RAVEN: Measuring Detection Instability in Anti-virus Engines under Semantics-Preserving Perturbations

Younghoon Ban*
School of Software
Soongsil University
Seoul, Republic of Korea
byhoon6279@soongsil.ac.kr

Beomjin Jin*
Department of Cyber Security
Ajou University
Suwon, Republic of Korea
jinbumjin@ajou.ac.kr

Doowon Kim
Department of EECS
University of Tennessee, Knoxville
Knoxville, Tennessee, USA
doowon@utk.edu

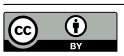
Hyoungshick Kim†✉
Department of Computer Science and Engineering
Sungkyunkwan University
Suwon, Republic of Korea
hyoung@skku.edu

Haehyun Cho†✉
School of Software
Soongsil University
Seoul, Republic of Korea
haehyun@ssu.ac.kr

Abstract

Real-world anti-virus (AV) engines continue to rely heavily on signature-based detection, whereas academic adversarial research has largely focused on ML-based models, leaving an evaluation gap. To better understand signature-based detection behavior, we analyze ClamAV's open-source signature database as a white-box reference point. Our analysis shows that raw byte-level signatures account for 94.54% of ClamAV detections, with triggered signatures predominantly mapping to content-bearing regions such as `.text`, `.rsrc`, and overlay. Common header-level perturbations achieve less than 2% evasion success on ClamAV, suggesting limitations in prior evaluation settings. We present RAVEN, a controlled original–perturbed pair framework for evaluating AV detection stability under 14 semantics-preserving perturbations, including two new methods targeting underexplored PE content and layout surfaces motivated by detection-surface analysis. Using over 140K adversarial examples (AEs), we evaluate detection stability across more than 70 AV engines on VirusTotal (VT). We find that content-level perturbations achieve up to 20.17% average evasion success, substantially higher than header-level perturbations. We also observe trade-offs across engines: some engines with lower false positive rates exhibit higher evasion rates under structural perturbations, while others with lower evasion rates incur higher false positive rates. In a 15-day longitudinal analysis, some engines show delayed recovery, and 20 of 76 engines still maintain ASR above 50% on the final day. These results indicate that snapshot-based evasion metrics are insufficient for evaluating AV robustness, as they do not capture structural sensitivity, engine-specific trade-offs, and temporal variability in detection behavior.

*Younghoon Ban and Beomjin Jin contributed equally to this work.
†Hyoungshick Kim and Haehyun Cho are co-corresponding authors.



ASE '26, Munich, Germany

ACM ISBN 979-8-4007-2882-2/2026/10
https://doi.org/10.1145/3832783.3837492

CCS Concepts

• **Security and privacy** → *Software security engineering*.

Keywords

anti-virus engines, malware detection, detection stability, semantics-preserving perturbations, robustness evaluation

ACM Reference Format:
Younghoon Ban, Beomjin Jin, Doowon Kim, Hyoungshick Kim, and Haehyun Cho. 2026. RAVEN: Measuring Detection Instability in Anti-virus Engines under Semantics-Preserving Perturbations. In *Proceedings of the 41st IEEE/ACM International Conference on Automated Software Engineering (ASE '26), October 12–16, 2026, Munich, Germany*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3832783.3837492

1 Introduction

AV systems struggle to handle non-semantic variations in malware binaries. Static pattern-based detection is sensitive to minor structural differences, while behavior-oriented analysis often fails to generalize beyond its execution context [10, 17, 27, 29, 34, 37]. Although modern AV products incorporate ML components, they continue to rely on hybrid pipelines combining signatures, heuristics, and behavioral analysis, as ML models often exhibit limited generalization [2, 18, 22, 50]. Prior adversarial malware research, however, primarily targets ML-based detectors [9, 33, 43, 53], creating an evaluation gap with real-world AV systems.

In this paper, we operationalize AV robustness as detection stability under semantics-preserving perturbations. Following a metamorphic testing perspective [13], given a malware sample s and a semantics-preserving transformation T, a robust AV engine should produce consistent detection decisions for s and $T(s)$. Detection inconsistency under such behavior-preserving transformations is a decision-level robustness defect. This framing is important because a single evasion result or snapshot ASR does not reveal whether the failure is tied to a specific binary region, perturbation operator, engine operating point, or update behavior over time.

Existing adversarial malware evaluations cover this problem only partially. Many perturbation strategies focus on convenient

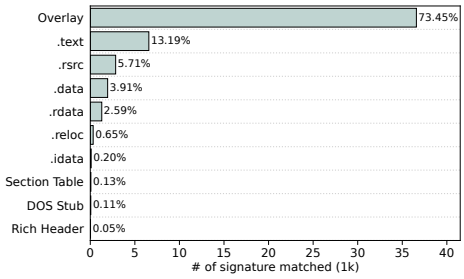
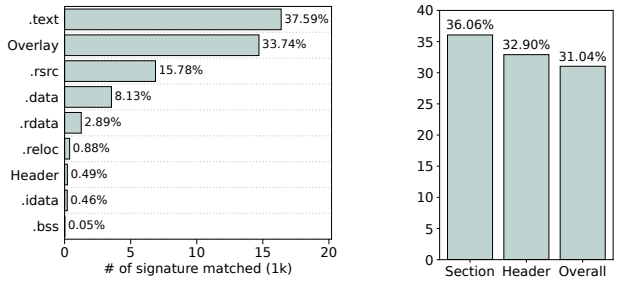


Figure 1: Top 10 regions of ClamAV signatures used to detect the malware samples we collected.



(a) MalConv **(b) EMBER**

Figure 2: Feature impact for MalConv and EMBER.

PE regions such as headers or metadata. As we show in § 2, header-oriented regions account for less than 1% of matched ClamAV signature locations, whereas content-bearing regions such as `.text`, `.rsrc`, and overlay dominate the matched signature locations. This pattern appears not only in ClamAV [14], but also in ML-based detectors such as EMBER [4] and MalConv [38]. Together, these observations reveal a mismatch between the perturbation surfaces that prior evaluations commonly target and the regions that materially affect AV detection.

Prior VirusTotal (VT) measurement studies have shown that AV labels and detection outcomes vary across engines and over time [26, 49, 56]. These studies primarily analyze natural label disagreement, label drift, and temporal inconsistency on identical samples [49, 56]. Such measurements show that detection outcomes change, but they cannot directly attribute instability to a controlled binary change. RAVEN takes an interventional approach. It holds the original sample fixed and systematically varies semantics-preserving perturbations, allowing us to attribute detection instability to specific perturbation operators, PE regions, and longitudinal re-detection behavior.

To address these gaps, we present RAVEN, a controlled original–perturbed pair measurement framework for evaluating AV detection stability under semantics-preserving perturbations. RAVEN applies 14 perturbation operators to Windows PE binaries and measures detection stability along four dimensions: perturbation impact, AV specificity, strategy efficiency, and resilience. Using RAVEN, we conduct white-box analysis of ClamAV and black-box evaluation across more than 70 VT engines, complemented by controlled endpoint validation. Our contributions are as follows:

- We show that raw byte-level signatures account for 94.54% of ClamAV detections, and that triggered signatures predominantly map to content-bearing regions such as `.text`, `.rsrc`, and overlay. Common header-level perturbations achieve less than 2% evasion success, revealing that header- and metadata-level perturbations capture only a small fraction of the practical detection surface.
- We introduce a controlled, interventional methodology for evaluating AV robustness under semantics-preserving perturbations. Unlike prior observational VT studies, our approach holds each malware sample fixed and varies only semantics-preserving perturbations, enabling attribution to specific operators, PE regions, and longitudinal re-detection behavior.
- We realize this methodology in RAVEN, an automated framework that structures 14 semantics-preserving operators as metamorphic relations over Windows PE binaries, including two novel operators targeting `.rsrc`, overlay, and section-layout metadata.
- We demonstrate substantial detection instability across over 140K AEs and more than 70 VT engines. Controlled endpoint validation supports that the main ASR–FPR trends are not solely VT-specific, and longitudinal analysis shows that instability varies across perturbation operators, engines, and time.

We release RAVEN as an open-source tool at https://github.com/byhoon6279/RAVEN.git

2 Detection Surface Analysis

This section analyzes the detection surfaces of malware detectors to identify which binary regions most strongly influence detection and to clarify why prior adversarial evaluations may fail to reflect real-world AV behavior. We ground this analysis in one real-world signature-based engine, ClamAV [14], and two widely used ML-based detectors, EMBER [4] and MalConv [38].

2.1 Critical Regions for Malware Detection

Designing effective perturbations requires identifying the binary regions that detectors actually rely on. We therefore analyze the detection surfaces of three representative systems: ClamAV as a signature-based engine, and EMBER and MalConv as ML-based detectors. We provide additional methodological details for each system in Supplementary Sections A and B. [1]

As shown in Figure 1, ClamAV signatures predominantly match content-bearing regions, particularly `.text`, `.rsrc`, and the overlay. The figure reports each PE region's share of matched ClamAV signatures, not a size-normalized per-byte detection density. The overlay alone accounts for approximately 35k matches, far exceeding those in structural headers such as the `DOS Stub` and `Rich Header`. This emphasis on content-bearing regions also appears in ML-based detectors. We estimate region importance using Grad-CAM [42] for MalConv and feature importance aggregation for EMBER. Figure 2 shows that MalConv's most influential windows concentrate in the `.text` (16k), overlay (14k), and `.rsrc` (6k) regions, consistent

[1] https://doi.org/10.5281/zenodo.19229602

with our ClamAV analysis. Similarly, EMBER assigns greater importance to section contents (36%) than to header-related attributes (31%). Although ClamAV, MalConv, and EMBER expose different forms of evidence, their region-level summaries show a consistent pattern in our analysis. Signature matches in ClamAV, post-hoc byte-window attribution in MalConv, and PE-feature importance in EMBER all assign greater weight to content-bearing regions than to header-oriented structural metadata. In this paper, we refer to DOS Stub, Rich Header, and Section Table regions as header-oriented structural metadata.

