

Are Tigress-Obfuscated Binaries Representative of Real Linux Malware? A Static Feature Measurement Study

Jayesh Gaba¹, Gurpreet Singh², Joseph Raymond V.³

Department of Networking and Communications, SRM Institute of Science and Technology, Kattankulathur, India.

E-mail: ¹jg9140@srmist.edu.in, ²gs3116@srmist.edu.in, ³josephrv@srmist.edu.in

Abstract

Malware corpora produced through source code obfuscation tools like Tigress are commonly employed for malware detection and binary analysis purposes; however, the representative nature of such malware corpora in relation to real-life Linux malware is far from sufficiently validated. This research work investigates whether binaries obfuscated via Tigress can replicate the static features of real Linux ELF malware. A total of 1,800 Linux ELF malware binaries from 2020-2025, 179 benign binaries, and 18 Tigress binaries were included in this study, 15 of which were successfully obfuscated. Section features were derived from 1,245 malware samples using the LIEF library, whereas entropy was computed for all 1,800 malware samples. Evidence of packing/obfuscation was determined via Detect-It-Easy signatures and entropy-based heuristics, after which comparative and architecture-controlled analysis was conducted. Obfuscation or packing was observed in 29.6% of the malicious samples, increasing from 19.7% in 2020 to 39.0% in 2025. The mean section entropy was higher for packed malware when compared to Tigress obfuscated binaries (6.98 vs. 5.90) and remained higher in the x86/x86-64 subset (6.91 vs. 5.90). Signature-based detection found 21.8% of the samples, whereas the entropy-based heuristic found 7.8% in total, but without any overlap. The distribution of sections was also significantly different between Tigress obfuscated binaries and packed malware. The considered configurations of Tigress thus fail to emulate several key static properties seen in the analysed set of Linux malwares.

Keywords: Malware Obfuscation, Synthetic Datasets, Tigress, Linux Malware, Static Analysis.

1. Introduction

Malwares routinely include obfuscation and packing to avoid detection and perform thwart analysis [1], [2]. To combat these malwares, there is an increasing need to develop controlled and reproducible experiments but often have no access to a variety of real malware samples. The standard solution consists of synthetic obfuscated datasets based on the source level such as Tigress [3]. Work across the field is based on Tigress-based corpora for training classifiers [4], [5], benchmarking deobfuscation tools [6], [7] and stress-testing static analyzers [8], [9].

This approach results in an assumption that the researchers rarely state explicitly: Tigress-obfuscated binaries must be representative of real-time malware along dimensions that are required for detection. If there is a systematic difference between synthetic samples and real samples from actual threats in observable static properties such as entropy distributions, symbol stripping patterns and section layouts, then detectors trained or evaluated on Tigress may generalize poorly to real attacks.

Despite the Tigress's widespread adoption, there is no empirical validation of this representativeness assumption, especially in the case of Linux ELF malware. Most of the obfuscation characterization work targets Windows PE binaries and commercial packers [10]–[12]. Meanwhile, Linux malware has evolved rapidly as it is driven by IoT botnets that target ARM and MIPS architectures [13]–[15].

A study of 1,800 Linux ELF malware samples from MalwareBazaar [16], spanning 2020 through 2025 is also conducted. After applying signature-based detection (Detect-It-Easy [17]) and entropy heuristics [18] to label packed samples, we compared these distributions against a Tigress-obfuscated corpus built from representative Linux utilities.

1.1 Contributions

This research work makes four contributions:

1. Quantitative characterization of Linux packing prevalence (29.6% overall) and its near-doubling from 2020 to 2025. There is no prior work that worked on analysing

- packing prevalence on Linux malware across multiple years at this scale.
2. Systematic comparison of Tigress versus real malware on static features, revealing a persistent entropy gap (5.90 vs. 6.98) that survives architecture controls.
 3. Empirical evidence that signature-based and entropy-based detection capture completely disjoint sample sets (21.8% vs. 7.8%, zero overlap), pointing to distinct obfuscation strategies—neither resembling Tigress.
 4. Actionable recommendations for improving Tigress-based evaluations through entropy augmentation, hybrid corpora, temporal recalibration, and transparent reporting.

