0.736	0.077	0.237
	E	same raw size grouped	0.745	0.076	0.155
	E+S	random	0.935 [0.929, 0.940]	0.479 [0.443, 0.547]	0.703
	E+S	identical sections grouped	0.830 [0.820, 0.840]	0.168 [0.154, 0.212]	0.492
	E+S	near-duplicates grouped	0.803	0.193	0.394
	E+S	same raw size grouped	0.786	0.170	0.373
EMBER 2018 (training split)	E	random	0.775	0.211	0.349
	E	identical sections grouped	0.696	0.052	0.220
	E	near-duplicates grouped	0.700	0.048	0.274
	E	families + identical grouped	0.686	0.059	0.260
	E+S	random	0.863	0.329	0.511
	E+S	identical sections grouped	0.769	0.213	0.342
	E+S	near-duplicates grouped	0.772	0.178	0.331
	E+S	families + identical grouped	0.731	0.124	0.270

Table 3 Balanced protocol on distinct sections of the PE Section Headers data: 1,331 benign vs. 1,331 random malicious sections, ten draws, stratified 10-fold cross-validation per draw. Mean ± standard deviation over draws.

Model	Features	Accuracy	F1	ROC AUC	TPR@1%FPR	TPR@5%FPR
DT	E	0.671 ± 0.014	0.566 ± 0.048	0.684 ± 0.022	0.166 ± 0.032	0.349 ± 0.017
DT	S	0.635 ± 0.008	0.707 ± 0.006	0.629 ± 0.006	0.003 ± 0.003	0.003 ± 0.004
DT	E+S	0.671 ± 0.014	0.566 ± 0.048	0.683 ± 0.022	0.154 ± 0.041	0.350 ± 0.014
RF	E	0.692 ± 0.010	0.689 ± 0.010	0.751 ± 0.010	0.000 ± 0.000[a]	0.312 ± 0.015
RF	S	0.666 ± 0.010	0.670 ± 0.009	0.744 ± 0.011	0.162 ± 0.014	0.305 ± 0.018
RF	E+S	0.791 ± 0.010	0.786 ± 0.011	0.876 ± 0.008	0.450 ± 0.024	0.600 ± 0.011
XGBoost	E	0.710 ± 0.008	0.699 ± 0.010	0.778 ± 0.009	0.222 ± 0.011	0.377 ± 0.018
XGBoost	S	0.664 ± 0.006	0.678 ± 0.007	0.736 ± 0.008	0.089 ± 0.014	0.257 ± 0.016
XGBoost	E+S	0.780 ± 0.008	0.773 ± 0.009	0.865 ± 0.009	0.362 ± 0.042	0.552 ± 0.014

[a]With a single feature, the random forest gives the same top score to many benign and malicious sections, so the operating point at 1% FPR is not reached.

are excluded. On the first dataset the matched sets contain 1,287 pairs and the matching is tight (median raw size 90 vs. 92 kB, Kolmogorov–Smirnov statistic 0.009, and raw size alone has an AUC of 0.500); on the distinct sections of the EMBER training split they contain 106,847 pairs. As a reference, we evaluate the same benign sections against an equally sized random malicious sample.

Table 6 shows that the two datasets disagree. On the first dataset, once size is controlled, the structural features alone fall close to chance (AUC 0.544 for XGBoost, from 0.731), while entropy is less affected (from 0.772 to 0.738), and E+S remains better than E (0.800). On EMBER, matching barely changes any result: the AUC of S falls only from 0.701 to 0.688 and that of E+S from 0.819 to 0.808. The strong size confound of the first dataset is therefore a property of that dataset, consistent with its much larger size gap between classes, and not a general property of malware. What does replicate is that E+S outperforms E after matching, i.e. that entropy relative to size carries more information than entropy alone.

6.4 Temporal drift: detections are memorised sections

Table 7 trains XGBoost on the EMBER training split and tests it on the files of the following two months. On the whole test set, entropy alone reaches an AUC of 0.704 and a TPR of 19.7% at 1% FPR. However, 92,557 of the 199,976 test files have a first section identical to one in the training split. On the remaining 107,419 files, whose first section was never seen in training, the AUC falls to 0.638 and the TPR at 1% FPR to 0.9%; with E+S, from 0.824 to 0.750 and from 30.4% to 7.9%.

