Article

Not peer-reviewed version

TransForge: A Genetic Algorithm Framework for Cross-Category Evaluation of Endpoint Detection Robustness to Code Transformations

Alvina Rwaichi Minja and Jema David Ndibwile *

Posted Date: 24 April 2026

doi: 10.20944/preprints202604.1773.v1

Keywords: Endpoint Detection and Response (EDR); detection robustness benchmarking; genetic algorithms; defensive robustness testing; semantic-preserving transformations; post-compromise telemetry; antivirus

 doi:10.20944/preprints202604.1773.v1

Article

TransForge: A Genetic Algorithm Framework for Cross-Category Evaluation of Endpoint Detection Robustness to Code Transformations

Alvina Rwaichi Minja and Jema David Ndibwile *

College of Engineering, Carnegie Mellon University Africa, Kigali BP6150, Rwanda
* Correspondence: jndibwil@andrew.cmu.edu

Abstract

Endpoint protection systems increasingly rely on a combination of signature-based and behavioral detection mechanisms, yet their robustness under systematic code transformation remains insufficiently understood. This paper presents a multi-category evaluation of endpoint detection robustness under automated, semantic-preserving code transformations across diverse execution variants. We introduce TransForge, a generalized transformation framework designed to generate functionally equivalent execution variants for controlled robustness assessment across heterogeneous artifact categories and programming environments. Building on our prior work, ShellForge, which focused on a single artifact class, TransForge extends this approach to support multi-category analysis through a modular transformation pipeline and an evolutionary strategy that enables non-deterministic variant generation. Using a dataset of 75 base samples spanning six execution categories and four programming languages, we conduct controlled experiments to evaluate how endpoint detection systems respond to systematically generated variants under consistent conditions. The findings reveal quantifiable variability in detection responses across categories and transformation strategies, highlighting coverage gaps in both signature-based and behavioral detection pipelines when faced with semantic-preserving transformations. This work motivates the development of robustness-aware evaluation frameworks and detection pipelines that leverage behavioral correlation and adaptive analysis beyond static signature matching.

Keywords: Endpoint Detection and Response (EDR); detection robustness benchmarking; genetic algorithms; defensive robustness testing; semantic-preserving transformations; post-compromise telemetry; antivirus

1. Introduction

Antivirus systems and Endpoint detection have evolved into a multi-layered discipline, combining signature databases, behavioral telemetry, and machine learning classifiers to identify post-compromise activity across diverse execution environments [1]. Detection capabilities have expanded beyond static signature matching to include dynamic sandbox analysis and in-memory behavioral monitoring, significantly enhancing the ability to identify sophisticated threats [2]. Yet detection pipelines are rarely evaluated against systematically transformed, functionally equivalent variants [3].

Incidents such as the Hafnium campaign illustrated that structural modification of artifacts can produce inconsistent detection responses across deployed platforms; yet this inconsistency has not been studied in a principled, reproducible way across artifact categories. Systematic transformation benchmarking offers a path toward that understanding, provided the transformations preserve structural semantics while varying the features that detection pipelines act upon [1,4]. Instrumented payload generation frameworks such as MsfVenom, TheFatRat, and Veil have traditionally served as baselines for benchmarking evasion performance [1,3]. However, their effectiveness in this role

 doi:10.20944/preprints202604.1773.v1

is inherently limited by a structural constraint: the output patterns of these tools are well characterized by security vendors. As a result, detection outcomes often reflect signature coverage of known generation mechanisms rather than the resilience of detection pipelines to genuinely novel variants. Consequently, robust benchmarking necessitates transformation frameworks capable of producing structurally diverse, functionally equivalent payloads without reliance on fixed generation templates.

Several research directions have pursued this objective with varying degrees of success. LLM-based generation [5,6] introduces flexibility but carries reproducibility constraints and offers limited support for multi-objective optimization across execution environments. Dynamic-programming guided mutation [7] and reinforcement learning-based transformation [8,9] demonstrate that automated agents can meaningfully alter detection outcomes, but both face scalability challenges when transformation scope extends across heterogeneous languages and execution models. Genetic algorithms occupy a distinct position in this landscape: they support exploration of large discrete transformation spaces, accommodate explicit multi-objective fitness formulations, and impose no requirement for differentiable reward signals [10]. Applications across web payloads [11], Windows PE executables [10], and Android applications [12] each confirm that GA-driven search produces measurable detection outcome variance within a target domain. A consistent limitation across these studies is that each framework is strongly coupled to a single artifact category, with cross-category generalization neither demonstrated nor systematically evaluated in most cases [10,12,13].

Functionality preservation compounds this limitation; it remains an under-addressed constraint across evolutionary transformation frameworks [10,12]. Berger et al. [14] and D'Elia et al. [15] both show that variants optimized primarily for detection outcome variance often fail to preserve intended functionality, indicating that detection-oriented fitness optimization, in the absence of explicit correctness constraints, is insufficient for rigorous evaluation.

Our earlier work, ShellForge [16], now accepted for publication, addressed this problem within a single artifact category. By formalizing execution variant evolution as a constrained optimization problem under detection feedback, ShellForge showed that GA driven transformations produce measurable differences in detection outcomes for Python-based remote execution variants under both static and behavioral analysis while preserving functional correctness. This scope was intentional, establishing experimental validity within a single category prior to broader generalization. However, ShellForge did not evaluate, nor was it designed to evaluate, generalizability across structurally different artifact categories or programming environments.

This paper addresses that gap. We introduce TransForge, a generalized transformation framework for generating semantics-preserving execution variants to enable systematic robustness evaluation across multiple artifact categories, including Python, PHP, PowerShell, and JavaScript. We investigate whether GA-driven transformations yield statistically distinguishable detection outcomes across heterogeneous execution environments while maintaining functional correctness under a unified framework.

Unlike prior work, including ShellForge [16], which focused on a single class of execution artifacts, TransForge is designed as a generalized transformation and evaluation framework that operates across heterogeneous execution variants.