2. Related Works

2.1 Linux Malware Ecosystems

Linux malware has seen a lot of development during the past 10 years and is using IoT botnets like Mirai [13], Gafgyt [15] and its various strains [19], [20]. They are targeting almost exclusively embedded devices that are based on non-x86 architecture, such as ARM, MIPS and others. In contrast to Windows, while Windows malware makes heavy use of packers such as UPX [21], Themida, and VMProtect, there is no systematic study of Linux malware obfuscation. Current research has done an excellent job of describing Linux malware characteristics and capabilities [22], [23], infection vectors [24] and networking behaviors [13]. What is missing is a systematic analysis of obfuscation prevalence and techniques through time. We present empirical evidence about Linux malware packing trends from 2020-2025.

2.2 Academic Obfuscation Tools

Tigress [3] performs source-level transformation techniques such as control flow flattening [25], virtualization [26], opaque predicate insertion, [27] and literal encoding. The use of Tigress in malware research is vast. Wang et al. [4] used Tigress to develop neural decompilers; Luo et al. [5] employed it to evaluate binary code similarity under obfuscation-resilient conditions; and Xu et al. [6] used Tigress in neural program deobfuscation. Nevertheless, there is no empirical verification to show if Tigress is a good enough model for practical obfuscation.

Alternative approaches O-LLVM [28] and VMProtect [29] exist but Tigress performed well due to its availability and broad transformation list.

2.3 Packing and Obfuscation Detection

Packing detection involves two major methods. One is based on signature analysis, where patterns are analysed in relation to packer signatures. Tools such as DIE (Detect-It-Easy) [17] and PEiD [30] employ signature databases. This method proved effective for detecting commercial packers but is inefficient in detecting customized packers. Entropy detection involves identifying high levels of randomness in the code, which indicates that it has been compressed or encrypted. Lyda and Hamrock [18] first used entropy in packing malware. Shafiq et al. [12] developed entropy using PE structures. Nearly all this work targets Windows PE binaries; Linux ELF malware has received far less attention.

2.4 Synthetic Malware Generation

Several studies have generated synthetic malware corpora [31]-[33] but to our knowledge, synthetic obfuscation reproduces real-time malware distributions on the static features that are utilized by detection systems.

3. Methodology

3.1 Datasets

Malware Corpus: We collected Linux ELF malware from MalwareBazaar [16], a public threat intelligence repository maintained by abuse.ch. To enable temporal analysis, we used stratified sampling based on the `first_seen` timestamp, selecting 300 samples from each year spanning 2020 through 2025. This yielded 1,800 samples with balanced temporal representation. The analysis was performed at the binary-sample level, and no additional family-level deduplication was applied.

After downloading binaries via the MalwareBazaar API and extracting them from password-protected ZIP archives (password: "infected"), we analyzed them in an isolated Ubuntu 24.04 LTS virtual machine with network access disabled. All analysis was entirely static—no execution.

Architecture Distribution: The corpus reflects Linux malware's architectural diversity: ARM (35.4%), MIPS (20.1%), x86 (12.6%), x86-64 (8.1%), PowerPC (6.4%), SH (5.7%), M68K (5.0%), SPARC (4.6%), and unrecognized/other (4.1%). This distribution aligns with IoT botnet prevalence patterns [13], [14].