The detections of the entropy-only model are almost entirely memorised sections. At 1% FPR

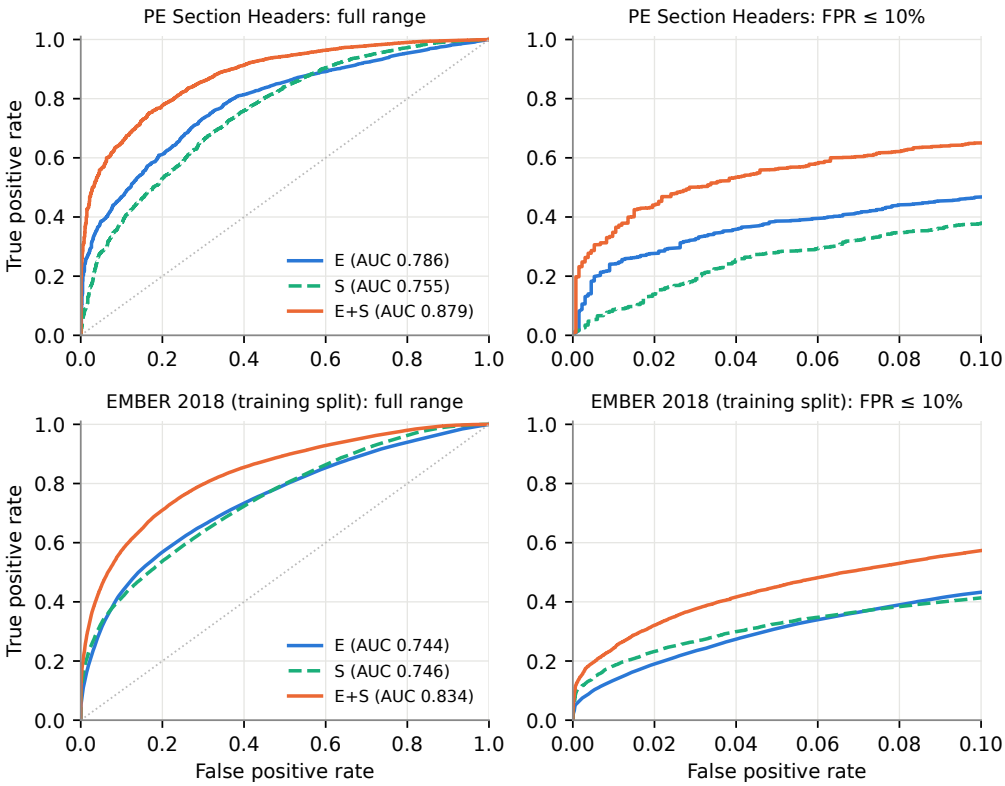


Fig. 3 ROC curves of XGBoost on distinct first sections (out-of-fold scores) for entropy alone (E), structural features alone (S) and both (E+S). Top: PE Section Headers (10-fold); bottom: EMBER 2018 training split (5-fold). Right: zoom on FPR $\leq 10\%$

it flags 19,657 malicious test files; 19,168 of them (97.5%) have a first section seen in training, 18,829 share the same entropy value (7.9597 bits/byte), and 18,830 belong to a single AVClass family. Very high entropy is not what is being detected: one recurrent packed section of one family is. The TPR at 0.1% FPR on the whole test set (18.9%) is almost equal to that at 1% FPR for the same reason.

The name of the first section gives a complementary view (Table 8). For the 80.2% of EMBER files whose first section is named `.text`, `CODE` or `.code`, i.e. looks like ordinary code, the entropy alone separates the classes with a rank AUC of only 0.600. The signal is concentrated in first sections with unusual names, where malicious sections have a median entropy of 7.96 against 4.02 for benign ones (AUC 0.942, or 0.922 excluding the largest family). In first sections with known packer names, which are 96.8% malicious, entropy is *inversely* related to the label (AUC 0.115): the benign files in this group are packed too, with entropy close to 8.

6.5 Robustness to padding

We now ask how easily the signal can be removed. Following the threat model of Section 3.3, the attacker appends k bytes of a single value to the first section, with $k/n \in \{0.05, 0.1, 0.25, 0.5, 1, 2, 4, 8\}$. By Proposition 1 the new entropy lies in $[\lambda H, \lambda H + h(\lambda)]$ for every possible section content. On the first dataset we train XGBoost with 10-fold cross-validation on all distinct sections, set the threshold at the given FPR on the out-of-fold benign scores, take the malicious sections it detects, and score their padded versions with the model of their own fold. On EMBER we train on the training split and attack the detected malicious files of the test split. A detected section *evades with certainty* if, for at least one padding fraction, its score falls below the threshold for every entropy value in the interval, i.e. whatever

Table 4 Full-data protocol on distinct first sections. PE Section Headers: 17,605 sections, stratified 10-fold cross-validation, 95% bootstrap intervals for AUC. EMBER: 329,855 sections of the training split, stratified 5-fold cross-validation