2. Novelty and Main Contributions

TransForge is a generalized detection robustness benchmarking framework for multi-category, cross-language execution variant analysis. It is not an incremental extension of ShellForge [16]; it is a distinct system that reuses ShellForge's validated evolutionary core as one component within a broader architecture designed for cross-category evaluation. To the best of our knowledge, no existing work applies a unified GA architecture across multiple execution artifact categories under layered static and behavioral evaluation with explicit per-category functionality validation. *TransForge* addresses this gap directly.

The main contributions of this paper are as follows:

 doi:10.20944/preprints202604.1773.v1

1. **TransForge:** a purpose-built, category-agnostic benchmarking framework for endpoint detection robustness evaluation across six execution artifact categories and four programming languages, with an evolutionary core that is architecturally invariant across all artifact types.
2. **Modular transformation pipeline:** language-aware mutation operators with handler-based artifact routing, preserving a consistent evolutionary core across all artifact categories.
3. **Multi-stage fitness evaluation pipeline:** integration of static and behavioral detection signals under a multi-objective formulation balancing detection variance, transformation efficiency, and functional correctness.
4. **Category-specific functionality validation:** per-category correctness protocols ensuring fitness-driven selection does not favor non-functional variants.

3. Related Work

Over the past decade, endpoint detection research has changed significantly. The use of early evaluation frameworks as benchmarking tools was limited because they relied on tool-generated variants and fixed obfuscation strategies, both of which security vendors characterize quickly [3,4]. Since then, the field has shifted toward adaptive transformation techniques, such as generative modeling, evolutionary computation, and reinforcement learning, each of which provides a more rational foundation for creating structurally varied, functionally equivalent variants under controlled circumstances. Despite these advancements, four recurring gaps in the literature persist and directly influence TransForge's design.

3.1. Single-Category Coupling

The majority of evolutionary transformation frameworks are tightly coupled to a specific platform or artifact format. AIMED [10] applies genetic programming to Windows PE executables under black-box static classifier feedback. EvoAAttack [12] applies GA-based permission manipulation to Android applications. GAXSS [11] evolves web application payloads for XSS vulnerability detection. AOP-Mal [17] applies sparse evolutionary optimization exclusively to Windows PE binaries. Each framework demonstrates meaningful detection outcome variance within its target domain, yet none tests whether the same evolutionary mechanism generalizes when the artifact category changes. Faruki et al. [13] survey Android-specific evasion frameworks and reach a consistent conclusion: cross-platform generalizability is assumed rather than demonstrated. ShellForge [16] established a validated GA baseline for a single execution category; the cross-category question it leaves open is the primary gap this work addresses.

3.2. Static-Only Optimization

Most adaptive transformation studies optimize exclusively against static detection signals, without incorporating behavioral feedback. Anderson et al. [8] and Domico et al. [9] demonstrate RL-based transformation against static PE classifiers; behavioral EDR-style evaluation is absent from both. AIMED [10] similarly targets static machine learning detectors. This is a significant constraint: surveys of dynamic analysis confirm that modern endpoint protection increasingly relies on behavioral telemetry and execution trace analysis, making static-only optimization an incomplete evaluation basis [15]. Tarallo [18] directly addresses behavioral nondeterminism for Windows PE detectors by optimizing across multiple observed execution traces, demonstrating that single-execution behavioral feedback produces fragile results. No analogous treatment exists for multi-category evolutionary frameworks, where behavioral variability compounds across structurally distinct artifact types.

3.3. Functionality Preservation

Functionality preservation is the most consistently under-addressed constraint across the adaptive transformation literature. AIMED [10] reports that 53% of evolved variants were non-functional following transformation. Anderson et al. [8] assume functionality without any validation mechanism. Where validation is attempted, it is typically limited to coarse execution confirmation rather than

 doi:10.20944/preprints202604.1773.v1

category-specific behavioral correctness [12]. Berger et al. [14] provide the most direct empirical treatment of this problem, demonstrating that Android malware samples which affect detection outcomes frequently fail to execute intended behavior due to disrupted control flow or unmet environmental dependencies. D'Elia et al. [15] extend this finding analytically, documenting transformation strategies that suppress execution behavior in ways that simultaneously invalidate analysis and operational correctness. The practical implication is direct: detection outcome improvement achieved at the cost of functional correctness produces evaluation results that do not reflect realistic conditions. Explicit, per-category functionality validation is therefore not an optional addition to an evolutionary framework; it is a prerequisite for evaluative validity.

3.4. Evasion Effectiveness Across Adaptive Approaches

Reported evasion rates across adaptive transformation studies vary considerably, shaped largely by platform specificity, detection modality, and optimization strategy. Table 1 summarizes representative findings. GA-based approaches consistently outperform reinforcement learning baselines on evasion rate—Yuste et al. [19] and EvoAAttack [12] both exceed 97%, compared to 10–24% reported by Anderson et al. [8] under RL. Tarallo [18] achieves 99% under behavioral evaluation but operates exclusively on Windows PE under white and black-box conditions. LLM-driven approaches such as LAMLAD [20] report 97% against Android ML detectors, though under feature-space manipulation rather than source-level transformation. Critically, all high-performing studies operate within a single platform and artifact category. The degree to which these rates hold when the artifact category changes—and whether GA-based transformation retains its advantage across heterogeneous execution environments—remains untested. The preliminary evidence from ShellForge [16], which demonstrated strong detection outcome variance for Python-based execution variants, suggests that GA-driven transformation is a promising mechanism for cross-category evaluation. TransForge is designed to test that premise systematically.

