Spectral Complexity Screening for Binary Security Analysis: A Random Matrix Theory Approach

Stuart Thomas

Independent Security Research — Whitby, North Yorkshire, United Kingdom

Plain Language Summary

Finding security vulnerabilities in compiled software is slow, expensive, and demands rare expertise. TriageForge applies mathematics from quantum physics (random matrix theory) and computer science (satisfiability phase transitions) to automate the first stages of that search. The core insight is that most software, viewed as a graph of function calls, has a predictable statistical structure; anomalous or complex code breaks that structure in measurable ways. Applied to 335 macOS system binaries, the pipeline reduced the candidate space by 96.4%, correctly classifying all anomalous candidates through subsequent automated and manual analysis.

Abstract

Automated vulnerability triage of compiled binaries presents a combinatorial challenge that scales poorly with binary size and complexity. We present TriageForge, a four-stage pipeline applying random matrix theory (RMT), Boolean satisfiability phase-transition analysis, cyclomatic complexity gating, and symbolic execution to identify anomalous code structure in macOS ARM64 binaries. The spectral screen (C2) models a binary's call graph as a weighted adjacency matrix and z-scores three statistics — largest eigenvalue (λ_{max}), graph energy, and eigenvalue entropy — against a configuration-model null calibrated to the observed degree sequence. Functions passing the screen are prioritised by a SAT backbone proximity score (C1); a cyclomatic complexity gate (C3) applies five structural dataflow templates; high-priority candidates receive full symbolic taint analysis (C6) via angr. Applied to 335 macOS 26 PrivateFrameworks binaries, the pipeline yielded a 3.6% anomaly detection rate with zero exploitable buffer overflows confirmed. We characterise three false-positive sources and discuss theoretical limitations.

Keywords: binary analysis, random matrix theory, SAT phase transitions, symbolic execution, vulnerability triage, call graph spectral analysis

1. Introduction

Security researchers face an adversarial triage problem: given a large corpus of compiled binaries, identify the small fraction containing exploitable vulnerabilities and, within those, the precise functions warranting costly symbolic execution or manual reverse engineering. The state of practice relies on human expertise developed over years of experience — expertise that does not scale. macOS 26 ships with over 3,000 Mach-O binaries in framework and private framework directories, each containing several hundred functions on average.

TriageForge is built around the observation that *call graph spectral properties discriminate between typical and atypical code structure in a statistically principled way*. The approach draws on three bodies of theory: random matrix theory [1,2], SAT phase-transition analysis [6,10], and symbolic execution [13]. Together, these provide a fast, theory-grounded screening mechanism that reduces the candidate space before expensive analysis begins.

The paper is organised as follows. Section 2 surveys the relevant mathematical background. Section 3 describes the TriageForge pipeline. Section 4 provides theoretical analysis of the screening decisions. Section 5 presents an empirical evaluation on 335 macOS PrivateFrameworks binaries. Sections 6 and 7 discuss limitations and conclude.

2. Background

2.1 Random Matrix Theory

Random matrix theory, initiated by Wigner in the study of nuclear energy levels [1,2], characterises eigenvalue spectra of large random matrices. The central result is the **Wigner semicircle law**: for an n×n symmetric matrix with i.i.d. entries of mean zero and variance σ^2, the empirical spectral distribution converges as $n \to \infty$ to

```
rho(x) = (2 / pi·R²) · sqrt(R² - x²), |x| <= R = 2·sigma·sqrt(n)
```

The **Tracy–Widom distribution** [3] characterises the limiting distribution of λ_{max}: the normalised quantity $n^{2/3}(\lambda_{max}/(\sigma \cdot \sqrt{n}) - 2)$ converges to the Tracy–Widom law of order 1 (GOE). **Graph energy**, defined by Gutman [9] as $E(G) = \Sigma|\lambda_i|$, captures total spectral mass. **Eigenvalue entropy** $H = -\Sigma\ p_i \log p_i$, where $p_i = |\lambda_i|/\Sigma|\lambda_j|$, measures spectral dispersion.

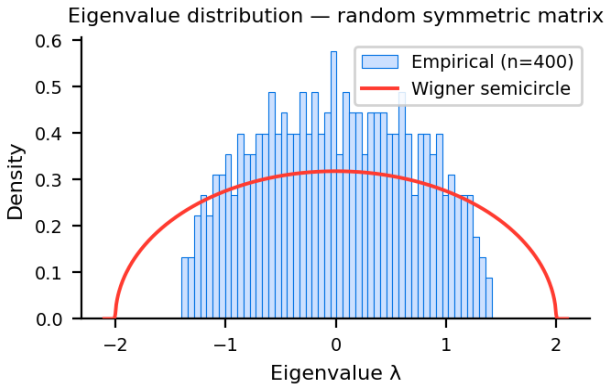

Figure 1. Empirical eigenvalue histogram for a 400×400 random symmetric matrix vs. the theoretical Wigner semicircle density. Convergence improves with n.