Data	Model	Feat.	ROC AUC	TPR@1%
PE S.H.	RF	E	0.708 [0.693, 0.722]	0.000[a]
	RF	S	0.767 [0.755, 0.779]	0.000[a]
	RF	E+S	0.896 [0.888, 0.903]	0.500[b]
	XGBoost	E	0.786 [0.775, 0.798]	0.241
	XGBoost	S	0.755 [0.741, 0.768]	0.087
	XGBoost	E+S	0.879 [0.870, 0.888]	0.348
EMBER	RF	E	0.702	0.150
	RF	S	0.812	0.395
	RF	E+S	0.900	0.524
	XGBoost	E	0.744	0.133
	XGBoost	S	0.746	0.183
	XGBoost	E+S	0.834	0.244

[a]The random forest gives the same top score to more than 1% of the benign sections, so the operating point is not reached.
[b]For the same reason this value is unstable under resampling (95% bootstrap interval 0.000–0.524).

Table 5 Ablation: entropy plus one structural feature at a time (ROC AUC). PE Section Headers: balanced protocol, mean over ten draws (standard deviations at most 0.011). EMBER: full-data protocol on the 329,855 distinct sections of the training split

	PE Section Headers		EMBER	
Features	RF	XGBoost	RF	XGBoost
E	0.751	0.778	0.702	0.744
E + raw size	0.864	0.856	0.869	0.816
E + virtual size	0.863	0.855	0.879	0.817
E + size ratio	0.829	0.823	0.850	0.806
E+S (all three)	0.876	0.865	0.900	0.834

its content. Tree ensembles are piecewise constant in each feature, so this can be checked exactly: once the padding fraction fixes the sizes, the split values of the model on entropy, and on information content, which is then monotone in entropy, cut the interval into finitely many pieces of constant score, and we evaluate one point in each piece. We also report the case in which the padding byte does not occur in the section, where the upper bound is exact.

Figure 4 shows, for the first dataset and the variant in which the virtual size grows, i.e. the one less favourable to the attacker, that small amounts of padding suffice: with padding equal to 25% of the section size, 60.8% of the sections detected by the entropy-only model at 1% FPR evade it with certainty, and 43.5% of those detected by E+S.

Table 6 Size-matched evaluation with XGBoost: benign and malicious sections matched on raw size, against an unmatched reference with the same benign sections. Mean over ten draws (PE Section Headers, 1,287 pairs) or three draws (EMBER training split, 106,847 pairs); standard deviations of the AUC are at most 0.015

		ROC AUC		TPR@1%FPR	
Data	Feat.	ref.	matched	ref.	matched
PE S.H.	E	0.772	0.738	0.202	0.170
	S	0.731	0.544	0.076	0.038
	E+S	0.863	0.800	0.361	0.258
EMBER	E	0.735	0.719	0.132	0.089
	S	0.701	0.688	0.181	0.123
	E+S	0.819	0.808	0.240	0.187

Table 7 EMBER temporal split: XGBoost trained on files that appeared up to October 2018 and tested on November–December 2018, on all test files and on those whose first section never occurs in the training split

Test files	Feat.	AUC	TPR@1%	TPR@5%
all (n=199,976)	E	0.704	0.197	0.284
	S	0.769	0.275	0.391
	E+S	0.824	0.304	0.453
unseen first section (n=107,419)	E	0.638	0.009	0.117
	S	0.636	0.024	0.120
	E+S	0.750	0.079	0.171

Table 8 EMBER: rank AUC of first-section entropy (no model) within groups defined by the name and size of the first section. Packer names: `UPX0`, `UPX1`, `.MPRESS1`, `.MPRESS2`, `packerBY`, `.aspack`, `.adata`, `.petite`, `nsp0`, `.nsp0`

First section	Files	Malicious	AUC of H
code names[a]	641,559	46.9%	0.600
other names, non-empty	102,695	58.7%	0.942
packer names, non-empty	10,120	96.8%	0.115
empty on disk	45,546	64.2%	0.500
all	799,920	50.0%	0.641

[a] `.text`, `CODE` or `.code`.

Large amounts of padding are counter-productive, because they push the entropy into the low range that the models also flag as malicious.

Table 10 summarises all attacks. Choosing the best padding fraction, guaranteed evasion of E reaches 63.7% [62.2, 65.2] in both variants, since

E ignores the sizes. In the variant with the virtual size unchanged, the size features do not protect the model either: guaranteed evasion of E+S reaches 91.5% [90.7, 92.2], and the model detects only 3.0% of the malicious sections after the attack. Only if the virtual size grows with the padding do the size fields partly absorb the attack, and guaranteed evasion of E+S falls to 51.6% [50.3, 52.9].