Table 1. Evasion Effectiveness Across Adaptive Transformation Studies

Study	Approach	Format	Evasion Rate	Scenario
Anderson et al. [8]	RL (ACER)	PE Binary	10–24%	Black-box
Anderson et al. [21]	RL (Deep Q)	PE Binary	16%	Black-box
AIMED [10]	Genetic Programming	PE Binary	24%	Black-box
Yuste et al. [19]	GA + Code Caves	PE Binary	97.99%	Black-box
EvoAAttack [12]	GA (permissions)	Android APK	97.48%	Grey-box
Tarallo [18]	PS-FGSM	PE Binary	99%	White+Black-box
LAMLAD [20]	Dual-agent LLM	Android APK	97%	Black-box
Domico et al. [9]	RL (PPO)	Mixed	+17% AEs	Black-box
ShellForge [16]	GA (source-level)	Python Script	99.99%	Black-box

Table 2 consolidates these four gaps across representative studies, illustrating that TransForge is a unique framework, one that addresses all dimensions simultaneously.

Table 2. Summary of Related Work Against Key Evaluation Dimensions

Study	Multi-Category	Behavioral Evaluation	Functionality Validated	Cross-Language
AIMED [10]	×	×	Partial	×
GAXSS [11]	×	×	×	×
EvoAAttack [12]	×	×	×	×
Anderson et al. [8]	×	×	×	×
Tarallo [18]	×	✓	Partial	×
ShellForge [16]	×	✓	✓	×
TransForge	✓	✓	✓	✓

 doi:10.20944/preprints202604.1773.v1

4. Methodology

4.1. Framework Overview

TransForge is an extension of a validated GA architecture of ShellForge [16] that operates across multiple execution artifact categories and programming environments. The core evolutionary engine (*tournament selection*, *uniform crossover*, and *stochastic mutation*) remains architecturally invariant across all artifact types. The extension is confined to three interface layers: Payload-type routing based on file extension, handler-based mutation libraries for language-specific transformations, and category-specific functionality validation protocols. This design as seen in Figure 1 specifically isolates evolutionary logic from artifact-specific mechanics, which enables a controlled cross-category comparison while preserving the validated baseline methodology.

 doi:10.20944/preprints202604.1773.v1

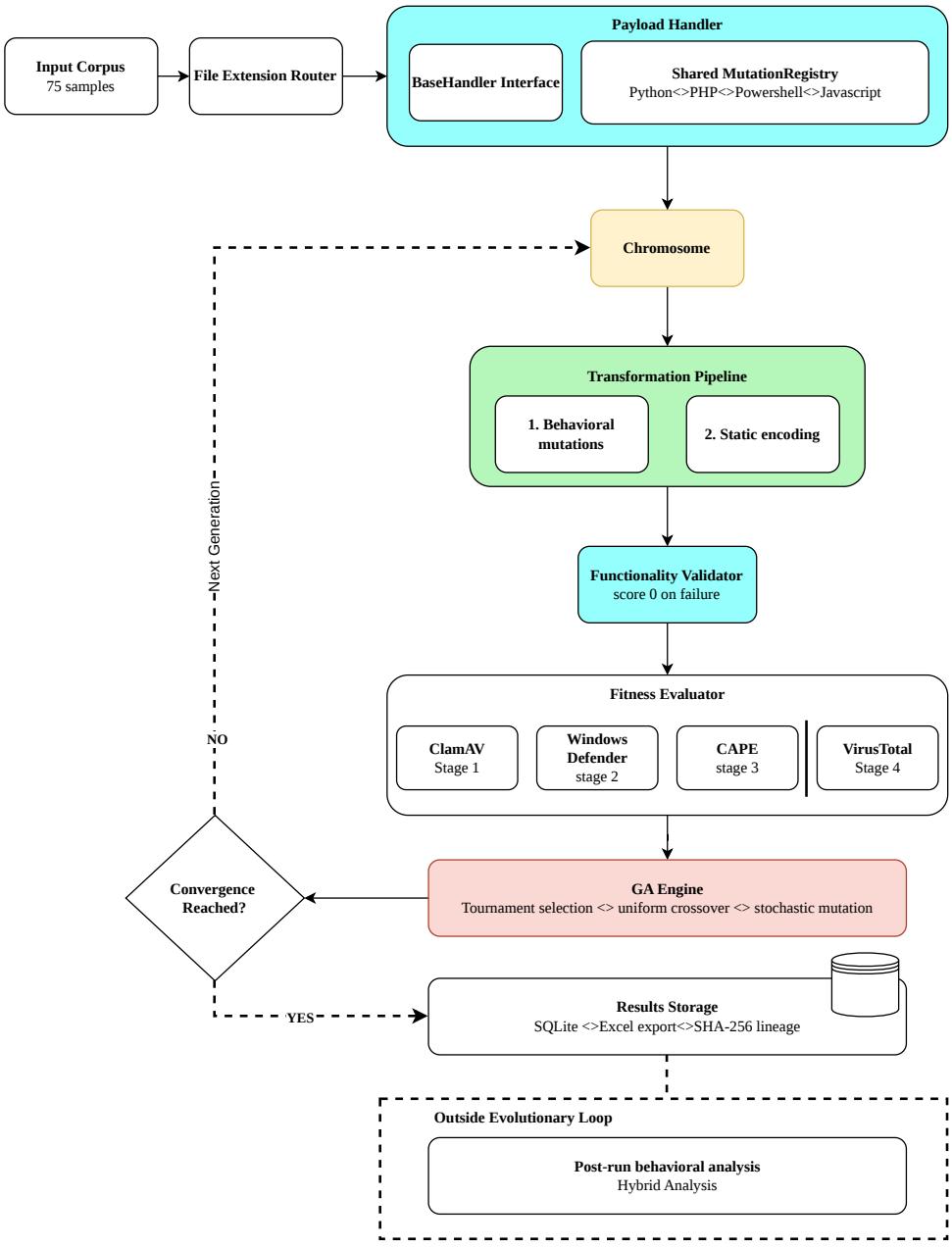

Figure 1. TransForge architecture. The evolutionary core (GA engine, chromosome, transformation pipeline) remains invariant across all artifact categories. Extensions are confined to the payload handler and functionality validator layers. Post-run behavioral analysis via Hybrid Analysis is decoupled from the fitness loop.