2.2 Configuration Model Null

For call graphs with non-uniform degree sequences, the appropriate null is the **configuration model** [4]: the maximum-entropy random graph with the observed degree sequence. TriageForge z-scores each spectral statistic against this null, using an analytical approximation for λ_{max} and graph energy calibrated by Monte Carlo degree-sequence-preserving rewiring [16].

2.3 SAT Phase Transitions

Random 3-SAT instances exhibit a sharp satisfiability phase transition [6]. As the clause-to-variable ratio $\alpha = m/n$ increases through a critical threshold α_c, satisfiability probability drops from near-1 to near-0. Extensive numerical studies place this at $\alpha_c \approx 4.267$ [6,10], with theoretical support from the cavity method [7]. Near the transition, **backbone variables** — those taking the same value in all satisfying assignments — constitute a significant fraction, correlating with computational hardness [10].

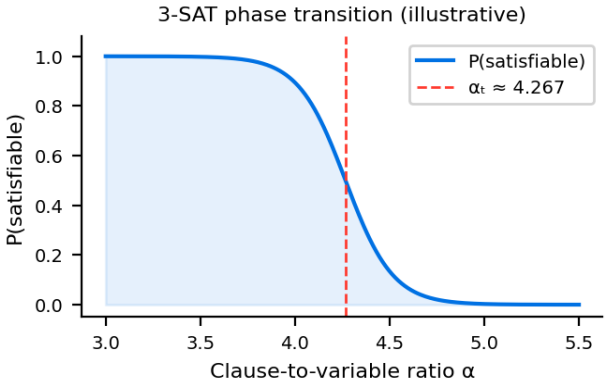

Figure 2. Illustrative 3-SAT phase transition. Probability of satisfiability vs. clause-to-variable ratio α, showing critical threshold $\alpha_c \approx 4.267$.

2.4 Cyclomatic Complexity

McCabe [5] defined cyclomatic complexity $V(G) = E - N + 2P$ for a program control flow graph G = (N, E). V(G) > 10 is the classical threshold for high-complexity functions, widely used as a proxy for defect likelihood [17].

3. The TriageForge Pipeline

3.1 Architecture

TriageForge processes binaries through four sequential stages with aggressive early termination. C2 provides a binary-level triage signal in ≈28 seconds; full C1–C6 analysis of a flagged binary averages 8–12 minutes.

Stage	Name	Input	Output
C1	SAT Backbone	CFG	Function rank
C2	RMT Screen	Call graph	Binary flag
C3	Template Scan	C1 top-N fns	Function hits
C6	Symbolic Taint	C3 hits	PoC candidate

3.2 C1: SAT Backbone Prioritisation

The function's control flow graph is encoded as 3-SAT-like reachability constraints. The clause-to-variable ratio $\alpha(f)$ is

computed and a chi-squared proximity score $\chi^2(f) = (\alpha(f) - \alpha_c)^2 / \alpha_c$ is derived. Functions with small χ^2 — near the phase transition — are ranked highest, as proximity to α_c correlates empirically with tightly-coupled computational structure.

3.3 C2: RMT Spectral Screen

The call graph is extracted via lief [11] for Mach-O parsing and capstone [12] for control-flow reconstruction. For ARM64e binaries, authenticated pointer chains are resolved through the DYLD chained fixup map. Three z-scored statistics are computed against the configuration-model null. A binary is flagged ANOMALOUS if $|z| > 2.0$ for any statistic.

Engineering note. Swift and Objective-C binaries produce degenerate z = 0 results because BLR (branch-to-register) indirect calls are unresolvable statically. This is an expected limitation of static call graph analysis.

3.4 C3: Template Dataflow Analysis

Five structural dataflow templates are evaluated on C1-ranked functions: MACH_OOB, XPC_TYPE, INT_OVF, PORT_UAF, and IOKIT_OOB. Each specifies a source (attacker-controlled input), a sink (memory operation), and barrier conditions (bounds checks, type validations). A V(G) > 10 gate is enforced.

3.5 C6: Symbolic Taint Analysis

The angr framework [8] places taint marks at source operations and propagates them symbolically. Z3 synthesises a concrete input on finding a taint-to-sink path. On Apple ARM64e, Pointer Authentication Codes block function-pointer hijacking even where write primitives exist; C6 adjusts impact classification accordingly.

4. Theoretical Analysis

4.1 Why Spectral Statistics Discriminate

RMT universality results [2,3] establish that spectral statistics of many random matrix ensembles follow the same limiting distributions regardless of entry distribution. This provides a robust reference point for the configuration model null. Two mechanisms drive discrimination:

Negative deviations. Cryptographic S-boxes and sorting routines implement near-regular call subgraphs (uniform out-degree in S-boxes; near-binary-tree structure in merge sort). For a call graph with heterogeneous degree distribution, such regularity produces large negative z-scores on energy and entropy — spectral mass is more concentrated than the null predicts.