On the EMBER test set, guaranteed evasion is 100% for E and 97.4% for E+S in both variants. This result says more about memorisation than about entropy. As shown in Section 6.4, almost all detections of these models on the test set are copies of one first section seen in training, with entropy 7.9597 bits/byte, and any change to its feature vector, padding or not, moves it away from the region the model has learned. The EMBER padding results therefore confirm that memorised detections are fragile, rather than measure the robustness of an entropy signal that generalises; the first dataset, where duplicates are removed, is the more informative test of the attack.

Benign-looking and constrained padding.

Table 9 extends the attack in the two directions of Section 3.3, keeping for each detected section the best padding content among a single repeated byte and $H(r) \in \{6.0, 6.25, 6.5\}$ bits/byte, with the exact interval of Proposition 1 for each $H(r)$. On the first dataset, choosing the content per section helps the attacker: guaranteed evasion of E+S rises to 96.1% with the virtual size fixed and to 97.7% with the virtual size growing, and that of E to 96.7%. Benign-looking padding alone is more effective than a single byte against E (96.7% against 63.7%) and against E+S when the virtual size grows (up to 97.5% against 51.6%), but less effective against E+S when the virtual size is fixed (65.4–73.7% against 91.5%). Discarding runs of identical bytes before computing the entropy would therefore leave most of the attack in place. The picture changes when the attacker must stay inconspicuous. Within the alignment slack, the growing variant can append less than 4 KiB (a median of about 2 KiB, 0.3–0.4% of the raw size) and guarantees evasion for only 5.8% of the sections detected by E+S (3.1% for E); keeping the excess of raw over virtual size below 4 KiB, the fixed variant guarantees evasion for

Table 9 Guaranteed evasion at 1% FPR (share of detected malicious samples) under further padding scenarios. Content: best of a single repeated byte and padding of entropy $H(r) \in \{6.0, 6.25, 6.5\}$ bits/byte. Slack: $k \leq \lceil v/4096 \rceil \cdot 4096 - v$; excess: raw size at most 4 KiB above virtual size; in both cases k is 25, 50 or 100% of the allowed amount. PE Section Headers: 10-fold cross-validation on distinct sections; EMBER: temporal split

	PE S.H.		EMBER	
Scenario	E	E+S	E	E+S
Single byte, vsize fixed	0.637	0.915	1.000	0.974
Single byte, vsize grows	0.637	0.516	1.000	0.974
Best content, vsize fixed	0.967	0.961	1.000	0.974
Best content, vsize grows	0.967	0.977	1.000	0.974
Best, grows within slack	0.031	0.058	0.005	0.165
Best, fixed, excess $\leq$ 4 KiB	0.068	0.235	0.969	0.855

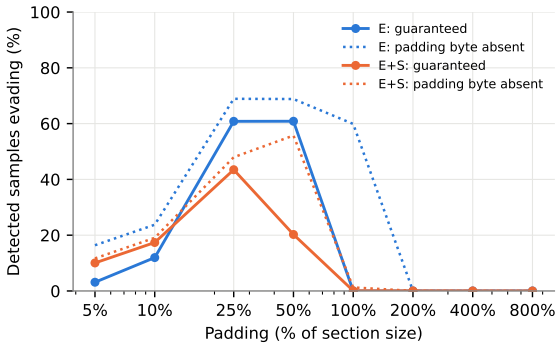


Fig. 4 PE Section Headers data, virtual size growing with the padding: share of malicious sections detected by XGBoost at 1% FPR that evade detection after low-entropy padding, as a function of the padding size. Solid: evasion guaranteed for any section content; dotted: padding byte absent from the section (exact upper bound of Proposition 1)

23.5% (6.8% for E). With these header features, detections can be removed with certainty only by appends large enough to make the section header unusual, which a consistency check on raw and virtual size would notice. On EMBER the constrained fixed attack still evades 85.5–96.9% of the detections, consistently with the memorisation discussed above, while within the slack, which is zero for most detected sections there, it evades at most 16.5%.

Adversarial training.

We retrain E+S with one padded copy of each malicious training section, with a random padding fraction and a random entropy within the bounds.