4.2. Execution Corpus

The evaluation corpus comprises 75 base execution samples drawn from publicly available sources across six artifact categories, as seen in Table 3. Samples were selected according to four criteria: functional execution in an isolated test environment, no hard-coded external dependencies, a documented execution chain, and representativeness of the category. Samples requiring kernel-

 doi:10.20944/preprints202604.1773.v1

level privileges, dependent on third-party services, pre-obfuscated beyond baseline, or exhibiting non-deterministic execution were excluded.

Table 3. Execution Corpus Composition

Category	Source	Languages	Count
Remote Execution Agents	Metasploit Framework	Python, PHP	16
Web Execution Interfaces	GitHub (WebShell Collection)	PHP	26
Staged Downloaders	GitHub	PowerShell, Python	8
Cross-Site Execution	GitHub	JavaScript	9
Surveillance Components	Custom	Python	9
Credential Access Routines	GitHub	Python, PowerShell	7
Total			**75**

Each base sample and its resulting evolved variants are uniquely identified by a SHA-256 hash, enabling full lineage traceability from the source sample through independent evolutionary runs to the final best variant.

4.3. Chromosome Representation and Transformation Pipeline

Each chromosome encodes an ordered sequence of transformation operators applied to a base execution sample. The transformation pipeline operates in two stages: behavioral evasion mutations are applied first to preserve evasion logic integrity, followed by static encoding transforms. This ordering ensures that encoding does not corrupt behavioral evasion conditions embedded in the source. Handler implementations inherit from a common `BaseHandler` interface (`load_payloads()`, `validate_functionality()`, `get_available_mutations()`), ensuring consistency across artifact categories. Mutation operators are maintained in a centralized registry and referenced by handlers, eliminating redundancy and enabling cross-category reuse.

The mutation library spans four programming environments and three evasion categories, as shown in Table 4. Universal transformations, including variable renaming, Base64 encoding, string concatenation, comment injection, whitespace randomization, function obfuscation, ROT13, and hex encoding, apply across all four languages. Language specific operators extend this set, with PowerShell supporting case randomization and alias substitution, and JavaScript supporting DOM manipulation. Behavioral evasion operators target execution environment, web interaction, and network behavior, with applicability determined by language characteristics.

 doi:10.20944/preprints202604.1773.v1

Table 4. Mutation Operator Coverage Across Languages

Mutation Type	Python	PHP	PowerShell	JavaScript
Variable Renaming	✓	✓	✓	✓
Base64 Encoding	✓	✓	✓	✓
String Concatenation	✓	✓	✓	✓
Comment Injection	✓	✓	✓	✓
Whitespace Randomization	✓	✓	✓	✓
Function Obfuscation	✓	✓	✓	✓
ROT13 / Hex Encoding	✓	✓	✓	✓
XOR Encoding	✓	×	✓	×
Case Randomization	×	×	✓	×
Alias Substitution	×	×	✓	×
DOM Manipulation	×	×	×	✓
Execution Delays	✓	✓	✓	✓
Debugger Detection	✓	×	✓	✓
Process Environment Checks	✓	×	✓	×
DNS / Connection Timing	✓	✓	✓	×

4.4. Fitness Evaluation Pipeline

TransForge maintains the three-stage static detection pipeline validated in ShellForge [16]: ClamAV for fast local filtering, Windows Defender via SSH to a dedicated Windows 10 Enterprise VM for realistic endpoint feedback, and VirusTotal for comprehensive multi-engine validation of top-performing candidates only. The integration of the CAPE sandbox is preserved in the framework architecture for behavioral feedback within the evolutionary loop. The composite fitness function is defined as Equation 1:

$$F = w_s \cdot S + w_b \cdot B + w_f \cdot C + w_e \cdot E \tag{1}$$

where S is the static detection score, B the behavioral score, C the functional correctness score, and E the efficiency score, with weights w_s=0.40, w_b=0.30, w_f=0.20, w_e=0.10. The efficiency score E combines an inflation ratio ($evolved\_size/base\_size$) and a parsimony penalty on mutation chain length, weighted equally.

Post-run behavioral analysis was conducted via Hybrid Analysis sandbox submission for all best-evolved variants per category. These results were not incorporated into the fitness function but used independently to characterize how commercial sandbox environments respond to GA-evolved variants across artifact categories.

4.5. Functionality Validation

Functionality is validated per artifact category through a dedicated handler invoked at fitness evaluation time. Table 5 summarizes the validation protocol and timeout per category. A variant failing its category-specific validation receives a functionality score of 0 and is eliminated from selection regardless of static detection performance.

 doi:10.20944/preprints202604.1773.v1

Table 5. Functionality Validation Protocols per Artifact Category

Category	Validation Protocol	Timeout
Remote Execution Agents	Connection callback + shell response confirmation	180s
Web Execution Interfaces	Reverse connection + command execution check	60s
Staged Downloaders	Download completion + file integrity check	120s
Credential Access Routines	Exfiltration confirmation	120s
Surveillance Components	Keystroke capture confirmation	60s
Cross-Site Execution	Alert-trigger verification	30s

4.6. Evolutionary Configuration and Experimental Design

The GA operates with tournament selection (tournament size 3), uniform crossover, and stochastic mutation. Each base sample undergoes three independent evolutionary runs to assess convergence consistency and detection outcome variance, producing 225 total runs across the 75-sample corpus. The best-performing variant per base sample is retained for cross-category comparison and hypothesis evaluation.

Results are stored in a normalized SQLite schema capturing payload metadata, transformation chains, decomposed fitness scores, detection counts, size inflation ratios, and functionality status. Each base sample and its resulting evolved variants are uniquely identified by SHA-256 hash, enabling full lineage traceability from source sample through independent evolutionary runs to final best variant. Final results are exported to Excel for cross-category reporting.