2.2 Gaps in Prior Adversarial Evaluations

Our analysis suggests that prior adversarial evaluations may not fully capture the detection behavior of real-world AV systems.

Perturbation Strategy. Existing methods often modify convenient regions such as PE headers or apply semantics-preserving code transformations such as `NOP` injection and instruction substitution [9, 10, 25, 33, 34, 39, 43, 53, 54]. However, many of these perturbations target header-oriented regions, which account for less than 1% of matched ClamAV signature locations in our analysis. This surface-coverage mismatch leaves detection-critical content-bearing regions, particularly `.rsrc` and overlay content, largely untouched by prior perturbation strategies.

Targeted AV Engines. Existing methods also primarily target ML-based detectors [4, 23, 32, 38, 51], which are typically evaluated in controlled laboratory settings. In contrast, production AV systems combine signature-based, heuristic, and behavioral analysis [14, 37, 40], resulting in more complex and diverse detection behavior. These observations motivate evaluations that target real-world AV engines and perturb detection-critical regions that have been underexplored in prior work.

3 Research Questions & Methodology

The main goal of this work is to evaluate the detection instability of real-world AV systems against semantics-preserving perturbations. This section presents our research questions and introduces the design of RAVEN, a framework for systematically assessing the robustness of real-world AV systems.

3.1 Research Questions

RQ1 (Perturbation impact): To what extent does each semantics-preserving perturbation method affect real-world AV detection performance? We apply 14 perturbations independently to malware samples and evaluate their impact on real-world AV systems and two ML-based detectors, EMBER [4] and MalConv [38]. We measure Attack Success Rate (ASR) and Label Change Rate (LCR) to identify which perturbations most strongly influence detection performance across different systems.

RQ2 (AV specificity): How does detection instability manifest across AV engines under AEs and benign samples? Evasion rates alone conflate robustness with over-sensitivity, making it difficult to distinguish resilient detectors from those that overreact to benign variation. To assess specificity, we include benign PE binaries and measure false negatives on perturbed malware and false positives on benign samples. Engines that consistently detect original and perturbed malware while ignoring benign samples indicate robust behavior, whereas engines that misclassify benign samples or miss AEs exhibit higher sensitivity. This enables characterization of robustness, perturbation sensitivity, and over-sensitivity.

RQ3 (Efficient strategy): What strategies can be employed to effectively evaluate the robustness of real-world AV systems? We investigate perturbation-selection strategies for robustness measurement. First, we cluster perturbations with similar effects across VT engines and select representative perturbations. Using these representatives, we construct multi-perturbation combinations to examine how perturbation coverage affects observed detection instability. Finally, we compare these combinations with AEs generated by four state-of-the-art (SOTA) frameworks: Gym-Malware (Gym) [3], MAB-Malware (MAB) [43], MalwareMakeover (MMO) [34], and Malguise [33]. These frameworks instantiate overlapping header-, layout-, and instruction-level perturbation strategies, allowing us to assess how different perturbation surfaces reveal detection instability across real-world AV systems. (See Section C in the supplementary material.)

RQ4 (Resilience): What is the response time of real-world AV systems to newly crafted AEs? To evaluate detection latency, we measure the time from each AE's upload to the first AV detection. Using the representative perturbation strategies identified in RQ3, we generate AEs and monitor detection results over a 15-day period by periodically resubmitting them to VT. This approach distinguishes short-lived detection failures from AEs that persist across multiple detection cycles, providing insight into how AV systems adapt to previously unseen perturbations over time.

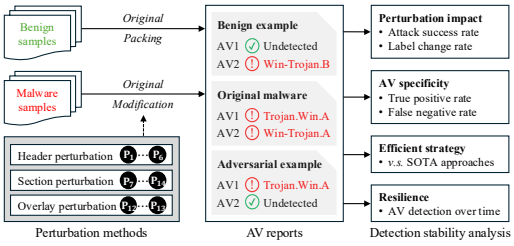


Figure 3: Overview of RAVEN.

3.2 Overview of RAVEN

Figure 3 illustrates the overall workflow of RAVEN. To model realistic false-positive scenarios, we include both unpacked and packed benign samples. We identify 12 representative perturbations from prior work and introduce two novel techniques targeting underexplored PE regions, resulting in 14 perturbations in RAVEN. These perturbations generate semantics-preserving and executable AEs, ensuring that detection failures reflect engine instability rather than malformed inputs. RAVEN then collects detection results from multiple AV engines via VT and analyzes them across four dimensions, namely perturbation impact, AV specificity, strategy efficiency, and resilience.

3.3 Perturbation Methods

Table 1 summarizes the 14 fine-grained, semantics-preserving perturbations that RAVEN implements. We define semantic preservation as preserving the loaded image and runtime behavior under

Table 1: Perturbation methods in RAVEN.

Region	ID	Perturbations	Description
Header	P_1	Modify DOS header(MDH)	Modify selected DOS header fields with random values
	P_2	Extend DOS stub (EDS)	Append NOP instructions and random data to the DOS stub
	P_3	COFF header (CH)	Modify timestamps and symbol pointers in the COFF header
	P_4	Rich header (RH)	Alter Rich header metadata and inject build information
	P_5	Optional header (OH)	Modify checksum and debug information fields
	P_6	Section rename (SRN)	Randomly rename selected PE sections
Section	P_7	Section add (SAD)	Add a dummy section containing random content
	P_8	Section append (SAP)	Fill padding areas within sections with random bytes
	P_9	Content shifting (CS)	Insert gaps between sections and update offsets
	P_{10}	Jump overlay back (JOB)	Redirect execution using JMP instructions
	P_{11}	Instruction change (IC)	Replace instructions with semantically equivalent alternatives
Overlay	P_{12}	Overlay append (OAP)	Append random content to the overlay region
Novel	P_{13}	**Printable string change (PSC)**	**Modify printable strings and DLL names in `.rsrc` and overlay**
	P_{14}	**Section increase (SIN)**	**Increase `SizeOfRawData` and `VirtualSize` to shift section offsets**

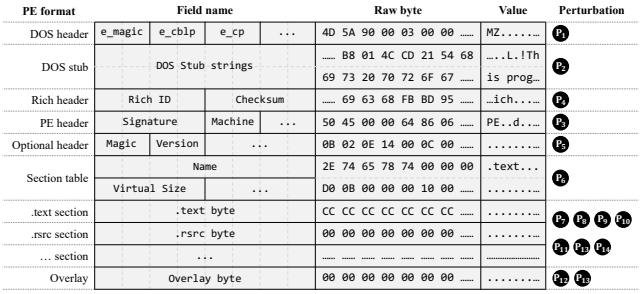


Figure 4: Windows PE structure and perturbation targets.

operator-specific constraints. Operators that modify non-executed or loader-ignored regions preserve behavior by construction. For operators that affect executable content or layout metadata, RAVEN applies them only when the loaded image and execution semantics remain unchanged. It updates affected offsets and references after layout-changing perturbations, preserves outputs and CPU flags for instruction substitutions, modifies strings only when executable code does not reference them, and uses entry-point redirection that transfers control back to the original entry point without altering program state. We exclude packing-based transformations because they rewrite the entire binary and shift evaluation toward packer recognition, making it difficult to attribute detection changes to specific malware content.

Instead, RAVEN applies controlled perturbations across multiple PE regions, as illustrated in Figure 4. We derive 12 representative perturbations from 78 non-packing variants in prior work [9, 10, 17, 18, 20, 25, 30, 33, 34, 39, 43, 53, 55] by eliminating semantic redundancies. These perturbations cover header-, layout-, and instruction-level transformations and serve as engine-agnostic measurement probes rather than detector-specific attack optimizations.

Novel Perturbations. Guided by the detection-surface analysis in § 2, we introduce two additional operators that target detection-relevant but underexplored surfaces: PSC and SIN. PSC targets printable strings and imported DLL-name surfaces in `.rsrc` and overlay regions. Unlike prior obfuscation-based approaches [11, 12], PSC applies functionality-preserving string randomization and DLL-name case modifications without altering the PE structure. SIN shifts section offsets through alignment-preserving updates to `SizeOfRawData` and `VirtualSize`, while preserving section boundaries and executability. Together, PSC and SIN serve as controlled probes for measuring content- and layout-level detection stability.

Listing 1: Printable string change example.

```
[-] 48 65 6C 6C 6F 20 57 6F 72 6C 64 21 0A 00 00 00
      ;"Hello World!"
[+] 75 6E 63 73 6F 6F 70 39 74 61 68 71 0A 00 00 00
      ;"uncsoop9tahq"
...
[-] 56 43 52 55 4E 54 49 4D 45 31 34 30 2E 64 6C 6C
      ;"VCRUNTIME140.dll"
[+] 76 63 72 75 6E 74 69 6D 65 31 34 30 2E 64 6C 6C
      ;"vcruntime140.dll"
```

4 Evaluation

4.1 Evaluation Setup

4.1.1 Dataset. We construct a dataset of Windows PE malware from VT [46], SOREL-20M [23], and DikeDataset [19]. SOREL-20M originally disarmed some samples by zeroing critical `PE header` fields (e.g., `NTHeader.Machine` and `OptionalHeader.Subsystem`). We restore these fields to enable realistic execution and perturbation analysis while preserving functionality. To mitigate label drift [49, 56], we retain samples that have been publicly available on VT for more than three years and group them into malware families using AVClass2 [41]. We select families with sufficient samples and retain only those detected by at least 25 AV engines to ensure high-confidence labels. Consistent with the exclusion criterion in § 3.3, we also exclude packed malware (e.g., "packed," "pack," or "UPX") from the original malware dataset. Packing introduces an unpacking layer that makes the original code not directly observable and makes it difficult to attribute detection changes to controlled perturbations rather than packer artifacts.