Table 10 Padding attacks at 1% FPR (threshold set on benign scores). TPR clean: share of malicious samples detected before the attack. Evasion: share of detected samples that evade with certainty for at least one padding fraction, with 95% Wilson intervals where computed. TPR after: share of all malicious samples still detected. Here the threshold is the 99th percentile of the benign scores and a sample is detected if its score exceeds it, so TPR clean can differ slightly from the ROC-based values of Tables 4 and 11. PE Section Headers: 10-fold cross-validation on distinct sections; EMBER: temporal split. Adv.: adversarial training on the growing variant only, or on both variants

			Virtual size grows		Virtual size fixed	
Data	Model (XGBoost)	TPR clean	Evasion	TPR after	Evasion	TPR after
PE Section Headers	E	0.241	0.637 [0.622, 0.652]	0.088	0.637 [0.622, 0.652]	0.088
	E+S	0.354	0.516 [0.503, 0.529]	0.171	0.915 [0.907, 0.922]	0.030
	B+raw+vsize	0.243	0.624	0.091	0.668	0.080
	E+S, adv. (grow)	0.323	0.379	0.201	0.364	0.206
	E+S, adv. (both)	0.330	0.396	0.200	0.021	0.323
	monotone B+raw+vsize	0.000	–	0.000	–	0.000
	idem, at 5% FPR	0.046	0.000	0.046	0.000	0.046
EMBER 2018	E	0.197	1.000	0.000	1.000	0.000
	E+S	0.304	0.974	0.008	0.974	0.008
	monotone B+raw+vsize	0.000	–	0.000	–	0.000

Trained only on the growing variant, the model reduces guaranteed evasion to 36–38% in both variants, at a small cost in clean performance (AUC 0.865 instead of 0.879). Trained on both variants, it almost neutralises the fixed-size attack (2.1% evasion) but not the growing one (39.6%). An attacker who tries both variants and keeps the better one still leaves only 19.3–20.0% of the malicious sections detected (bounds from the two marginal results), against at most 3.0% without adversarial training. Adversarial training helps, but only for the perturbations it has seen, and it does not restore the clean detection rate.

The price of provable robustness.

By Corollary 1, an XGBoost model constrained to be non-decreasing in $\log(1 + B)$, raw size and virtual size cannot be evaded by appending bytes, and Table 10 confirms that no detected sample evades it. Table 11 shows what this guarantee costs. On the first dataset the monotone model reaches an AUC of 0.621 and detects no malicious section at 1% FPR; in the size-matched design its AUC is 0.487, i.e. chance. On the EMBER temporal split its AUC is 0.520. The same three features without the constraint reach an AUC of 0.820, but they are evaded as easily as E (Table 10). With these header features, provable robustness to padding and useful detection are incompatible: the information that separates the classes is exactly the information that padding can remove. The monotone model is a way to measure the cost of a guarantee, not a proposed defence; simpler checks, such as flagging a raw size much larger than the virtual size, address specific variants of the attack (Table 9).

Table 11 Clean performance (XGBoost, AUC; TPR at 1% FPR in parentheses) of models on information content B, raw size and virtual size, unconstrained or monotone (Corollary 1), compared with E and E+S. PE Section Headers data unless stated

Features	Full data	Balanced	Matched
E	0.786 (0.241)	0.778	0.738
E+S	0.879 (0.348)	0.865	0.800
B+raw+vsize	0.820 (0.242)	0.801	0.651
monotone	0.621 (0.000)	0.619	0.487
EMBER, monotone[a]	0.520 (0.000)	–	–

[a]Temporal split.

6.6 Sensitivity to empty raw sections

Repeating the balanced protocol on the first dataset without the 129 empty raw sections changes accuracy by at most 0.85 percentage points and AUC by at most 0.66 points, for both ensembles and all feature sets. These records are not what drives the results.

7 Discussion and limitations

How far does entropy go?

Without duplicates, entropy alone gives an AUC between about 0.63 (EMBER, sections of the same raw size grouped) and 0.79 depending on the dataset and protocol, and detects at best about one malicious section in four at 1% false positives.

On EMBER, almost all of these detections are recurrent sections already seen in training, and on new sections the entropy-only model is close to useless at low false positive rates. Header size fields add a significant improvement, partly through a size confound that is dataset-specific. Entropy is useful to prioritise samples for deeper analysis or as one feature among many [4], not as a detector.

What generalises and what does not.

The two datasets agree on the duplication rate, on the leakage it causes under random splits, on the benefit of combining entropy with size fields, and on the fragility to padding, although on EMBER this fragility largely reflects memorisation. They disagree on the size confound and on the meaning of empty first sections. Conclusions drawn from a single dataset of this kind should therefore be read as properties of that dataset until replicated, as Arp et al. [29] and Pendlebury et al. [30] recommend for ML security evaluations in general.

Labels and data quality.