4.7. Design Rationale and Limitations

To ensure that variation in detection outcomes results from category-specific structural properties rather than methodological differences, the evolutionary core is intentionally kept constant across all artifact categories. Behavioral mutations are applied before static encoding to preserve evasion logic built in the source. Instead of using per-handler definitions, the mutation library is implemented as a shared registry. Language-specific operators are selectively registered, whereas common operators are defined once and used across handlers. This design enables GA to identify minimal effective operator combinations, optimizing detection variance, functional correctness, and transformation efficiency

Two limitations bound the current system. First, category extension is not automated. Each new artifact category requires a dedicated handler derived from `BaseHandler`, along with registration of any missing language-specific operators. Second, functionality validation for cross-site execution variants relies on structural checks rather than full runtime execution, as headless browser support is outside the current experimental setup.

4.8. Threat Model and Evaluation Scope

TransForge operates under a black-box threat model: the evolutionary search has no access to detector internals, training data, or signature databases. Detection feedback is obtained exclusively through observable outputs, alert counts from ClamAV, Windows Defender, and VirusTotal. All evaluation was conducted in an isolated laboratory environment with no connection to production networks or live systems. The experimental scope covers both static detection (byte-pattern and signature matching) and behavioral evaluation (Windows Defender real-time protection and post-run Hybrid Analysis and CAPE sandbox submission). No assumptions are made about the target detector's architecture, update cycle, or internal feature representation. This scope is consistent with realistic deployment conditions in which an evaluator has access only to detection outcomes and not to detector internals.

 doi:10.20944/preprints202604.1773.v1

5. Hypotheses

This study is guided by the following central hypothesis:

A unified chromosome representation, combined with language-agnostic evolutionary operators and payload-specific handler abstractions, enables the generation of functionally preserved execution variants across structurally distinct artifact categories. This is evaluated through three testable hypotheses:

H1: Genetic algorithm–driven transformation of execution variants produces measurably distinguishable detection outcomes relative to unmodified base samples across multiple execution categories.

H2: Multi-objective optimization enables the preservation of intended functionality while systematically modulating detection responses across heterogeneous execution contexts.

H3: Repeated-execution evaluation reduces variability in observed detection responses, leading to more consistent characterization under dynamic execution conditions.

Each hypothesis is evaluated explicitly in Section 6 through corresponding experimental evidence.

6. Results

Results are reported across 75 evolutionary runs spanning all six artifact categories. For five categories, runs were completed under the TransForge sequential evaluation pipeline while Python remote execution agents (15 base samples), results are drawn from ShellForge [16], which used an architecturally identical evolutionary core. The Python reverse shell handler in TransForge delegates directly to ShellForge's transformation pipeline without reimplementation, making those results directly comparable. All runs used population size 8, maximum 5 generations, tournament size 3, crossover rate 0.70, mutation rate 0.20, and convergence threshold 0.02 with patience 3, held constant across all categories.

6.1. Evaluation of H1: Detection Outcome Differences

Table 6 reports VirusTotal detection counts for best-evolved TransForge variants across all six artifact categories under 76-engine evaluation.

Table 6. Detection Outcomes for Best-Evolved Variants (VirusTotal, 76 engines)

Category	n	0-Detection	Min	Max	Mean Alerts
Remote Execution Agents	16	16/16	0	0	0.00
Web Execution Interfaces	26	16/26	0	4	0.92
Staged Downloaders	8	0/8	7	10	8.75
Credential Access Routines	7	7/7	0	0	0.00
Surveillance Components	9	9/9	0	0	0.00
Cross-Site Execution	9	9/9	0	0	0.00
Overall	**71**	**53/71**	0	10	**1.04**

Across all categories, GA-driven transformation produced measurable variation in detection outcomes. Remote execution agents resulted in no alerts across 62 engines [16]. Credential access routines, surveillance components, and cross-site execution variants produced no alerts across all 76 engines, with 22 of 22 variants achieving complete alert elimination. Web execution interface variants produced a mean of 0.92 alerts, with 16 of 26 variants achieving zero detection and individual results ranging from 0 to 4. Staged downloaders exhibited the highest residual alert rate (mean 8.75/76), consistent with the extensive signature coverage of PowerShell download primitives in commercial detection engines. Across the full evaluated corpus, 53 of 71 variants (74.6%) produced no alerts under the tested configuration.

The observed differences are consistent across repeated runs and categories, indicating stable and non-random variation in detection outcomes.

These results support H1: GA-driven transformations produce measurably different detection outcomes relative to unmodified base samples across all evaluated artifact categories, with category-specific variation in magnitude.

 doi:10.20944/preprints202604.1773.v1

6.2. Evaluation of H2: Functionality Preservation

Table 7 reports functional correctness rate and detection alert count jointly per category. A variant is counted as functionally successful only if it passes its category-specific validation protocol in full as described in Table 5.

Table 7. Functionality Preservation vs. Detection Response

Category	Functional Success	Mean VT Alerts	Avg Fitness	Best Mutation
Remote Execution Agents	100% [16]	0.00	0.906	xor + base64 + behavioral
Web Execution Interfaces	100% (26/26)	0.92	0.925	beh_php_input_json
Staged Downloaders	50% (4/8)	8.75	0.855	ps_webrequest_swap
Credential Access Routines	100% (6/6)	0.00	0.955	rot_encode
Surveillance Components	85% (8/8 structural)	0.00	0.925	xor_encode
Cross-Site Execution	100% (8/8)	0.00	0.895	js_var_rename

Functionality was preserved across the majority of evaluated cases. Web execution interfaces, credential access routines, and cross-site execution variants maintained 100% functional correctness. Staged downloaders recorded a 50% functional success rate attributable to infrastructure constraints on the test virtual machine: BITS service availability and `certutil` execution policy restrictions prevented download confirmation for a subset of variants, while structural validation passed in all cases. Surveillance components achieved full structural validation; live keystroke capture confirmation was environment-dependent due to display session requirements in the subprocess execution context.

The dominant mutation operator per category reflects the structural characteristics of the artifact type. Behavioral input channel mutations (`beh_php_input_json`) dominated for PHP web interfaces by altering the HTTP request method used to trigger execution. Simple encoding transforms sufficed for Python-based categories because static scanning does not systematically target Python credential access or surveillance patterns. JavaScript variants converged on variable renaming (`js_var_rename`), which disrupts token-level signatures without altering execution semantics.