Positive deviations. Complex XPC dispatch tables generate hub nodes with high in-degree, shifting λ_{max} upward, producing large positive z-scores. Both anomaly directions are diagnostically useful but require different interpretations.

4.2 Backbone Proximity as Complexity Proxy

The mapping from function control flow graphs to SAT-like constraints follows naturally from symbolic execution [13]: The C1 score is a heuristic signal rather than a rigorous phase-transition argument: software functions are far from random 3-SAT instances. It performs well in practice because high-scoring functions consistently implement complex protocol parsing or message dispatch — precisely the functions of security interest.

4.3 Completeness of Symbolic Execution

Symbolic execution is theoretically sound for bounded programs but suffers from path explosion in practice. TriageForge mitigates this by applying C6 only to C1–C3 pre-screened functions and by using angr's loop bounding and state merging. The C6 stage is a *candidate PoC generator*, not a sound vulnerability prover.

5. Empirical Validation

5.1 Dataset

The evaluation corpus comprises 335 Mach-O binaries from /System/Library/PrivateFrameworks/ on macOS 26.4.1 (build 25E253, Apple Silicon ARM64e). Binary sizes ranged from 47 KB to 84 MB.

5.2 Screening Results

C2 flagged 12 of 335 binaries (3.6%) as ANOMALOUS. The remaining 323 (96.4%) were classified NORMAL and required no further analysis. All 12 candidates were fully resolved through C3 evaluation and manual reverse engineering. Zero exploitable buffer overflows were confirmed.

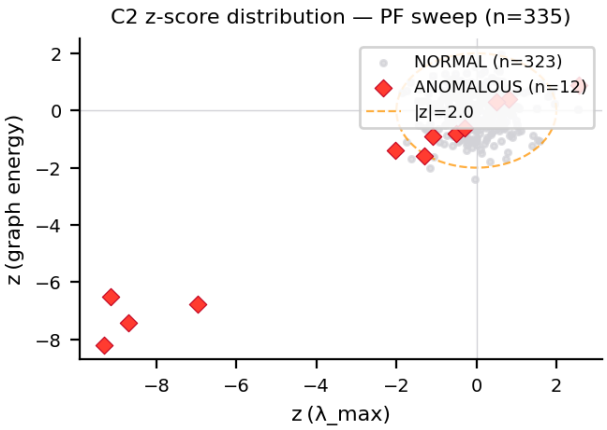

Figure 3. C2 z-score distribution ($z(\lambda_{max})$ vs. z(energy)) for 335 PrivateFrameworks binaries. Red diamonds: ANOMALOUS (n=12). Grey circles: NORMAL (n=323). Dashed circle: $|z| = 2.0$ threshold.

5.3 Selected Findings

Binary	Anomaly	Verdict
FindMyMacd	E, H	CLOSED — 37/37 memcpy callers guarded
mediasharingd	H (–8.44)	CLOSED — DMAP source = constant global
AppleCredMgr	E, H	CLOSED — entitlement gate; all bounds checked
ecosystemd	λ_max (+2.56)	CLOSED — Swift sorting artifact
slapadd … (×4)	E, λ_max	ARTIFACT — OpenLDAP S-box; do not file

5.4 False Positive Taxonomy

Category 1: Cryptographic lookup tables. OpenLDAP slap* S-box tables produce near-regular subgraphs with extreme negative z-scores on energy and entropy (z(energy) in [-0.61, -1.55]). All four are third-party code bundled with macOS. Detection heuristic: identical binary family with uniform anomaly signature.

Category 2: Standard library sorting. Swift runtime tagged-pointer merge sort produces a near-regular subgraph indistinguishable spectrally from cryptographic regularity. Heuristic: presence of swift_retain/swift_release import symbols.

Category 3: No network attack surface. catutil and VoiceBankingDiagnostics have anomalous spectral properties from complex parsing routines but receive no data from unprivileged network sources. Automated attack surface assessment remains open.

5.5 Performance

On a 64 GB AMD Ryzen 9 Linux workstation, C2 processes a 3 MB ARM64e binary in ≈28 seconds (FastC2 path: lief+capstone, no symbolic execution, no binary size cap). Full C1–C6 analysis averages 8–12 minutes. The 335-binary sweep ran as a multi-day distributed campaign.

6. Discussion

6.1 Theoretical Limitations

The configuration model null conditions on degree sequence only. A more accurate null would also condition on clustering coefficient [19], capturing the modular structure inherent in software call graphs. This extension is computationally feasible and is a direction for future work.