Labels come from the original datasets. In the first dataset, benign files with high entropy are those whose label is most likely to be wrong, so we re-checked all 35 benign files with first-section entropy above 6.75 bits/byte on VirusTotal (October 2026). All 35 were found; each had been scanned by 48 to 75 engines. Twelve were flagged by at least one engine, but none by more than two: eight by one engine and four by two. With a threshold of three or more engines, a common choice to filter out isolated false positives of individual engines, no label changes, and we keep the original labels. EMBER labels and families are those distributed with the dataset.

Scope of the feature and of the data.

We measure the entropy of one section only, the first of the section table, and we isolate it on purpose, so that its limits can be stated exactly. Packers that keep a normal-looking first section and store the payload elsewhere are not captured; per-section or windowed entropy measures [11] would be more informative, although the padding bounds of Proposition 1 apply to each section separately. In a full static detector such as the EMBER model [4], entropy enters through several features at once, including the byte-entropy histogram of the whole file and the entropy of every section, and padding moves all of them. Whether the duplication and padding effects we measure persist in a model on the full feature vector, and whether padding then becomes detectable rather than merely ineffective, is not answered here.

The first dataset is from 2019 and its benign class is small (1,695 files); EMBER 2018 is larger but now eight years old, current packers may differ, and its temporal split covers only two months. More recent benchmarks with longer time spans, such as BODMAS [38] and EMBER2024 [39], would allow the duplicate audit and the temporal evaluation to be repeated on more recent malware.

Realism of the padding analysis.

Our padding analysis works in feature space: we do not modify binaries, and we assume that the two modifications of Section 3.3 preserve functionality within the limits discussed there, as previous work on section padding does [14]. Empirical work on adversarial PE malware applies functionality-preserving modifications to real binaries [15, 26]; validating our feature-space results on binaries is left to future work. The bounds of Proposition 1 make the guaranteed-evasion results independent of the unknown section content, but other features of a real detector could reveal the modification. The two variants have different limits, which Table 9 quantifies: within the alignment slack, or keeping the excess of raw over virtual size below 4 KiB, guaranteed evasion on the deduplicated data falls to 3–24%. The slack is estimated assuming a 4 KiB section alignment and contiguous sections, since our data do not record the address of the next section. Low-entropy packing is not hypothetical: Mantovani et al. [8] found it in a large share of real-world malware, and such samples are invisible to entropy-based checks by construction.

8 Conclusion

We measured how much the entropy of the first PE section tells about maliciousness on two datasets of different origin and size. In both, almost two thirds of malicious files share their first section with other files, and random cross-validation largely measures memorisation; on a

temporal split, almost all detections of an entropy-only model are sections seen in training, and on new sections it detects less than 1% of malware at 1% false positives. Size fields add useful information, but the strong size confound of one dataset does not replicate in the other, and empty first sections are a trace of packing, not of malice. Finally, standard bounds on the entropy of padded sections, computable from header data alone, show that unconstrained appends guarantee evasion for most detected samples, whatever their content, whereas appends small enough to keep the section header unremarkable rarely do; provable robustness is possible, through the monotonicity of information content, but only at the cost of almost all detection ability.

Future work includes repeating the analysis on more recent benchmarks such as BODMAS and EMBER2024, on per-section entropy profiles and on detectors that use the full static feature vector, validating the padding attacks on real binaries, extending the bounds to other modifications such as section insertion, and studying which combinations of static features make padding detectable rather than merely ineffective.

Data and code availability

The PE Section Headers dataset is publicly available on IEEE Dataport [16]. EMBER 2018 is publicly available from its authors [4]; the first-section features used here can be regenerated from it with the script in the code repository.

The code that reproduces all cleaning steps, tables and figures, the EMBER extraction script and the scripts used for the VirusTotal label check are available in an anonymised repository: https://github.com/paper9-design/paper. All experiments use fixed random seeds.

References

[1] Schultz, M.G., Eskin, E., Zadok, E., Stolfo, S.J.: Data mining methods for detection of new malicious executables. In: Proceedings of the 2001 IEEE Symposium on Security and Privacy, pp. 38–49 (2001)

[2] Kolter, J.Z., Maloof, M.A.: Learning to detect malicious executables in the wild. In: Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 470–478 (2004). https://doi.org/10.1145/1014052.1014105

[3] Saxe, J., Berlin, K.: Deep neural network based malware detection using two dimensional binary program features. In: 10th International Conference on Malicious and Unwanted Software (MALWARE) (2015). https://doi.org/10.1109/MALWARE.2015.7413680

[4] Anderson, H.S., Roth, P.: EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. arXiv:1804.04637 (2018). https://arxiv.org/abs/1804.04637