Original Malware (M_O). We construct two malware datasets for different evaluation purposes. Malware-I contains 5,000 samples (50 per family) for perturbation-based evaluation (§ 4.2.1–4.2.3), and Malware-II contains 2,000 samples across 20 families (100 per family) for longitudinal analysis (§ 4.2.4).

Benign (B). To evaluate AV specificity, we collect benign samples following prior work [33], including 2,194 unique executables from clean Windows 8, 10, and 11 installations and benign samples from DikeDataset. To reflect the prevalence of packed software in real environments, we additionally pack 2,711 binaries using UPX [44] and PEtite [36], following prior evaluation methodologies [1, 31]. The final benign dataset contains 4,905 samples for AV-specificity analysis.

Adversarial Examples (M_{AE}). We generate adversarial variants from the M_O dataset under different perturbation settings, as summarized in Table 2:

- **RAVEN-Single** Individual perturbations applied to Malware-I samples.
- **RAVEN-Multi** Perturbation combinations applied to Malware-I samples using four representatives selected from perturbation groups with similar AV impact.
- **RAVEN-Longitudinal** Perturbations applied to Malware-II samples for evaluating AV response latency over time.

Table 2: Summary of our dataset.

Type	Dataset	Samples	Families
Original malware	Malware-I	5,000	100
	Malware-II	2,000	20
Benign	-	4,905	-
Adversarial example	Gym-EMBER	1,038	90
	Gym-MalConv	2,407	99
	MAB-EMBER	2,752	99
	MAB-MalConv	2,103	98
	MMO	1,536	77
	Malguise-MalConv	563	47
	Malguise-MalGraph	4,897	100
	RAVEN-Single	69,833	100
	RAVEN-Multi	54,205	100
	RAVEN-Longitudinal	1,724	20

Table 3: Behavioral consistency check for executable original–perturbed pairs.

Behavioral signal	Executed pairs	Matched pairs	Match rate
API/syscall traces	62	62	100%
Dropped files	62	62	100%
Network indicators	62	62	100%

- **SOTA Baselines** AEs generated using Gym [3], MAB [43], MMO [34], and Malguise [33] against ML-based detectors (EMBER [4], MalConv [38], and MalGraph [32]).

Some samples exhibit malformed or non-standard PE structures (e.g., inconsistent headers, invalid section metadata, or missing code regions) that prevent reliable perturbations. We exclude these samples during AE generation, resulting in slightly fewer AEs than the theoretical maximum.

Behavioral Validation. From the RAVEN-Single set in Table 2, we randomly sampled 100 original–AE pairs for behavioral validation beyond PE executability. Among them, 62 originals completed execution in the Pin-based sandbox. The remaining 38 originals did not complete under the sandbox conditions and therefore provided no reliable behavioral baseline for original–AE comparison. For the 62 originals that completed execution, all 62 corresponding AEs also completed execution in the same sandbox. These 62 executable original–AE pairs matched in API/syscall traces, dropped files, and network indicators, except for non-semantic address-valued arguments. Table 3 summarizes the results, and Supplementary Section F details the execution accounting, matching criteria, trace-log format, and validation logs.

4.1.2 Target AV Engines. We evaluate perturbation impact using ClamAV [14] as a local signature-based baseline and VT [46] to access more than 70 commercial AV engines. Although discrepancies may exist between VT and endpoint deployments [56], we additionally perform endpoint validation to examine whether the main VT-observed trends also appear under controlled local scanning (see Figure 11). This validation assesses trend consistency in a controlled on-demand setting, rather than modeling all deployment-time factors such as real-time protection, user interaction, or vendor-specific cloud telemetry. We use VT as our primary evaluation platform because it provides broad multi-engine coverage under a consistent query interface.

4.1.3 Evaluation Metrics. **Attack Success Rate (ASR)** is widely used in prior work [33, 34, 39, 43, 53]. It measures the proportion of AEs that evade detection by a target AV engine, among original malware samples that are correctly detected.

$$\text{ASR} = \frac{|\{z \mid f_{detect}(z) = 1,\ f_{detect}(z_{AE}) = 0\}|}{|\{z \mid f_{detect}(z) = 1\}|} \times 100$$

where z denotes an original malware sample, z_{AE} denotes its corresponding AE, and $f_{detect}(\cdot)$ returns 1 if a sample is detected by the AV engine and 0 otherwise.

Label Change Rate (LCR) measures classification instability among samples whose original malware is detected. We compute LCR when the AE also remains detected but receives a different label from the original malware sample.

$$\text{LCR} = \frac{|\{z \mid f_{label}(z_{AE}) \neq f_{label}(z)\}|}{|\{z \mid f_{detect}(z) = 1\}|} \times 100$$

where $f_{label}(\cdot)$ returns the label assigned by an AV engine. ASR is our primary metric for detection instability, while LCR serves as an auxiliary signal of label-level volatility among samples that remain detected after perturbation. AV labels are vendor-specific, use different naming vocabularies, and can change over time for the same sample [41, 49, 56]. Therefore, we interpret LCR only as an operational signal for malware triage and cross-vendor consistency analysis, not as taxonomic ground truth or evidence of semantic or behavioral change. Together, ASR and LCR capture complementary inconsistencies under semantics-preserving metamorphic transformations: detection loss and label-level volatility [5]. In practice, we aggregate these inconsistencies by computing the average ASR and LCR across multiple VT engines.

True positive rate (TPR) is the fraction of malware samples in M that an engine detects as malicious. **False positive rate (FPR)** is the fraction of benign samples in B that the engine flags as malicious. Analyzing the trade-off between these metrics reveals each AV engine's detection focus, whether it prioritizes minimizing false negatives (aggressive detection) or reducing false positives (conservative detection).

$$\text{TPR} = \frac{|\{s \in M \mid f_{detect}(s) = 1\}|}{|M|} \quad \text{FPR} = \frac{|\{s \in B \mid f_{detect}(s) = 1\}|}{|B|}$$

where s, M, and B denote a binary sample, the set of malware samples, and the set of benign samples, respectively.

4.2 Evaluation Results & Analysis

4.2.1 Answer to RQ1 (Perturbation impact). To evaluate the impact of each perturbation, we compute ASR and LCR by comparing detection results between Malware-I samples and their corresponding RAVEN-Single.

Content vs. Structural Impact. Several section- and content-level perturbations yield substantially higher ASR and LCR than header-based perturbations across both ClamAV and VT (Table 4). In particular, PSC and SAP achieve the highest ASR on ClamAV

Table 4: ASR and LCR for each perturbation method across four detection systems.

		ClamAV				VirusTotal*				EMBER*		MalConv*	
ID	Perturbation	ASR % (#)		LCR % (#)		ASR % (#)		LCR % (#)		ASR % (#)		ASR % (#)	
P_1	MDH	0.76%	(38)	1.91%	(95)	7.70%	(358)	13.57%	(679)	0.37%	(18)	0.0%	(0)
P_2	EDS	1.40%	(70)	2.47%	(122)	12.62%	(594)	19.51%	(967)	1.38%	(67)	5.14%	(173)
P_3	CH	1.40%	(70)	1.54%	(76)	7.93%	(369)	12.94%	(646)	0.97%	(47)	0.86%	(29)
P_4	RH	1.38%	(69)	1.40%	(69)	2.43%	(114)	4.44%	(222)	0.56%	(27)	0.83%	(28)
P_5	OH	1.40%	(70)	1.54%	(76)	8.51%	(398)	14.61%	(729)	2.62%	(127)	0.83%	(28)
P_6	SRN	1.58%	(79)	1.77%	(87)	6.54%	(306)	14.67%	(732)	0.58%	(28)	0.83%	(28)
P_7	SAD	1.12%	(56)	0.22%	(11)	6.73%	(319)	13.46%	(673)	0.06%	(3)	1.28%	(43)
P_8	SAP	17.79%	(888)	2.29%	(94)	16.67%	(800)	9.66%	(482)	3.67%	(178)	0.51%	(17)
P_9	CS	5.04%	(247)	2.30%	(107)	11.29%	(539)	23.34%	(1,144)	1.42%	(69)	3.66%	(123)
P_{10}	JOB	3.29%	(164)	2.46%	(119)	17.20%	(820)	25.1%	(1,253)	2.29%	(111)	2.76%	(93)
P_{11}	IC	7.35%	(367)	11.95%	(553)	20.14%	(964)	**30.64%**	(1,530)	0.43%	(21)	3.51%	(118)
P_{12}	OAP	0.68%	(34)	0.14%	(7)	6.23%	(291)	11.37%	(568)	0.04%	(2)	0.03%	(1)
P_{13}	PSC	**28.72%**	(1,436)	**28.25%**	(1,007)	**20.17%**	(963)	26.65%	(1,332)	**7.09%**	(344)	**8.62%**	(290)
P_{14}	SIN	8.17%	(408)	2.05%	(94)	11.83%	(563)	22.83%	(1,140)	1.36%	(66)	4.19%	(141)

* VirusTotal results are averaged over all engines; EMBER and MalConv report ASR only.

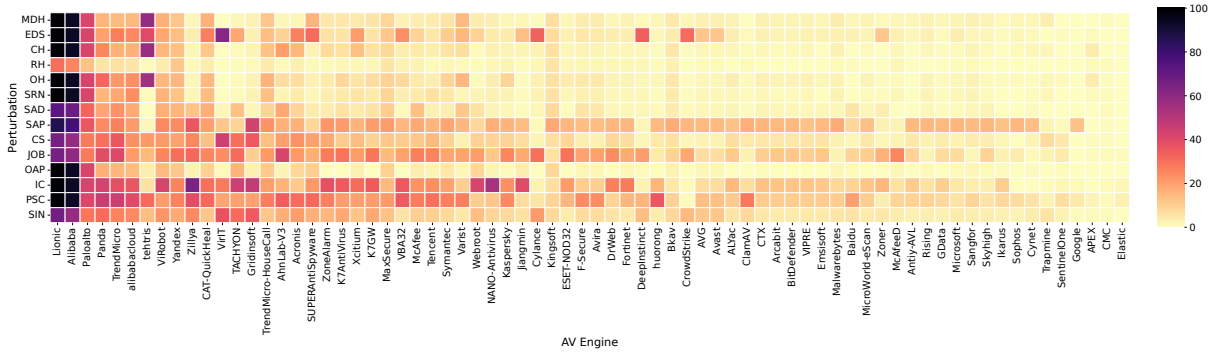


Figure 5: ASR of perturbation methods across AV engines in VirusTotal.