Benign Corpus: The 179 benign ELF binaries used in our research include those from the core utilities (coreutils), binary utilities (binutils), system utilities (util-linux), networking, development, and system daemons of a stock Ubuntu 22.04 LTS distribution. They are all x86-64 binaries, compiled using normal distribution build options (-O2 with GCC). All the malware used for analysis originated from open-source threat intelligence feeds that are clearly meant for academic purposes [16]. The whole process of analysis was performed in isolation using only static analysis methods [34]. The malware dataset has different architectures, while the benign dataset comprises x86-64 executables. Since executables and their entropy can differ depending on the architecture, direct comparisons between these variables with respect to the whole malware dataset were taken in a descriptive manner. An extra analysis was carried out with only x86 and x86-64 malware. This subset contained 371 malware samples, including 21 packed samples, and the resulting entropy differences were compared with those obtained from the complete corpus.

3.2 Feature Extraction

We used LIEF (Library to Instrument Executable Formats) version 0.14.1 [35] to extract structural and entropy-based features. For each ELF binary, we computed: per-section Shannon entropy (maximum and mean), symbol table presence (.symtab, .dynsym), binary stripping status, .text section size and entropy, and whole-file entropy for samples where LIEF could not parse section structures.

LIEF successfully parsed section structures for 1,245 of the 1,800 malware samples (69.2%). Section-level features, including maximum and mean section entropy, section count, symbol information, and .text characteristics, were therefore computed only for these parseable samples. For the remaining 555 samples (30.8%), which were predominantly associated with UPX-packed binaries, section-level measurements were unavailable, and whole-file entropy was used instead. Accordingly, section-level comparisons are reported only for samples with successfully parsed ELF section structures, whereas whole-file entropy is used for the complete malware corpus.

3.3 Identifying Packed Malware

We employed two complementary detection methods following established approaches [10], [18]:

Signature-Based Detection (DIE): Detect-It-Easy (DIE) version 3.10 [17] scanned all 1,800 samples via command-line interface with JSON output. DIE identified packers in 392 samples (21.8%). UPX [21] dominated overwhelmingly: 390 of 392 detections (99.5% of signature-based identifications).

Entropy-Based Heuristic: We labeled a sample as "likely packed" when: $\text{max_section_entropy} > 6.5$ AND $\text{is_stripped} = 1$. The 6.5 threshold came from empirical analysis of our corpus—specifically, the 85th percentile of maximum section entropy among parseable samples. This falls below the 7.0–7.5 range typical for Windows PE malware [12], reflecting different obfuscation practices in Linux ecosystems. The stripping requirement captures the widespread practice of removing symbol tables to hinder reverse engineering [2]. This heuristic flagged 141 samples (7.8%).

Entropy Threshold: A maximum section entropy threshold of 6.5 was used for the entropy-based heuristic. This threshold was selected based on the 85th percentile of maximum section entropy among the parseable malware samples. The threshold was applied together with the symbol-stripping criterion ($\text{is_stripped} = 1$) to identify samples exhibiting likely packing or obfuscation characteristics.

Combined Obfuscation/Packing Label: The two detection methods were combined using a logical OR: $\text{evidence_of_packing_or_obfuscation} = \text{DIE_detected} \text{ OR } \text{entropy_heuristic}$. This produced 533 samples (29.6%). Samples detected by DIE were treated as signature-confirmed packer detections, whereas samples identified only by the entropy heuristic were treated as exhibiting likely packing or obfuscation characteristics rather than confirmed use of a specific packer.

3.4 Tigress-Obfuscated Corpus

Tigress operates at the C source level and proves incompatible with production codebases that rely on GCC-specific extensions, complex macro systems, and intricate build configurations. We extracted core algorithms from three representative coreutils utilities (cat, wc, and echo) and reimplemented them in ISO C99 without GNU extensions. Our implementations preserve the original control-flow structure, algorithmic complexity, and functional equivalence, verified through output comparison on test inputs.

We applied six Tigress (version 4.0.11) [3] transformation profiles: (1) Flatten—control-flow flattening using dispatcher-based switch statements [25]; (2) Virtualize—VM-based obfuscation translating code to custom bytecode [26]; (3) Opaque—opaque predicate insertion (5 predicates per function) [27]; (4) Literals—encoding of string and integer literals through arithmetic transformations; (5) Arithmetic—mixed Boolean-Arithmetic (MBA) encoding [37]; and (6) Combined—Flatten and Opaque transformations applied together.