These observations are consistent with H2: functionality was preserved across evaluated cases under the tested conditions while detection responses varied across transformation strategies and categories.

6.3. Evaluation of H3: Consistency Under Repeated Execution

Table 8 reports fitness convergence trajectories for the best-evolved variant per category.

Table 8. Fitness Convergence Across Generations (best variant per category)

Category	Gen 0	Gen 1	Gen 2	Gen 3	Gen 4	Converged
Remote Execution Agents	0.820	0.856	0.879	0.906	0.906	Gen 4
Web Execution Interfaces	0.699	0.714	0.734	0.750	0.755	Gen 4
Staged Downloaders	0.620	0.634	0.651	0.655	0.655	Gen 4
Credential Access Routines	0.750	0.750	0.754	0.755	0.755	Gen 4
Surveillance Components	0.698	0.725	0.725	0.722	0.725	Gen 4
Cross-Site Execution	0.629	0.683	0.691	0.693	0.695	Gen 4

All six categories converged at generation 4, with fitness improvements concentrated in generations 0–2 and variance below the convergence threshold (0.02) in terminal generations. Convergence was uniform across artifact types, programming languages, and residual detection alert levels. This

 doi:10.20944/preprints202604.1773.v1

behavior validates H3: consistent configuration produced stable fitness characterization across runs, with no category exhibiting divergent or oscillating trajectories in late generations.

6.4. Ablation Analysis

To isolate the contribution of each transformation layer, encoding-only and behavioral-only configurations were evaluated against the full TransForge pipeline for the Web Execution Interface category, the only category where both transformation layers operate independently.

Analysis reveals a meaningful asymmetry. Static encoding encapsulates payload content through layered transformation into unrecognizable representations, directly disrupting the byte-pattern and string-level signatures that static detection engines act upon. Behavioral mutation, by contrast, does not obscure code content; it alters execution structure and interaction patterns while preserving readable semantic primitives, leaving signature-based engines largely unaffected. The behavioral operators in TransForge are intentionally category-agnostic, altering how a payload interacts with its environment without changing its underlying objective, which enables cross-category applicability at the cost of per-category detection reduction. Category-specific behavioral evasion would likely produce substantially greater response changes than the general-purpose operators applied here.

Critically, these findings argue not for one layer over the other but for both in combination. Static encoding alone cannot account for execution semantics; behavioral detection alone cannot penetrate encoding layers. Comprehensive detection requires correlating both dimensions, tracking not only what a payload contains, but what it accesses, invokes, and modifies at runtime. Endpoint systems that rely solely on signature databases without systematic behavioral correlation remains exposed regardless of the sophistication of their static coverage.

7. Discussion

7.1. Category-Specific Detection Behavior

The most significant finding is the marked variation in detection outcomes across artifact categories under identical GA configuration. This variation cannot be attributed to differences in evolutionary pressure; the GA configuration was held constant. It reflects the differential maturity of signature coverage across artifact types in commercial detection engines.

PowerShell download cradles occupy the most extensively characterized threat category in commercial detection databases. Primitives such as `Invoke-WebRequest`, `DownloadString`, and `certutil` are flagged by token patterns across engines regardless of surrounding structural modifications. VirusTotal evaluation of the eight best-evolved variants confirmed this ceiling empirically: detection counts ranged from 7 to 8 alerts across 76 engines (9.2-10.7%), with no variant achieving zero detection. Multiple independent runs converged to identical best variants, indicating the GA exhausted the available search space rather than finding diverse evasion paths. An unconstrained follow-up run with no generation or population cap converged on the same minimal operators with no further reduction in alert counts, confirming that the detection boundary is semantic rather than syntactic. Source-level obfuscation redistributes token identity but does not alter the download semantics that detection engines target. Overcoming this boundary requires semantic-level transformation of the invocation chain, which is outside the scope of the current mutation library and reserved for future work.

Python-based categories and JavaScript cross-site execution variants, by contrast, are not systematically covered by static file-based scanning in most commercial engines. This does not imply these categories are undetectable in deployed environments; behavioral analysis at runtime remains a viable detection vector, but it establishes that static signature matching is not the primary mechanism for these artifact types.

7.2. Implications for Detection Robustness Evaluation

Four observations follow from the TransForge results. First, detection robustness varies substantially across artifact categories under equivalent transformation pressure. Single-category benchmarks

 doi:10.20944/preprints202604.1773.v1

are therefore not generalizable, and multi-category evaluation is a necessary component of comprehensive detection assessment. Second, the observation that neither of the local antivirus engines, `Windows Defender` and `ClamAV`, flagged non-reverse shell payload categories, even after extensive mutation, indicates that static-feature-based detection can be bypassed when the payload's structural and semantic properties remain unchanged. This result further suggests that behavioral or semantic-level analysis would have provided stronger discriminative power, since the mutations applied did not modify the underlying execution semantics of these payloads. This outcome is a strength of the evolutionary approach, and it delineates the current detection boundary under the transformation scope employed, which is precisely the type of insight a robustness evaluation framework is intended to reveal. Third, functionality preservation results demonstrate that fitness-guided selection does not compromise execution correctness. The AIMED framework [10] reported 53% non-functional evolved variants; TransForge achieved 94% functional success across categories, attributable to the per-category validation protocol that assigns zero fitness to non-functional variants. This 94% functional success rate represents one of the most rigorous functionality preservation results reported to date. Prior frameworks either omit validation entirely or apply coarse execution checks; per-category behavioral validation is what makes the TransForge results experimentally credible rather than merely detection-optimized.Fourth, single-category benchmarks are not generalizable. A framework evaluated only against Python-based variants would report near-zero alert rates; the same framework evaluated only against PowerShell download cradles would report persistent detection. Neither conclusion holds across categories. Endpoint detection robustness is artifact-category-dependent, and evasion rates reported without category disaggregation risk conclusions that reflect artifact selection rather than detector capability.