(28.72% and 17.79%), while IC and PSC dominate on VT (20.14% and 20.17%). These results indicate that modifications to content-bearing regions produce substantially stronger detection changes than header-level alterations. While certain header perturbations such as EDS show moderate impact, their effects remain limited and less consistent across engines. Notably, detection changes extend beyond the `.text` section. Perturbations in the `.rsrc` section and overlay also induce substantial ASR and LCR, indicating that these regions serve as detection-critical surfaces. Furthermore, PSC and SAP are effective because they target regions that prior work has explored less extensively.

Model vs. Production AV Behavior. EMBER and MalConv exhibit lower overall ASR but follow a similar regional pattern. PSC remains the most effective perturbation (7.09% on EMBER and 8.62% on MalConv). In contrast, instruction-level changes such as IC show limited impact on EMBER (0.43%) but higher ASR on MalConv (3.51%). This contrast indicates that ML-based detectors exhibit lower sensitivity to instruction-level perturbations compared to production AV engines in our evaluation.

Per-Engine Variability. As shown in Figure 5, AV engines exhibit heterogeneous susceptibility across perturbations. While most engines show high sensitivity to section-based perturbations, a subset of engines (e.g., Lionic, Alibaba) exhibits consistently high ASR across both header and section modifications. This pattern indicates broad susceptibility to structural changes. In contrast, header-based perturbations generally show limited impact across engines. EDS produces the most visible response among header-level operators. Section-level changes, particularly IC and SAP, induce widespread detection shifts across vendors (e.g., Zillya, SUPERAntiSpyware, and Tencent). These results indicate that content-bearing regions serve as a more consistent detection surface, while responses to header modifications remain sparse and engine-specific.

> **Takeaway:** Detection-critical regions extend beyond PE headers and the `.text` section to include `.rsrc` and overlay content. Content-level perturbations targeting these regions induce stronger detection changes than header-level perturbations across both ClamAV and VT engines.

4.2.2 Answer to RQ2 (AV Specificity). While § 4.2.1 quantifies perturbation impact, we examine how these effects translate into detection behavior across AV engines. We measure TPR on Malware-I,

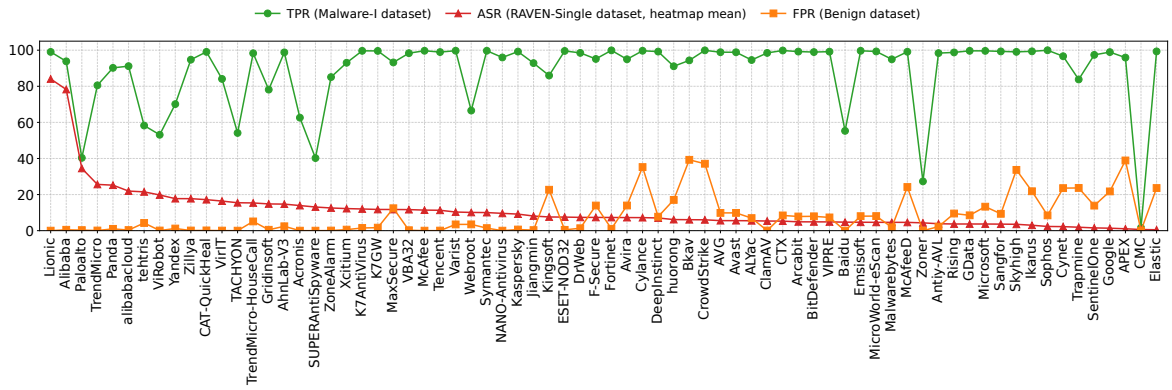


Figure 6: TPR, ASR, and FPR of individual AV engines against the original malware sample, AE, and benign dataset.

ASR on RAVEN-Single, and FPR on benign samples, and jointly analyze these metrics to capture detection behavior across both malicious and benign inputs (Figure 6). Overall, engines with higher ASR against AEs often maintain lower FPR on benign samples, while engines with lower ASR often incur higher FPR. This pattern indicates a trade-off between robustness to AEs and benign precision, and it shows that ASR alone cannot characterize AV robustness.

To further characterize these behaviors, we cluster AV engines using TPR, ASR, and FPR. We group the AV engines into K_{engine} clusters and select $K_{\text{engine}} = 3$ based on the highest silhouette score, capturing distinct detection trade-offs among original-malware coverage, resistance to semantics-preserving perturbations, and benign precision (see Section D in the supplementary material).

Robust AV engines maintain low ASR and low FPR, and they detect both original malware and AEs reliably while avoiding false positives. BitDefender, VIPRE, and Emsisoft each keep ASR near 1% and FPR near 2%.

Over-sensitive AV engines detect most AEs and keep ASR low, but they frequently misclassify benign samples as malicious. Google, Elastic, and Cynet each hold ASR near 0% while FPR reaches 20–40%, suggesting sensitivity to benign structural variation.

Perturbation-sensitive AV engines classify benign samples reliably and keep FPR near 0%, but they fail to detect most AEs. Lionic and Alibaba each show ASR above 75% while FPR stays near 0%, and this combination indicates a focus on benign precision at the expense of robustness to semantics-preserving perturbations.

These profiles suggest that engines expose different operating points between perturbation robustness and benign precision. More robust engines appear to rely on more stable detection patterns that are less sensitive to semantics-preserving perturbations. These profiles correspond to the Balanced (Robust), High-FPR (Over-sensitive), and High-ASR (Perturbation-sensitive) clusters reported in Supplementary Section D.

Takeaway: Similar ASR values can reflect fundamentally different detection behaviors, making ASR alone insufficient to assess AV robustness.

4.2.3 Answer to RQ3 (Efficient strategy). To evaluate perturbation selection strategies for large-scale AV robustness measurement, we construct AEs by combining multiple perturbation methods.

Table 5: Average ASR and LCR of perturbation selection strategies across AV engines on VirusTotal.

Strategies	ASR % (#)		LCR % (#)	
Gym-EMBER	40.39%	(419)	31.79%	(330)
Gym-MalConv	38.49%	(926)	35.41%	(852)
MAB-EMBER	15.13%	(416)	16.84%	(464)
MAB-MalConv	13.05%	(274)	15.90%	(334)
MMO	24.46%	(1,197)	27.93%	(1,368)
Malguise-MalConv	26.37%	(405)	34.82%	(535)
Malguise-MalGraph	30.48%	(171)	36.63%	(206)
IC+CS	25.22%	(1,210)	39.61%	(1,940)
IC+JOB	32.04%	(1,536)	42.19%	(2,099)
JOB+CS	25.43%	(1,220)	30.62%	(1,499)
PSC+CS	25.57%	(1,226)	36.10%	(1,768)
PSC+IC	34.60%	(1,657)	46.27%	(2,307)
PSC+JOB	32.61%	(1,562)	35.78%	(1,783)
IC+JOB+CS	35.21%	(1,688)	43.71%	(2,137)
PSC+IC+CS	37.25%	(1,783)	48.23%	(2,358)
PSC+IC+JOB	44.45%	(2,128)	50.40%	(2,509)
PSC+JOB+CS	35.88%	(1,718)	39.08%	(1,908)
PSC+IC+JOB+CS	**45.40%**	(2,172)	**50.92%**	(2,481)

We cluster perturbation methods based on similarity in their evasion patterns across AV engines to identify effective combinations, following prior work [49]. This clustering is separate from the AV-engine clustering in § 4.2.2; here the data points are the 14 perturbation methods, not AV engines. For each perturbation, we record a sample-level evasion outcome for every sample–AV pair: 1 when the original sample is detected but the AE evades detection, 0 when both remain detected, and −1 when the original sample is not detected. We restrict to the sample–AV pairs with AE reports for all 14 perturbations, and represent each perturbation as the vector of its evasion outcomes over this common set of pairs. We apply K-means clustering to these vectors and select the number of perturbation clusters K by the silhouette coefficient, reporting scores for $K \in \{2, \ldots, 7\}$ in Section G of the supplementary material. Since the scores are low and nearly flat near the maximum, we use the clustering only to reduce the perturbation combinations to a small candidate set rather than as an optimal taxonomy or a claim about a unique K. Using $K = 4$, we select from each cluster the perturbation with the best balance between ASR and LCR, yielding PSC, IC, JOB, and CS.

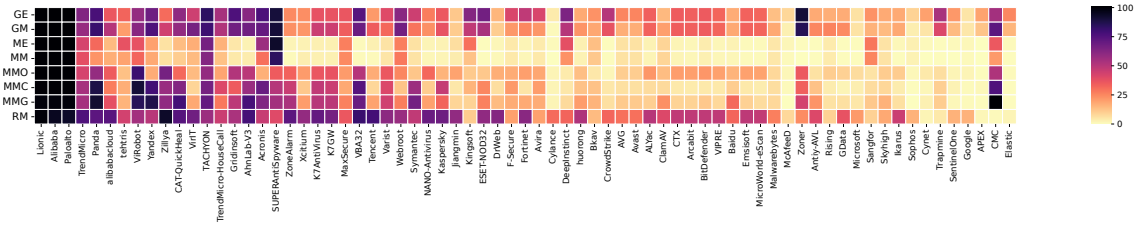


Figure 7: ASR of AV engines across perturbations: Gym-EMBER (GE), Gym-MalConv (GM), MAB-EMBER (ME), MAB-MalConv (MM), MalwareMakeover (MMO), Malguise-MalConv (MMC), Malguise-MalGraph (MMG), RAVEN-Multi (RM).