Across 18 attempted transformations, we obtained 3 baseline binaries and 15 obfuscated binaries (83% success rate). We applied GNU strip to all Tigress binaries, removing symbol tables to match the 99% stripping rate in packed malware.

3.5 Analysis Methodology

Given the substantially different sample sizes across groups and the small number of Tigress-obfuscated binaries, the analysis emphasizes descriptive statistics, distributional comparisons, and effect sizes rather than relying solely on statistical significance testing. Mean values, medians, and standard deviations are provided for the main entropy metrics, while Cohen's *d* is applied to measure the effect size between the samples of obfuscated and packed malware. In some cases, it is necessary to include confidence intervals, which show the level of uncertainty related to the difference estimates, especially for the comparison of the Tigress dataset.

For the qualitative validation of the automatically generated detection patterns, 50 samples were picked randomly from each detection category for manual examination. The manual examination was carried out for packer signatures, section structure, entropy values, symbol stripping, and other aspects of ELF structural analysis.

3.6 Limitations and Conservative Bias

There were 555 out of 1766 samples (30.8%) that LIEF was unable to analyze, most of which were UPX packed binaries with corrupted/encrypted section headers [21]. Of these 555 samples, manual analysis found that 390 samples (70.3%) had been categorized as UPX packed by DIE. These exclusions lead us to favorably bias the analysis towards Tigress, as the most heavily obfuscated samples are excluded and hence are more entropic.

Only 15 obfuscated binaries were observed due to the failure of 17% of Tigress compilations. Nevertheless, the enormous effect size (Cohen's $d = 3.5$), along with the fact that all 15 samples cluster below the median of the packed malware sample set, indicate otherwise.

The Tigress corpus is comprised of reimplementations of coreutils ranging in size from 31 to 85 lines of code—far simpler than actual malware botnets which include cryptography, network protocol implementation, and persistence mechanisms.

Static-only features: This is because we specifically concentrate on static binary features that have been used previously in literature [11], [12] and form the basis of several Tigress experiments [7], [8].

4. Results and Discussion

For the implementation of the analysis process explained in Section 3.5, the extracted static features were subjected to descriptive statistics, effect size computation, architecture-controlled comparison, and manual inspection. For the main entropy metrics, the means, medians, and standard deviations were computed for benign programs, non-packed malicious software, packed malicious software, the Tigress baseline set, and the Tigress-obfuscated set. The analysis focuses mainly on effect size and distributional properties because the set of Tigress-obfuscated binaries only has 15 samples. The difference between the sets of Tigress-obfuscated binaries and the packed malicious software was quantified using Cohen's d . Furthermore, an additional architecture-controlled analysis was carried out by considering only the x86 and x86-64 malicious software to see whether the difference in entropy can be explained by the composition of the architectures. In addition, the random selection of samples from the DIE-detected and the entropy heuristic classes was done for manual inspection of packer signature, section structure, entropy values, and symbol-stripping patterns.

4.1 Prevalence and Temporal Evolution

Overall Prevalence: From a total of 1,800 malware instances, 533 samples (29.6%) showed signs of packaging or obfuscation. This table shows an interesting trend of detection: DIE signature alone: 392 samples (21.8%) – mostly UPX (390 out of 392).

There is no overlap in the detection technique used, which means that there are only two distinct patterns. The signature-based detection detects commercial packers that include packer

compressions and unpacking stubs. Manual inspection of 50 DIE-detected samples revealed that they have medium entropy (mean 6.85 in the chosen subset) and UPX-like structural indicators. On the other hand, the entropy-based detection method detects high entropy samples (mean 7.12 for 50 samples analyzed) and heavy symbol removal without any packer signatures, meaning that these could be new packers not yet in the DIE database.

Temporal Trends: Table 1 records the prevalence rate of packing over the years. Prevalence rises from 19.7% in 2020 to 39.0% in 2025, more than doubling in five years with an average annual growth.