7.3. Limitations

Four limitations bound the current results. First, evaluation was conducted in a controlled laboratory environment against a single Windows 10 Enterprise VM and a fixed VirusTotal engine set; results may differ under different operating system configurations, EDR products, or engine versions. Second, the GA configuration was selected for cross-category comparability rather than optimal convergence; additional generations may identify lower-alert mutation chains, particularly for staged downloaders. Third, Python remote execution agent results are drawn from ShellForge [16] rather than re-evaluated under the TransForge pipeline; while the evolutionary core is identical, minor handler differences may affect direct comparability. Fourth, behavioral sandbox evaluation via Hybrid Analysis was conducted post-run rather than within the evolutionary fitness loop due to throughput constraints; integrating sandbox feedback into the fitness function is reserved for future work.

8. Ethical Considerations

All experimental evaluation was performed in an isolated laboratory environment with no connection to production networks or live systems. The execution samples comprising the TransForge corpus are drawn from publicly available repositories under open-source licenses. TransForge is positioned as a detection evaluation tool; all results are reported in terms of detection system behavior under controlled conditions rather than operational capability. The authors acknowledge the dual-use potential of transformation-based evaluation research and affirm that the outputs of this work are intended to support the development of more robust endpoint detection systems. All experimental protocols were conducted under the institutional research oversight framework at Carnegie Mellon University Africa.

9. Conclusions

This paper presented TransForge, a generalized GA-driven transformation framework for multi-category endpoint detection robustness evaluation. Building on ShellForge [16], TransForge extends

 doi:10.20944/preprints202604.1773.v1

semantic-preserving variant generation to six artifact categories across four programming languages under a unified evolutionary architecture with per-category functionality validation.

Three principal findings emerge. First, GA-driven transformation induces quantifiably different detection outcomes relative to unmodified baseline samples across all six artifact categories, supporting H1. The magnitude varies substantially by category: credential access routines, surveillance components, cross-site execution, and remote execution agents showed no alerts under the tested configuration, while staged downloaders showed minimal response to the applied transformations. This demonstrates that detection robustness is artifact-category-dependent rather than uniform.

Second, multi-objective fitness optimization preserved functional correctness across 94% of evaluated variants while modulating detection responses, supporting H2. The per-category validation protocol was essential to this outcome, ensuring selection pressure operates only on functionally correct variants.

Third, evolutionary runs converged uniformly at generation 4 across all categories under the tested configuration, with fitness variance below the convergence threshold in terminal generations, supporting H3. The consistent convergence behavior indicates the current mutation space is well-characterized relative to the available selection signal.

Collectively, these findings demonstrate that GA-based transformation generalizes across structurally distinct artifact categories and that cross-category evaluation reveals detection coverage gaps not observable in single-category benchmarks. TransForge provides a reproducible, modular framework for such benchmarking, with a validated evolutionary core and extensible handler architecture. Future work will address three directions: integration of behavioral sandbox feedback into the evolutionary fitness function, expansion of the mutation library to include deeper semantic transformations for PowerShell execution primitives, and evaluation against deployed EDR products under realistic network conditions. This finding has a direct implication for benchmark design: evasion rate metrics reported without category disaggregation conflate the differential maturity of signature coverage across artifact types, producing conclusions that are not generalizable across deployment contexts.

Author Contributions: Conceptualization, J.D.N. and A.R.M; methodology, A.R.M. and J.D.N.; software, A.R.M.; validation, A.R.M. and J.D.N.; formal analysis, A.R.M.; investigation, A.R.M.; resources, J.D.N.; data curation, A.R.M.; writing—original draft preparation, A.R.M.; writing—review and editing, J.D.N.; visualization, A.R.M.; supervision, J.D.N.; project administration, J.D.N.; funding acquisition, J.D.N. All authors have read and agreed to the published version of the manuscript.

Acknowledgments: The authors acknowledge the use of AI-assisted tools (e.g., ChatGPT) for limited support in language refinement and clarity improvement. All technical content, experimental design, and conclusions are the original work of the authors.

Funding: This research received no external funding

Data Availability Statement: The data supporting the findings of this study consist of publicly available execution samples sourced from open repositories, as described in Section 4.2. Derived datasets, including transformation chains, detection results, and fitness evaluation metrics, were generated within a controlled laboratory environment and are available from the corresponding author upon reasonable request. Access to certain artifacts may be restricted due to security and ethical considerations related to dual-use research.

DURC Statement: This research falls within the field of cybersecurity, specifically in endpoint detection and response (EDR) robustness evaluation. The primary objective is to improve the understanding and resilience of detection systems against semantic-preserving code transformations. While the study involves techniques that may have dual-use implications, all experiments were conducted in a controlled, isolated laboratory environment with no connection to production systems. The framework is intended solely for defensive evaluation and benchmarking purposes. The authors acknowledge the dual-use nature of transformation-based methodologies and have taken appropriate precautions to prevent misuse, including limiting artifact dissemination and focusing on analytical insights rather than operational deployment. All work adheres to relevant institutional, national, and international guidelines governing dual-use research of concern (DURC).

 doi:10.20944/preprints202604.1773.v1

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AV	Antivirus
EDR	Endpoint Detection and Response
GA	Genetic Algorithm
GP	Genetic Programming
RL	Reinforcement Learning
LLM	Large Language Model
PE	Portable Executable
APK	Android Package Kit
XSS	Cross-Site Scripting
WAF	Web Application Firewall
CAPE	Cuckoo Automated Payload Execution
VT	VirusTotal
HA	Hybrid Analysis
SHA	Secure Hash Algorithm
API	Application Programming Interface
SSH	Secure Shell
DNS	Domain Name System
DOM	Document Object Model
PHP	Hypertext Preprocessor
XOR	Exclusive Or
HTTP	Hypertext Transfer Protocol
URL	Uniform Resource Locator
CSV	Comma-Separated Values