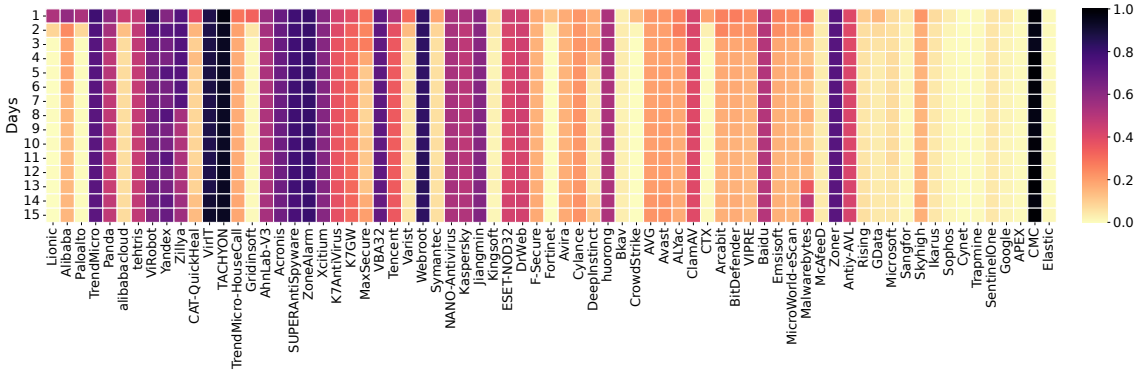


Figure 8: Daily ASR of all AV engines over a 15-day period.

To confirm that the RQ3 conclusion does not depend on this partition, we evaluate the selected combination and its sub-combinations directly (Table 5). PSC+IC+JOB+CS yields 45.40% ASR and 50.92% LCR; removing CS (PSC+IC+JOB) still yields 44.45% ASR, and removing the strongest single operator PSC (IC+JOB+CS) still yields 35.21% ASR. The effect therefore arises from combining perturbations across distinct modification surfaces rather than from any single operator or a particular cluster assignment. We interpret Table 5 as a robustness-measurement comparison rather than an attack-optimization benchmark. Gym-EMBER also yields substantial ASR (40.39%), showing that prior AE frameworks can reveal instability when their perturbation surfaces overlap with production AV detection surfaces. We therefore select PSC+IC+JOB+CS as a representative multi-surface configuration for longitudinal robustness measurement.

We further analyze the ASR of each perturbation selection strategy across AV engines to assess per-engine impact, as shown in Figure 7. Zillya, NANO-Antivirus, Kaspersky, and Tencent exhibit high ASR across most strategies, indicating broad susceptibility to structural manipulation. Gym and MAB affect SUPERAntiSpyware most strongly, consistent with earlier observations that these strategies heavily affect header-related regions. Similarly, ViRobot and Panda show relatively higher ASR under Malguise, consistent with earlier results showing stronger detection changes when perturbations target the `.text` section (see Figure 5). Notably, the High-FPR detection profile in § 4.2.2 includes engines with low ASR under RAVEN-Single (see Table 2 in the supplementary material). Some of these engines, including Trapmine, CrowdStrike, and Kingsoft, show relatively higher ASR under Gym, indicating that robustness does not generalize uniformly across perturbation strategies.

> **Takeaway:** No single perturbation strategy consistently characterizes AV robustness across engines. Multi-surface perturbation combinations provide a stronger robustness-measurement lens because they expose detection instability that single-surface strategies miss.

4.2.4 Answer to RQ4 (Resilience). To analyze how quickly AV engines adapt to newly crafted AEs, we track detection latency and state transitions for each AE–AV pair across repeated rescans. For this purpose, we construct the RAVEN-Longitudinal dataset with 1,724 AEs that RAVEN generates using the representative perturbation combination from § 4.2.3 (PSC+IC+JOB+CS) on samples from the Malware-II dataset. We perform daily rescans over a 15-day period and collect VT reports at each rescan to track detection updates across engines. Measuring ASR daily allows us to analyze how detection performance evolves over time and reveals engine-specific delays in adapting to newly generated AEs.

Overall Results. Figure 8 presents the daily ASR for each AV engine. We observe heterogeneous temporal responses across engines. At the initial submission, 41,749 AE–AV pairs exhibit initial evasion, where the AV engine detects the original malware sample but misses the corresponding AE. Within the 15-day observation window, the same AV engines re-detect 12,606 of these 41,749 pairs. Among these 12,606 recovered pairs, the first rescan re-detects 64.91%. Over the 15-day window, 76 distinct VT engines returned at least one verdict, and we report longitudinal results over this engine set. Several engines show sharp early ASR reductions. Six

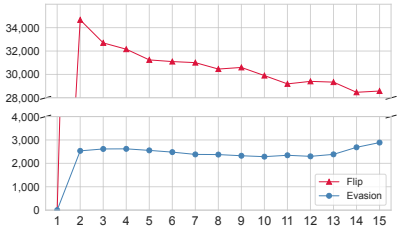


Figure 9: Temporal dynamics of flip and evasion.

of 76 engines (8%) reduce ASR by more than 25 percentage points after the first rescan, with the largest reduction reaching about 0.44. For example, Paloalto and Lionic drop from about 0.52 initial ASR to below 0.09 after one day and below 0.01 after two days. In contrast, Zillya maintains high ASR through Day 7 and then drops on Day 8 from about 0.73 to about 0.54, reflecting delayed adaptation. Fifteen days after initial submission, 20 of 76 engines (26%) still show ASR above 0.50, indicating persistent evasion despite rapid early recovery by engines such as Paloalto and Lionic. Meanwhile, engines such as Cynet, Google, McAfeeD, and Trapmine maintain consistently low ASR throughout the 15-day window, indicating stable detection performance.

Detection Latency per AE–AV Pair. We define a "flip" as an AE–AV pair that is initially missed but later detected, and an "evasion" as a pair in which the original is detected but the AE is missed. Figure 9 illustrates the temporal dynamics of these transitions. Flip counts peak early, reaching approximately 34k on Day 2, and then decrease to about 28–29k by the end of the observation window. This pattern aligns with the rapid ASR decline in Figure 8 and shows that many detection corrections occur shortly after the first rescan. Evasion events remain lower in volume, around 2.3k–2.9k, but persist across the observation period, indicating that some AEs remain undetected across repeated rescans. We also observe reversions to evasion: 5,745 AE–AV pairs show at least one transition from detected back to undetected. These pairs account for 4.61% of AE–AV pairs whose original sample was detected and span 64 of the 76 AV engines in the RAVEN-Longitudinal evaluation.

AV-wise Transition Frequency. Figure 10 shows that flip and evasion events are highly engine-specific. Figure 10a shows that Gridinsoft, Malwarebytes, and MaxSecure exhibit high flip counts, indicating frequent correction after initial misses. In contrast, Figure 10b shows that VBA32, ZoneAlarm, and VirIT accumulate high evasion counts, indicating persistent or incomplete adaptation. CrowdStrike, Cynet, and Google show low flip and evasion counts together with low ASR, reflecting stable detection behavior. These patterns separate engines into rapid-response, persistent-evasion, and stable-detection profiles, while the reversion events above capture volatile detection where recovery does not persist.

Takeaway: AE durability depends on temporal detection behavior, not only on initial evasion success. Longitudinal analysis reveals rapid-response, persistent-evasion, and volatile-detection profiles that snapshot ASR alone cannot capture.

4.2.5 Endpoint Validation. We examine whether VT-observed instability trends also appear under controlled local scanning. We use endpoint validation as a trend-level sanity check for the VT-scale results, not as a full deployment-mode comparison. We evaluate five endpoint AV products[2] and ClamAV as local baseline, with each endpoint AV product installed on a separate isolated Windows 10 VM. We disable real-time protection and Windows Defender, and revert each VM state after each experiment. For the VT side, we use the same population as in § 4.2.4 and compute ASR over the full 15-day observation window. We compute FPR on the fixed benign population from § 4.1.1.

Figure 11 shows that the ASR–FPR ordering is largely preserved under local scanning. Kaspersky and ClamAV remain the most AE-sensitive engines in both settings, while Avast and AVG remain the most benign-sensitive engines. This pattern is consistent with the perturbation-sensitivity/FPR trade-off in § 4.2.2. TPR shows weaker correspondence. The Malwarebytes engine hosted on VT detects only 58.4% of original samples, whereas the locally installed Malwarebytes endpoint detects 99.1%. We therefore treat endpoint validation as evidence for the ASR–FPR trend, not as evidence that VT-hosted engines and endpoint AV products are equivalent.

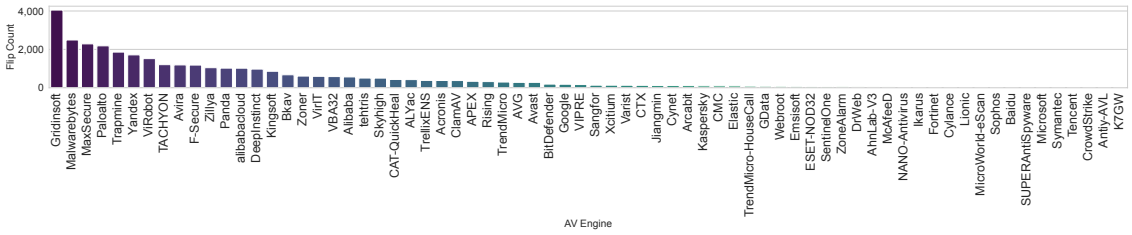
Takeaway: Endpoint validation shows similar ASR–FPR trends to the VT-scale results. We interpret these ASR–FPR trends as supporting evidence that the observed instability is not solely a VT-specific artifact within our controlled on-demand setting.