[5] Collberg, C., Thomborson, C., Low, D.: A taxonomy of obfuscating transformations. Technical Report 148, Department of Computer Science, University of Auckland (July 1997)

[6] Moser, A., Kruegel, C., Kirda, E.: Limits of static analysis for malware detection. In: Proceedings of the 23rd Annual Computer Security Applications Conference (ACSAC 2007), pp. 421–430 (2007)

[7] Ugarte-Pedrero, X., Balzarotti, D., Santos, I., Bringas, P.G.: SoK: Deep packer inspection: A longitudinal study of the complexity of run-time packers. In: 2015 IEEE Symposium on Security and Privacy, pp. 659–673 (2015). https://doi.org/10.1109/SP.2015.46

[8] Mantovani, A., Aonzo, S., Ugarte-Pedrero, X., Merlo, A., Balzarotti, D.: Prevalence and impact of low-entropy packing schemes in the malware ecosystem. In: Network and Distributed System Security Symposium (NDSS) (2020). https://doi.org/10.14722/ndss.2020.24297

[9] Samociuk, D.: Antivirus evasion methods in modern operating systems. Applied Sciences **13**(8), 5083 (2023) https://doi.org/10.3390/app13085083

[10] Asghar, H.J., Zhao, B.Z.H., Ikram,

M., Nguyen, G., Kaafar, D., Lamont, S., Coscia, D.: Use of cryptography in malware obfuscation. Journal of Computer Virology and Hacking Techniques **20**(1), 135–152 (2024) https://doi.org/10.1007/s11416-023-00504-y

[11] Lyda, R., Hamrock, J.: Using entropy analysis to find encrypted and packed malware. IEEE Security & Privacy **5**(2), 40–45 (2007) https://doi.org/10.1109/MSP.2007.48

[12] Baysa, D., Low, R.M., Stamp, M.: Structural entropy and metamorphic malware. Journal of Computer Virology and Hacking Techniques **9**(4), 179–192 (2013) https://doi.org/10.1007/s11416-013-0185-4

[13] Abazar, M., Manavi, F., Altulyan, M., Ayari, M., Boostani, R.: Multiscale and Kolmogorov entropies as effective features for malware detection. Journal of Computer Virology and Hacking Techniques **21**(1), 20 (2025) https://doi.org/10.1007/s11416-025-00567-z

[14] Anderson, H.S., Kharkar, A., Filar, B., Evans, D., Roth, P.: Learning to Evade Static PE Machine Learning Malware Models via Reinforcement Learning. arXiv:1801.08917 (2018). https://arxiv.org/abs/1801.08917

[15] Kozák, M., Jureček, M., Stamp, M., Di Troia, F.: Creating valid adversarial examples of malware. Journal of Computer Virology and Hacking Techniques **20**(4), 607–621 (2024) https://doi.org/10.1007/s11416-024-00516-2

[16] Oliveira, A.: Malware Analysis Datasets: PE Section Headers. IEEE Dataport (2019). https://doi.org/10.21227/2czh-es14

[17] Cover, T.M., Thomas, J.A.: Elements of Information Theory, 2nd edn. Wiley-Interscience, Hoboken, NJ (2006). https://doi.org/10.1002/047174882X

[18] Belaoued, M., Mazouzi, S.: A real-time PE-malware detection system based on CHI-square test and PE-file features. In: Computer Science and Its Applications (CIIA 2015). IFIP Advances in Information and Communication Technology, vol. 456, pp. 416–425. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19578-0_34

[19] Singh, P., Borgohain, S.K., Kumar, J.: Optimized feature representation and selection for malware detection using portable executable headers and machine learning. Journal of Computer Virology and Hacking Techniques **21**(1), 10 (2025) https://doi.org/10.1007/s11416-025-00555-3

[20] Liţă, C.V., Cosovan, D., Gavriluţ, D.: Anti-emulation trends in modern packers: A survey on the evolution of anti-emulation techniques in UPA packers. Journal of Computer Virology and Hacking Techniques **14**(2), 107–126 (2018) https://doi.org/10.1007/s11416-017-0291-9

[21] Kancherla, K., Donahue, J., Mukkamala, S.: Packer identification using Byte plot and Markov plot. Journal of Computer Virology and Hacking Techniques **12**(2), 101–111 (2016) https://doi.org/10.1007/s11416-015-0249-8

[22] Bernardinetti, G., Di Cristofaro, D., Bianchi, G.: PEzoNG: Advanced packer for automated evasion on Windows. Journal of Computer Virology and Hacking Techniques **18**(4), 315–331 (2022) https://doi.org/10.1007/s11416-022-00417-2