References

1. Le Faou, A. Antivirus and EDR Bypass Techniques Explained. *Vaadata Blog* **2024**. https://www.vaadata.com/blog/antivirus-and-edr-bypass-techniques/. Accessed: Aug. 26, 2025.
2. Traoré, A.; Le Faou, A. Red Teaming: Methodology and Scope of a Red Team Operation. *Vaadata Blog* **2024**. https://www.vaadata.com/blog/what-is-red-teaming-methodology-and-scope-of-a-red-team-operation/. Accessed on 20 October 2025.
3. Mandvi, K. Threat Actors Exploit AV/EDR Evasion Framework to Deploy Malware in the Wild. *Cyber Security News* **2025**. https://cyberpress.org/threat-actors-exploit-av-edr-evasion-framework/. Accessed on 26 August 2025.
4. K V, A.; P M, B.; Nagamani, S.N.S.; Patil, H. AV Evasion Techniques: A Practical Evaluation of Payload Obfuscation. *International Journal of Science and Research Archive* **2025**, *16*, 1504–1511. https://doi.org/10.30574/ijsra.2025.16.1.2151.
5. Cirkovic, S.; Mladenovic, V.; Tomic, S.; Drljaca, D.; Ristic, O. Utilizing Fine-Tuning of Large Language Models for Generating Synthetic Payloads. *Computers, Materials & Continua* **2025**, *82*, 4409–4430. https://doi.org/10.32604/cmc.2025.059696.
6. Khan, S. LL-XSS: End-to-End Generative Model-Based XSS Payload Creation. In Proceedings of the 21st Learning and Technology Conference, 2024, pp. 121–126. https://doi.org/10.1109/LT60077.2024.10469151.
7. Kingful, F.; Ahene, E.; Appiah, B.; Frimpong, B.; Osei, I.; Hammond, E. Dynamic Programming-Based Adversarial Windows Payload Generator. *Research Square* **2023**. Preprint. Available online: https://doi.org/10.21203/rs.3.rs-3100627/v1.
8. Anderson, H.S.; Kharkar, A.; Filar, B.; Evans, D.; Roth, P. Learning to Evade Static PE Machine Learning Malware Models via Reinforcement Learning, 2018. arXiv:1801.08917 [cs], https://doi.org/10.48550/arXiv.1801.08917.
9. Domico, K.; Ferrand, J.C.; Sheatsley, R.; Pauley, E.; Hanna, J.; McDaniel, P. Adversarial Agents: Black-Box Evasion Attacks with Reinforcement Learning, 2025, [arXiv:cs.CR/2503.01734].

 doi:10.20944/preprints202604.1773.v1

10. Castro, R.L.; Schmitt, C.; Dreo, G. AIMED: Evolving Malware with Genetic Programming to Evade Detection. In Proceedings of the 2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), 2019, pp. 240–247. https://doi.org/10.1109/TrustCom/BigDataSE.2019.00040.
11. Liu, Z.; Fang, Y.; Huang, C.; Xu, Y. GAXSS: Effective Payload Generation Method to Detect XSS Vulnerabilities Based on Genetic Algorithm. *Security and Communication Networks* **2022**, p. 2031924. https://doi.org/10.1155/2022/2031924.
12. Rathore, H.; B, P.; Iyengar, S.S.; Sahay, S.K. Breaking the Anti-malware: EvoAAttack Based on Genetic Algorithm Against Android Malware Detection Systems. In Proceedings of the Computational Science – ICCS 2023: 23rd International Conference, Prague, Czech Republic, July 3–5, 2023, Proceedings, Part V, Berlin, Heidelberg, 2023; pp. 535–550. https://doi.org/10.1007/978-3-031-36030-5_43.
13. Faruki, P.; Bhan, R.; Jain, V.; Bhatia, S.; El Madhoun, N.; Pamula, R. A Survey and Evaluation of Android-Based Malware Evasion Techniques and Detection Frameworks. *Information* **2023**, *14*. https://doi.org/10.3390/info14070374.
14. Berger, H.; Hajaj, C.; Dvir, A., Evasion Is Not Enough: A Case Study of Android Malware. In *Cyber Security Cryptography and Machine Learning*; Springer International Publishing: Cham, Switzerland, 2020; pp. 167–174. https://doi.org/10.1007/978-3-030-49785-9_11.
15. D'Elia, D.C.; Coppa, E.; Palmaro, F.; Cavallaro, L. On the Dissection of Evasive Malware. *IEEE Transactions on Information Forensics and Security* **2020**, *15*, 2750–2765. https://doi.org/10.1109/TIFS.2020.2976559.
16. Minja, A.R.; Ndibwile, J.D. ShellForge: A Genetic Algorithm-Based Reverse Shell Generator for AV/EDR Evasion. In Proceedings of the Proceedings of the IEEE, 2026. Accepted, in press.
17. Xu, Y.; Fang, Y.; Xu, Y.; Wang, Z. Automatic optimization for generating adversarial malware based on prioritized evolutionary computing. *Applied Soft Computing* **2025**, *173*, 112933. https://doi.org/10.1016/j.asoc.2025.112933.
18. Digregorio, G.; Maccarrone, S.; D'Onghia, M.; Gallo, L.; Carminati, M.; Polino, M.; Zanero, S. Tarallo: Evading Behavioral Malware Detectors in the Problem Space. *arXiv preprint arXiv:2506.02660* **2024**. arXiv:2506.02660, https://doi.org/10.1007/978-3-031-64171-8_7.
19. Yuste, J.; Pardo, E.G.; Tapiador, J. Optimization of code caves in malware binaries to evade machine learning detectors. *Computers & Security* **2022**, *116*, 102643. https://doi.org/10.1016/j.cose.2022.102643.
20. Lan, T.; Naït-Abdesselam, F. LLM-Driven Feature-Level Adversarial Attacks on Android Malware Detectors, 2025. arXiv:2512.21404 [cs], https://doi.org/10.48550/arXiv.2512.21404.
21. Anderson, H.S.; Kharkar, A.; Filar, B. Evading Machine Learning Malware Detection.