5 Discussion

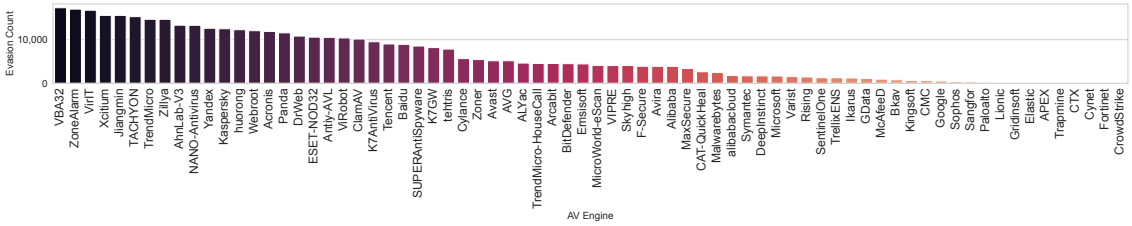
Fundamental Limits of Semantics-Preserving Evasion. Semantics-preserving perturbations substantially degrade detection but rarely defeat it completely: RAVEN-Multi achieves 45.40% average ASR across VT engines, while complete evasion (100% ASR) occurs in fewer than 3% of cases. This limitation reflects two practical constraints. First, RAVEN leaves execution-critical and semantically ambiguous regions unchanged, including entry-point code, control-flow targets, loader-relevant metadata, and writable/executable sections with ambiguous semantics. A byte-level accounting over Malware-I seed binaries shows that these regions account for 17.48% of binary bytes on average. Second, some ClamAV byte-pattern signatures span multiple adjacent instructions, as shown in Listing 2. Disrupting them through local instruction substitution can alter branch structure, flags, or operand semantics and therefore violates the constraints that § 3.3 defines. Together, these unmodified regions and cross-instruction patterns act as stable detection anchors and help explain why substantial evasion remains possible but complete evasion remains rare.

Detection Mechanisms and Trade-offs. Signature-based detection remains attractive because it costs under 2% CPU overhead, compared with 10–30% for behavioral or ML-based alternatives [7]. Raw byte-level signatures drive 94.54% of ClamAV detections, and Table 4 shows that content-level perturbations achieve higher ASR than header-only ones. For example, the RAVEN-Single operators PSC, SAP, SIN, CS, and IC probe distinct content, layout, and

[2]Avast Free Antivirus 32.11.8635, AVG Free 25.9.10453, Malwarebytes Free 5.3.7.131, Bitdefender Total Security 27.0.54.271, and Kaspersky Free 21.21.7.384.



(a) AV engines ranked by the number of detection flips.



(b) AV engines ranked by cumulative evasion counts.

Figure 10: AV-specific patterns in flip and evasion events.

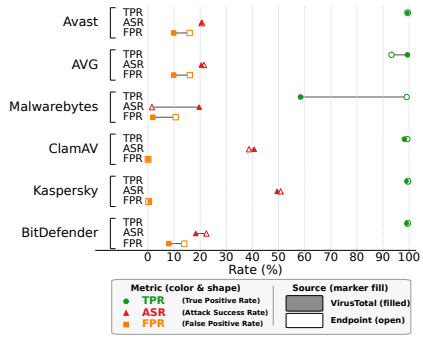


Figure 11: TPR, ASR, and FPR for endpoint AV products and the local ClamAV baseline, compared with VirusTotal results.

Listing 2: Operand-level signature limiting IC.

```
1  .text:00ED534D    80 F9 75       cmp  cl, 75h  ; 'u'
2  .text:00ED5350    75 2A          jnz  short loc_ED537C
3  .text:00ED5352    3C 23          cmp  al, 23h  ; '#'
4  .text:00ED5354    75 E0          jnz  short loc_ED5336
```

instruction-level detection surfaces. This surface mapping helps explain the ASR–FPR profiles in § 4.2.2. High-ASR/low-FPR engines appear brittle to localized content or layout surfaces. Low-ASR/high-FPR engines suppress evasion by reacting to broader structural signals, but incur more benign false positives. Low-ASR/low-FPR engines are consistent with less reliance on broad structural over-detection. This surface interpretation follows directly from the measured evasion data. On VirusTotal, the operators with the highest average ASR are content-, layout-, and instruction-level ones (PSC, IC, JOB, SAP, SIN, and CS in Table 4), whereas most header operators such as MDH, CH, and RH stay below 8%, with EDS the main exception among header operators. The High-ASR profile engines (Table 2 in Supplementary Section D) are precisely those that lose detections under these content and layout operators rather than under header changes, which explains why they combine high ASR with low FPR: their detection appears anchored to specific content or layout patterns rather than to broad structural signals. This account rests on the evasion measurements themselves, independent of any vendor documentation. As additional context, we cross-referenced a subset of engines with publicly documented detection compositions (Supplementary Section I). The documented ML-first engines in our subset appear in the High-FPR profile [15, 16, 21, 47]. In contrast, many documented multi-layer hybrid engines fall in the Balanced profile [6, 8, 45]. We limited the mapping to engines whose public documentation described concrete detection components. We added four documented ML-first engines to represent that category and therefore treat the observed correspondence as exploratory rather than independent evidence. The documented signature-centric Paloalto scanner appears in the High-ASR profile [48]. Multiple counterexamples remain; documented composition alone therefore does not determine the observed profiles, and the surface-level evidence above provides the more direct account with the documentation offering only weak corroboration. ClamAV provides a white-box anchor for plausible mechanisms. The ASR–FPR trends also remain consistent across VT and endpoint settings (§ 4.2.5), suggesting that the observed instability is not solely a VT-specific artifact. Our analysis supports a surface- and composition-level interpretation of the observed detection profiles. However, production engines expose only detection outcomes, not their internal detection logic. We therefore cannot attribute an individual verdict to a specific vendor rule, threshold, model feature, or update component without vendor-internal access.

Temporal Adaptation and AV Update Behaviors. Our 15-day longitudinal study shows that AV engines differ not only in which perturbations they fail against, but also in how quickly and stably they recover. *Rapid-response engines* reduce ASR shortly after the first rescan, suggesting quick adaptation to new AEs through signature, model, heuristic, or backend-rule updates. *Persistent-evasion engines* continue to miss many AEs across repeated rescans, indicating slower or incomplete responses to the representative multi-surface perturbation. *Volatile-detection engines* show reversions to evasion, where an AE is detected at one rescan but later missed again. These temporal profiles complement the perturbation and detection-profile analyses in § 4.2.1–§ 4.2.3 and Supplementary Sections D and E. They show that AV robustness depends on both surface sensitivity and temporal recovery stability.

Implications for Robustness Evaluation. Taken together, the results in § 4.2.1–§ 4.2.4 show that snapshot-based metrics such as single-point ASR cannot characterize real-world AV robustness. This challenges the common assumption that evasion success alone reliably reflects detector robustness. Robustness evaluation should therefore report which perturbation surfaces trigger detection loss. It should evaluate per-perturbation ASR alongside FPR to distinguish robust detection from over-sensitive detection. Longitudinal analysis should distinguish short-lived gaps from persistent evasion or unstable transitions. These diagnostics identify fragile detection surfaces, false-positive trade-offs, delayed re-detection, and cross-engine variability.

Limitations and Future Directions. Several limitations remain. First, our black-box evaluation focuses on static detection and excludes packed malware because packing introduces confounding signals and major structural changes. Second, behavioral validation covers the executable portion of a randomly sampled subset rather than the full AE corpus. Third, our PE-focused results may not generalize to other formats such as ELF or Mach-O. Fourth, endpoint validation shows consistent ASR–FPR trends, but VT's cloud-based environment may differ from local deployments. For example, VT's hosted Malwarebytes engine reports a substantially lower TPR than the installed endpoint product (§ 4.2.5). This gap indicates that VT-hosted engines do not always reflect the detection capability of their commercial counterparts. Finally, our longitudinal study relies on repeated VT submissions. Thus, the measured latencies characterize re-detection behavior within VT; endpoint update behavior may differ in ordinary deployments. Future work may extend this analysis to dynamic perturbations and other binary formats.

6 Related Work

VT Measurement. Prior VT-based measurement studies show that AV engines may return different detection results or labels for the same samples over time. Wang et al. [49] analyzed large-scale VT reports and reported frequent label fluctuations across engines and scans. Zhu et al. [56] showed that over 40% of single-engine labels change over time due to short-lived "hazard flips" and inter-engine correlations. Other studies further report delayed detections and inconsistent labeling across engines and threat intelligence feeds [26, 35]. These studies provide important observational evidence that AV detection and labeling are unstable.

Unlike prior studies that observe natural label drift on identical samples, RAVEN holds the original malware fixed and varies semantics-preserving perturbations. This interventional original–perturbed pair design attributes instability to specific perturbation operators and PE regions, while measuring longitudinal re-detection behavior on fixed AEs. Thus, RAVEN complements prior VT measurement studies by adding controlled attribution of detection instability under semantics-preserving changes.

AE Generation and Evaluation. Prior adversarial malware frameworks mainly generate AEs to evade specific malware classifiers. Byte-level methods modify raw bytes without modeling PE structure [9, 10]; PE-structure methods manipulate headers, sections, or metadata [18, 24]; and semantic methods modify control flow or instructions while preserving functionality [32, 54]. However, these frameworks typically optimize for ML-based detectors or single-shot evasion success, rather than measuring detection stability across AV engines (see Section H in the supplementary material). Metamorphic testing evaluates robustness through semantics-preserving transformations and output consistency [5, 52], and recent work applies this view to Wasm malware detectors [28]. RAVEN extends this line of work to Windows PE malware and real-world AV engines by using detection-surface-guided perturbations as engine-agnostic measurement probes. It further captures ASR–FPR trade-offs and longitudinal re-detection behavior, which single-target or snapshot-based AE generation does not expose.