[23] Grosse, K., Papernot, N., Manoharan, P., Backes, M., McDaniel, P.: Adversarial Perturbations Against Deep Neural Networks for Malware Classification. arXiv:1606.04435 (2016). https://arxiv.org/abs/1606.04435

[24] Xu, W., Qi, Y., Evans, D.: Automatically evading classifiers: A case study on PDF malware classifiers. In: Network and Distributed System Security Symposium (NDSS) (2016). https://doi.org/10.14722/ndss.2016.23115

[25] Hu, W., Tan, Y.: Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN. arXiv:1702.05983 (2017). https://arxiv.org/abs/1702.05983

[26] Louthanová, P., Kozák, M., Jureček, M.,

Stamp, M., Di Troia, F.: A comparison of adversarial malware generators. Journal of Computer Virology and Hacking Techniques **20**(4), 623–639 (2024) https://doi.org/10.1007/s11416-024-00519-z

[27] Crasta, D.B., Kumar, V.: Enhancing robustness of deep learning-based malware detection against adversarial attacks. Journal of Computer Virology and Hacking Techniques **22**(1), 72 (2026) https://doi.org/10.1007/s11416-026-00655-8

[28] Aghakhani, H., Gritti, F., Mecca, F., Lindorfer, M., Ortolani, S., Balzarotti, D., Vigna, G., Kruegel, C.: When malware is packin' heat: Limits of machine learning classifiers based on static analysis features. In: Network and Distributed System Security Symposium (NDSS) (2020). https://doi.org/10.14722/ndss.2020.24310

[29] Arp, D., Quiring, E., Pendlebury, F., Warnecke, A., Pierazzi, F., Wressnegger, C., Cavallaro, L., Rieck, K.: Dos and don'ts of machine learning in computer security. In: 31st USENIX Security Symposium (USENIX Security 22), pp. 3971–3988 (2022)

[30] Pendlebury, F., Pierazzi, F., Jordaney, R., Kinder, J., Cavallaro, L.: TESSERACT: Eliminating experimental bias in malware classification across space and time. In: 28th USENIX Security Symposium (USENIX Security 19), pp. 729–746 (2019)

[31] Rossow, C., Dietrich, C.J., Grier, C., Kreibich, C., Paxson, V., Pohlmann, N., Bos, H., Steen, M.: Prudent practices for designing malware experiments: Status quo and outlook. In: 2012 IEEE Symposium on Security and Privacy, pp. 65–79 (2012). https://doi.org/10.1109/SP.2012.14

[32] Shannon, C.E.: A mathematical theory of communication. Bell System Technical Journal **27**, 379–423 (1948) https://doi.org/10.1002/j.1538-7305.1948.tb01338.x . Continued in vol. 27, pp. 623–656

[33] Microsoft: PE Format. Microsoft Learn, Win32 apps documentation. Accessed 8 October 2026. https://learn.microsoft.com/en-us/windows/win32/debug/pe-format

[34] Sebastián, M., Rivera, R., Kotzias, P., Caballero, J.: AVclass: A tool for massive malware labeling. In: Research in Attacks, Intrusions, and Defenses (RAID 2016). Lecture Notes in Computer Science, pp. 230–253. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45719-2_11

[35] Wilcoxon, F.: Individual comparisons by ranking methods. Biometrics Bulletin **1**(6), 80–83 (1945) https://doi.org/10.2307/3001968

[36] DeLong, E.R., DeLong, D.M., Clarke-Pearson, D.L.: Comparing the areas under two or more correlated receiver operating characteristic curves: A nonparametric approach. Biometrics **44**(3), 837–845 (1988) https://doi.org/10.2307/2531595

[37] Sun, X., Xu, W.: Fast implementation of DeLong's algorithm for comparing the areas under correlated receiver operating characteristic curves. IEEE Signal Processing Letters **21**(11), 1389–1393 (2014) https://doi.org/10.1109/LSP.2014.2337313

[38] Yang, L., Ciptadi, A., Laziuk, I., Ahmadzadeh, A., Wang, G.: BODMAS: An open dataset for learning based temporal analysis of PE malware. In: 2021 IEEE Security and Privacy Workshops (SPW), pp. 78–84 (2021). https://doi.org/10.1109/SPW53761.2021.00020

[39] Joyce, R.J., Miller, G., Roth, P., Zak, R., Zaresky-Williams, E., Anderson, H., Raff, E., Holt, J.: EMBER2024 – a benchmark dataset for holistic evaluation of malware classifiers. In: Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2, pp. 5516–5526 (2025). https://doi.org/10.1145/3711896.3737431