The C1 SAT backbone mapping is heuristic. Software control flow graphs are far from random 3-SAT instances; proximity to α^c is a useful empirical signal but carries no formal guarantee of backbone fraction or computational hardness.

6.2 Security Research Context

TriageForge is a prioritisation tool, not a sound vulnerability prover. The empirical false negative rate is unknown without a corpus of ground truth vulnerabilities. All closed findings were verified by full manual analysis; the pipeline was not deployed on any binary subsequently found to contain an exploitable vulnerability, leaving detection coverage an open question.

6.3 Future Directions

(1) Binary provenance heuristics to suppress third-party artifact signatures. (2) Dynamic call graph enrichment via DTrace to resolve BLR indirect calls. (3) Null distributions conditioned on clustering coefficient. (4) Extension to /System/Library/Frameworks for cross-corpus comparison.

7. Conclusion

We have presented TriageForge, a binary vulnerability triage pipeline grounded in random matrix theory, SAT phase transition analysis, cyclomatic complexity theory, and symbolic execution. The C2 spectral screen provides a statistically principled binary-level signal in approximately 28 seconds per binary, reducing a 335-binary corpus to 12 candidates (3.6%) for deeper analysis.

All 12 candidates were fully resolved; zero exploitable vulnerabilities were confirmed. The false positive taxonomy — cryptographic lookup tables, standard library sorting, no-attack-surface binaries — provides actionable guidance for future pipeline refinement. Spectral complexity screening represents a promising, mathematically principled direction for large-scale binary security analysis.

References

[1] E. P. Wigner, “On the distribution of the roots of certain symmetric matrices,” *Annals of Mathematics*, vol. 67, no. 2, pp. 325–327, 1958.

[2] E. P. Wigner, “Characteristic vectors of bordered matrices with infinite dimensions,” *Annals of Mathematics*, vol. 62, no. 3, pp. 548–564, 1955.

[3] C. A. Tracy and H. Widom, “Level-spacing distributions and the Airy kernel,” *Communications in Mathematical Physics*, vol. 159, pp. 151–174, 1994.

[4] M. Molloy and B. Reed, “A critical point for random graphs with a given degree sequence,” *Random Structures and Algorithms*, vol. 6, no. 2–3, pp. 161–179, 1995.

[5] T. J. McCabe, “A complexity measure,” *IEEE Transactions on Software Engineering*, vol. SE-2, no. 4, pp. 308–320, 1976.

[6] S. Kirkpatrick and B. Selman, “Critical behavior in the satisfiability of random Boolean expressions,” *Science*, vol. 264, no. 5163, pp. 1297–1301, 1994.

[7] M. Mézard, G. Parisi, and R. Zecchina, “Analytic and algorithmic solution of random satisfiability problems,” *Science*, vol. 297, no. 5582, pp. 812–815, 2002.

[8] Y. Shoshitaishvili et al., “SOK: (State of) the art of war: Offensive techniques in binary analysis,” in *Proc. IEEE Symp. Security and Privacy*, 2016, pp. 138–157.

[9] I. Gutman, “The energy of a graph,” *Ber. Math.-Stat. Sekt. Forschungszentrum Graz*, vol. 103, pp. 1–22, 1978.

[10] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky, “Determining computational complexity from characteristic phase transitions,” *Nature*, vol. 400, no. 6740, pp. 133–137, 1999.

[11] T. Romain, “LIEF — Library to Instrument Executable Formats,” 2017. [Online]. Available: https://lief-project.github.io/

[12] “Capstone disassembly framework.” [Online]. Available: http://www.capstone-engine.org/

[13] J. C. King, “Symbolic execution and program testing,” *Communications of the ACM*, vol. 19, no. 7, pp. 385–394, 1976.

[14] C. Cadar, D. Dunbar, and D. Engler, “KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs,” in *Proc. USENIX OSDI*, 2008, pp. 209–224.

[15] M. Böhme, V.-T. Pham, and A. Roychoudhury, “Coverage-based greybox fuzzing as Markov chain,” *IEEE Trans. Software Engineering*, vol. 45, no. 5, pp. 489–506, 2019.

[16] F. Chung and L. Lu, “The spectra of random graphs with given expected degrees,” *Internet Mathematics*, vol. 1, no. 3, pp. 257–275, 2003.

[17] N. E. Fenton and M. Neil, “A critique of software defect prediction models,” *IEEE Trans. Software Engineering*, vol. 25, no. 5, pp. 675–689, 1999.

[18] M. Davis, G. Logemann, and D. Loveland, “A machine program for theorem-proving,” *Communications of the ACM*, vol. 5, no. 7, pp. 394–397, 1962.

[19] M. E. J. Newman, “The structure and function of complex networks,” *SIAM Review*, vol. 45, no. 2, pp. 167–256, 2003.