7 Conclusion

This work highlights a gap between academic adversarial evaluation settings and the detection behavior of deployed AV engines. While prior studies have primarily emphasized ML-based detectors and header-level perturbations, our analysis shows that signature-based detection in content-bearing regions remains important in AV detection pipelines. Region-aware perturbations therefore yield substantially higher evasion rates in our evaluation and expose trade-offs between evasion resistance and benign precision.

Our findings further show that single-point detection rates are insufficient for evaluating AV robustness under semantics-preserving perturbations. Detection outcomes vary across semantically equivalent variants and over time, revealing both structural sensitivity and temporal variability. Controlled endpoint validation shows similar ASR–FPR trends under on-demand scanning, supporting the conclusion that the observed ASR–FPR trends are not solely VT-specific artifacts. Together, these results characterize AV robustness as a property of decision stability under behavior-preserving structural variation and temporal consistency, rather than static detection outcomes alone.

Acknowledgments

We thank the anonymous reviewers for their constructive comments. This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grants (RS-2026-25530213, RS-2024-00439819, and RS-2025-25394739) funded by the Korean government (MSIT) and the U.S. National Science Foundation (CNS-2440819 and DGE-2335798). During the preparation of this work, the authors used OpenAI ChatGPT to improve language and readability. The authors reviewed and edited the

resulting text and take full responsibility for the content of this publication.

Data Availability Statement

We publicly release the source code and analysis scripts at https://github.com/byhoon6279/RAVEN.git, and the supplementary materials at https://doi.org/10.5281/zenodo.19229602. Due to security and legal constraints, we do not release raw malware binaries or generated AEs. We instead provide cryptographic hashes and processed analysis data, subject to VirusTotal's data-sharing restrictions and applicable legal requirements.

References

[1] Hojjat Aghakhani, Fabio Gritti, Francesco Mecca, Martina Lindorfer, Stefano Ortolani, Davide Balzarotti, Giovanni Vigna, and Christopher Kruegel. 2020. When Malware is Packin' Heat; Limits of Machine Learning Classifiers Based on Static Analysis Features. In *Network and Distributed System Security Symposium*. Internet Society, San Diego, United States. https://doi.org/10.14722/ndss.2020.24310

[2] Rabie Ahmed, Albia Maqbool, Jihane Ben Slimane, Ahmad Alshammari, Nasser S. Albalawi, and Abdulaziz Alanazi. 2025. Leveraging AI for Automated Malware Classification and Detection in Large-Scale Networks. *Journal of Information Systems Engineering and Management* 10, 9s (2025), 146 – 159. https://doi.org/10.52783/jisem.v10i9s.1175

[3] Hyrum S Anderson, Anant Kharkar, Bobby Filar, David Evans, and Phil Roth. 2018. Learning to Evade Static PE Machine Learning Malware Models via Reinforcement Learning. *arXiv preprint arXiv:1801.08917* (Jan. 2018). https://doi.org/10.48550/arXiv.1801.08917 arXiv:1801.08917 [cs.CR]

[4] H. S. Anderson and P. Roth. 2018. EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. *ArXiv e-prints* (April 2018). https://doi.org/10.48550/arXiv.1804.04637 arXiv:1804.04637 [cs.CR]

[5] Leonhard Applis, Annibale Panichella, and Arie van Deursen. 2022. Assessing robustness of ML-based program analysis tools using metamorphic program transformations. In *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering* (Melbourne, Australia) *(ASE '21)*. IEEE Press, 1377–1381. https://doi.org/10.1109/ASE51524.2021.9678706

[6] Bitdefender. 2026. Antimalware Technology. https://businessresources.bitdefender.com/whitepaper-technologies-used-in-the-antimalware-engine. Accessed: July 2026.

[7] Marcus Botacin, Felipe Duarte Domingues, Fabrício Ceschin, Raphael Machnicki, Marco Antonio Zanata Alves, Paulo Lício de Geus, and André Grégio. 2022. AntiViruses under the microscope: A hands-on perspective. *Computers & Security* 112 (2022), 102500. https://doi.org/10.1016/j.cose.2021.102500

[8] Broadcom (Symantec). 2026. How Symantec Endpoint Protection Technologies Protect Your Computers. https://techdocs.broadcom.com/us/en/symantec-security-software/endpoint-security-and-management/endpoint-protection/all/what-is-v45096464-d43e1648/how-symantec-endpoint-protection-technologies-prot-v97539434-d43e1669.html. Accessed: July 2026.

[9] Raphael Labaca Castro, Corinna Schmitt, and Gabi Dreo. 2019. Aimed: Evolving malware with genetic programming to evade detection. In *IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. IEEE Computer Society, Los Alamitos, CA, USA, 240–247. https://doi.org/10.1109/TrustCom/BigDataSE.2019.00040

[10] Raphael Labaca Castro, Corinna Schmitt, and Gabi Dreo Rodosek. 2019. Armed: How automatic malware modifications can evade static detection?. In *International Conference on Information Management (ICIM)*. IEEE, Cambridge, UK, 20–27. https://doi.org/10.1109/INFOMAN.2019.8714698

[11] Fabrício Ceschin, Marcus Botacin, Heitor Murilo Gomes, Luiz S. Oliveira, and André Grégio. 2020. Shallow Security: on the Creation of Adversarial Variants to Evade Machine Learning-Based Malware Detectors. In *Proceedings of the 3rd Reversing and Offensive-Oriented Trends Symposium* (Vienna, Austria) *(ROOTS'19)*. Association for Computing Machinery, New York, NY, USA, Article 4, 9 pages. https://doi.org/10.1145/3375894.3375898

[12] Fabricio Ceschin, Marcus Botacin, Gabriel Lüders, Heitor Murilo Gomes, Luiz Oliveira, and Andre Gregio. 2021. No need to teach new tricks to old malware: Winning an evasion challenge with xor-based adversarial samples. In *Reversing and Offensive-Oriented Trends Symposium* (Vienna, Austria) *(ROOTS'20)*. Association for Computing Machinery, New York, NY, USA, 13–22. https://doi.org/10.1145/3433667.3433669

[13] Tsong Yueh Chen, Fei-Ching Kuo, Huai Liu, Pak-Lok Poon, Dave Towey, TH Tse, and Zhi Quan Zhou. 2018. Metamorphic testing: A review of challenges and opportunities. *ACM Computing Surveys (CSUR)* 51, 1 (2018), 1–27. https://doi.org/10.1145/3143561

[14] ClamAV. 2024. ClamAV. https://docs.clamav.net/.

[15] CrowdStrike. 2026. Falcon Prevent: Next-Generation Antivirus. https://www.crowdstrike.com/en-us/platform/endpoint-security/falcon-prevent-ngav/. Accessed: July 2026.

[16] Cynet. 2026. Cynet Endpoint Security: AI Static and Behavioral Analysis. https://www.cynet.com/platform/endpoint-security/. Accessed: July 2026.

[17] Luca Demetrio, Battista Biggio, Giovanni Lagorio, Fabio Roli, and Alessandro Armando. 2021. Functionality-preserving black-box optimization of adversarial windows malware. *IEEE Transactions on Information Forensics and Security* 16 (2021), 3469–3478. https://doi.org/10.1109/TIFS.2021.3082330

[18] Luca Demetrio, Scott E Coull, Battista Biggio, Giovanni Lagorio, Alessandro Armando, and Fabio Roli. 2021. Adversarial EXEmples: A Survey and Experimental Evaluation of Practical Attacks on Machine Learning for Windows Malware Detection. *ACM Transactions on Privacy and Security* 24, 4, Article 27 (Sept. 2021), 31 pages. https://doi.org/10.1145/3473039

[19] DikeDataset. 2024. DikeDataset. https://github.com/iosifache/DikeDataset#downloading-step.

[20] Mohammadreza Ebrahimi, Jason Pacheco, Weifeng Li, James Lee Hu, and Hsinchun Chen. 2021. Binary black-box attacks against static malware detectors with reinforcement learning in discrete action spaces. In *IEEE Security and Privacy Workshops (SPW)*. IEEE, San Francisco, CA, USA, 85–91. https://doi.org/10.1109/SPW53761.2021.00021

[21] Elastic. 2025. Malicious File Detected — Elastic Defend. https://www.elastic.co/guide/en/security/8.19/malicious-file-detected-elastic-defend.html. Accessed: July 2026.

[22] Matthew G Gaber, Mohiuddin Ahmed, and Helge Janicke. 2024. Malware detection with artificial intelligence: A systematic literature review. *Comput. Surveys* 56, 6, Article 148 (Jan. 2024), 33 pages. https://doi.org/10.1145/3638552

[23] Richard Harang and Ethan M. Rudd. 2020. SOREL-20M: A Large Scale Benchmark Dataset for Malicious PE Detection. https://doi.org/10.48550/arXiv.2012.07634

[24] Shuai He, Cai Fu, Hong Hu, Jiahe Chen, Jianqiang Lv, and Shuai Jiang. 2024. MalwareTotal: Multi-Faceted and Sequence-Aware Bypass Tactics against Static Malware Detection. In *IEEE/ACM International Conference on Software Engineering (ICSE)* (Lisbon, Portugal) *(ICSE '24)*. Association for Computing Machinery, New York, NY, USA, Article 172, 12 pages. https://doi.org/10.1145/3597503.3639141

[25] Beomjin Jin, Jusop Choi, Hyoungshick Kim, and Jin B. Hong. 2021. FUMVar: a practical framework for generating Fully-working and Unseen Malware Variants. In *Annual ACM Symposium on Applied Computing (SAC)* (Virtual Event, Republic of Korea) *(SAC '21)*. Association for Computing Machinery, New York, NY, USA, 1656–1663. https://doi.org/10.1145/3412841.3442039

[26] Beomjin Jin, Eunsoo Kim, Hyunwoo Lee, Elisa Bertino, Doowon Kim, and Hyoungshick Kim. 2024. Sharing Cyber Threat Intelligence: Does It Really Help?. In *Network and Distributed System Security Symposium (NDSS)*. Internet Society, San Diego, CA, USA. https://doi.org/10.14722/ndss.2024.24228

[27] Minho Kim, Haehyun Cho, and Jeong Hyun Yi. 2022. Large-scale analysis on anti-analysis techniques in real-world malware. *IEEE access* 10 (2022), 75802–75815. https://doi.org/10.1109/ACCESS.2022.3190978

[28] Taeyoung Kim, Sanghak Oh, Kiho Lee, Weihang Wang, Yonghwi Kwon, Sanghyun Hong, and Hyoungshick Kim. 2025. When Does Wasm Malware Detection Fail? A Systematic Analysis of Their Robustness to Evasion. In *2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 2957–2969. https://doi.org/10.1109/ASE63991.2025.00243

[29] Eric Landril, Samuel Valente, Gregory Andersen, and Christopher Schneider. 2024. Ransomware Detection through Dynamic Behavior-Based Profiling Using Real-Time Crypto-Anomaly Filtering. https://doi.org/10.31219/osf.io/nvg65

[30] Kun Li, Wei Guo, Fan Zhang, and Jiayu Du. 2023. GAMBD: Generating adversarial malware against MalConv. *Computers & Security* 130, C (July 2023), 9 pages. https://doi.org/10.1016/j.cose.2023.103279

[31] Shijia Li, Jiang Ming, Pengda Qiu, Qiyuan Chen, Lanqing Liu, Huaifeng Bao, Qiang Wang, and Chunfu Jia. 2023. PackGenome: Automatically Generating Robust YARA Rules for Accurate Malware Packer Detection. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)* (Copenhagen, Denmark) *(CCS '23)*. Association for Computing Machinery, New York, NY, USA, 3078–3092. https://doi.org/10.1145/3576915.3616625

[32] Xiang Ling, Lingfei Wu, Wei Deng, Zhenqing Qu, Jiangyu Zhang, Sheng Zhang, Tengfei Ma, Bin Wang, Chunming Wu, and Shouling Ji. 2022. Malgraph: Hierarchical graph neural networks for robust windows malware detection. In *IEEE Conference on Computer Communications (INFOCOM)*. IEEE, London, UK, 1998–2007. https://doi.org/10.1109/INFOCOM48880.2022.9796786

[33] Xiang Ling, Zhiyu Wu, Bin Wang, Wei Deng, Jingzheng Wu, Shouling Ji, Tianyue Luo, and Yanjun Wu. 2024. A wolf in sheep's clothing: practical black-box adversarial attacks for evading learning-based windows malware detection in the wild. In *USENIX Security Symposium (USENIX Security)* (Philadelphia, PA, USA) *(SEC '24)*. USENIX Association, USA, Article 413, 18 pages.

[34] Keane Lucas, Mahmood Sharif, Lujo Bauer, Michael K Reiter, and Saurabh Shintre. 2021. Malware makeover: Breaking ml-based static analysis by modifying executable bytes. In *ACM Asia Conference on Computer and Communications Security (AsiaCCS)* (Virtual Event, Hong Kong) *(ASIA CCS '21)*. Association for Computing Machinery, New York, NY, USA, 744–758. https://doi.org/10.1145/3433210.3453086

[35] Shang Ma, Chaoran Chen, Shao Yang, Shifu Hou, Toby Jia-Jun Li, Xusheng Xiao, Tao Xie, and Yanfang Ye. 2025. Careful About What App Promotion Ads Recommend: Detecting and Explaining Malware Promotion via App Promotion Graph. In *Network and Distributed System Security Symposium (NDSS)*. Internet Society, San Diego, CA, USA. https://doi.org/10.14722/ndss.2025.230051

[36] Petite. 2024. Win32 Executable Compressor. https://www.un4seen.com/petite/ Accessed: 2025-07-18.

[37] D. L. S. I. Punyasiri. 2023. Signature & Behavior Based Malware Detection. Proposal report, Department of Cyber Security, Sri Lanka Institute of Information Technology. https://doi.org/10.13140/RG.2.2.22127.20640 Research proposal/report. Available via ResearchGate and Scribd.

[38] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles Nicholas. 2018. Malware Detection by Eating a Whole EXE. In *Proceedings of the Workshops of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*. AAAI Press, Palo Alto, California, USA, 268–276. https://doi.org/10.48550/arXiv.1710.09435

[39] Maria Rigaki and Sebastian Garcia. 2023. The Power of MEME: Adversarial Malware Creation with Model-Based Reinforcement Learning. In *European Symposium on Research in Computer Security (ESORICS)*. Springer Nature Switzerland, Cham, 44–64. https://doi.org/10.1007/978-3-031-51482-1_3

[40] V Sai Sathyanarayan, Pankaj Kohli, and Bezawada Bruhadeshwar. 2008. Signature generation and detection of malware families. In *Australasian Conference on Information Security and Privacy (ACISP)* (Wollongong, Australia) *(ACISP '08)*. Springer-Verlag, Berlin, Heidelberg, 336–349. https://doi.org/10.1007/978-3-540-70500-0_25

[41] Silvia Sebastián and Juan Caballero. 2020. Avclass2: Massive malware tag extraction from av labels. In *Annual Computer Security Applications Conference (ACSAC)* (Austin, USA) *(ACSAC '20)*. Association for Computing Machinery, New York, NY, USA, 42–53. https://doi.org/10.1145/3427228.3427261

[42] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. 2017. Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. In *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, Venice, Italy, 618–626. https://doi.org/10.1109/ICCV.2017.74

[43] Wei Song, Xuezixiang Li, Sadia Afroz, Deepali Garg, Dmitry Kuznetsov, and Heng Yin. 2022. Mab-malware: A reinforcement learning framework for blackbox generation of adversarial malware. In *ACM Asia Conference on Computer and Communications Security (AsiaCCS)* (Nagasaki, Japan) *(ASIA CCS '22)*. Association for Computing Machinery, New York, NY, USA, 990–1003. https://doi.org/10.1145/3488932.3497768

[44] UPX. 2024. Ultimate Packer for eXecutables. https://upx.github.io/ Accessed: 2025-07-18.

[45] VIPRE Security. 2022. VIPRE Endpoint Security Cloud: Next-Generation Endpoint Security. https://vipre.com/wp-content/uploads/2023/12/vipre-eb-endpoint-security-cloud-next-generation-endpoint-security-made-simple.pdf. Accessed: July 2026.

[46] VirusTotal. 2012. VirusTotal-Free Online Virus, Malware and URL scanner. https://www.virustotal.com access date: 27 May 2022.

[47] VirusTotal. 2017. VirusTotal += Cylance. https://blog.virustotal.com/2017/07/virustotal-cylance.html. Accessed: July 2026.

[48] VirusTotal. 2017. VirusTotal += Palo Alto Networks. https://blog.virustotal.com/2017/03/virustotal-palo-alto-networks.html. Accessed: July 2026.

[49] Jingjing Wang, Liu Wang, Feng Dong, and Haoyu Wang. 2023. Re-measuring the label dynamics of online anti-malware engines from millions of samples. In *ACM on Internet Measurement Conference (IMC)* (Montreal QC, Canada) *(IMC '23)*. Association for Computing Machinery, New York, NY, USA, 253–267. https://doi.org/10.1145/3618257.3624800

[50] Jia Yan, Xiangkun Jia, Lingyun Ying, Jia Yan, and Purui Su. 2022. Understanding and Mitigating Label Bias in Malware Classification: An Empirical Study. In *International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, Guangzhou, China, 492–503. https://doi.org/10.1109/QRS57517.2022.00057

[51] Jiaqi Yan, Guanhua Yan, and Dong Jin. 2019. Classifying malware represented as control flow graphs using deep graph convolutional neural network. In *Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, Portland, OR, USA, 52–63. https://doi.org/10.1109/DSN.2019.00020

[52] Yuanyuan Yuan, Qi Pang, and Shuai Wang. 2023. Unveiling Hidden DNN Defects with Decision-Based Metamorphic Testing. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering* (Rochester, MI, USA) *(ASE '22)*. Association for Computing Machinery, New York, NY, USA, Article 113, 13 pages. https://doi.org/10.1145/3551349.3561157

[53] Dazhi Zhan, Wei Bai, Xin Liu, Yue Hu, Lei Zhang, Shize Guo, and Zhisong Pan. 2023. PSP-Mal: Evading Malware Detection via Prioritized Experience-based Reinforcement Learning with Shapley Prior. In *Annual Computer Security Applications Conference (ACSAC)* (Austin, TX, USA) *(ACSAC '23)*. Association for Computing Machinery, New York, NY, USA, 580–593. https://doi.org/10.1145/3627106.3627178

[54] Lan Zhang, Peng Liu, Yoon-Ho Choi, and Ping Chen. 2023. Semantics-Preserving Reinforcement Learning Attack Against Graph Neural Networks for Malware Detection. *IEEE Transactions on Dependable and Secure Computing* 20, 2 (March 2023), 1390–1402. https://doi.org/10.1109/TDSC.2022.3153844

[55] Fangtian Zhong, Xiuzhen Cheng, Dongxiao Yu, Bei Gong, Shuaiwen Song, and Jiguo Yu. 2024. MalFox: Camouflaged adversarial malware example generation based on conv-GANs against black-box detectors. *IEEE Trans. Comput.* 73, 4 (2024), 980–993. https://doi.org/10.1109/TC.2023.3236901

[56] Shuofei Zhu, Jianjun Shi, Limin Yang, Boqin Qin, Ziyi Zhang, Linhai Song, and Gang Wang. 2020. Measuring and Modeling the Label Dynamics of Online Anti-Malware Engines. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Boston, MA, USA, 2361–2378. https://www.usenix.org/conference/usenixsecurity20/presentation/zhu

Received 2026-03-26; accepted 2026-06-18