Table 1. Packing prevalence by year (2020–2025)

Year	Packed	Total	Prevalence (%)
2020	59	300	19.7
2021	73	300	24.3
2022	102	300	34.0
2023	107	300	35.7
2024	75	300	25.0
2025	117	300	39.0

This trend exhibits non-monotonic behavior, with a significant trough in 2024 (25.0%). This could be due to sampling error, changes in the MalwareBazaar submission process, or the actual dynamics of the ecosystem. However, what is important is that the trend has clear implications: a Tigress corpus created once will be dated very quickly.

4.2 Tigress vs. Packed Malware

Descriptive and Effect-Size Analysis: The extracted section-level entropy values were summarized by calculating the means, medians, and standard deviations. Table 2 shows that the mean maximum section entropy for packed malware was 6.98 (median = 6.89, standard deviation = 0.24), while for Tigress-obfuscated binaries, it was 5.90 (median = 5.88, standard deviation = 0.27). This resulted in an entropy difference of 1.08 bits. Cohen's *d* value was 3.5. In addition, all 15 Tigress-obfuscated binaries had lower entropy values than the packed malware's median of 6.89, while 14 of the 15 had entropy values below the 25th percentile.

Table 2. Section entropy summary statistics by group

Group	Mean Entropy	Median Entropy	Std Dev
Benign	6.22	6.28	0.36
Malware (Non-packed)	6.07	6.10	0.31
Malware (Packed)	6.98	6.89	0.24
Tigress (Baseline)	5.69	5.72	0.28
Tigress (Obfuscated)	5.90	5.88	0.27

In comparison with within-group variance (SDs of 0.24 to 0.36), the difference is quite large. The Cohen’s *d* value between Tigress-obfuscated and packed malware is approximately 3.5, which constitutes a large effect according to standard conventions [38]. All 15 of the Tigress-obfuscated malware samples are below the median of packed malware (6.89).

To examine whether the entropy gap was caused by architectural heterogeneity in the malware dataset, the experiment was carried out again on x86 and x86-64 malware. This sample consisted of 371 malware instances, which included 21 instances of packed malware. The average maximum entropy of sections in packed x86/x86-64 malware was 6.91 (median = 6.85) while that of Tigress-obfuscated x86-64 binaries was 5.90. This resulted in a gap of 1.01 bit, which is nearly the same as the 1.08-bit gap obtained in the entire dataset, with a difference of only 0.07 bits.

Table 3. Feature comparison across groups

Group	Entropy	Stripped	Sections
Benign	6.22	0%	30.4
Malware (Non-packed)	6.07	61%	16.5
Malware (Packed)	6.98	99%	11.9
Tigress (Baseline)	5.69	100%	30.0
Tigress (Obfuscated)	5.90	100%	30.0

4.3 Two Detection Patterns

The tool DIE marked up 21.8% (392) of files, all but one as UPX packed. The manual analysis of 50 randomly chosen UPX packed binaries showed characteristic features as such as the presence of structural abnormalities via signatures (UPX headers, packing stubs on entry points), average entropy (the observed average is 6.85 for inspected binaries), and simple section layouts. The entropy heuristic identified another 7.8% (141 binaries) that were not flagged by DIE. Analysis of 50 randomly chosen samples from this group resulted in the following observations:

High maximum entropy of sections (the observed average of maximum entropy is 7.12 in inspected binaries, median of maximum entropy is 7.08), aggressive symbol stripping (100%), simple section layouts (average number of sections is 8.3), and no packer signatures.

The lack of overlap in these two approaches means that each approach is detecting something that is totally different from the other. Tigress does not fit either category. It agrees on stripping symbols (100%) but disagrees significantly on entropy (5.90 against 6.85–7.12) and number of sections (30 sections against 8–12).

5. Conclusion

The study was conducted to determine whether binaries are encoded using Tigress obfuscation techniques provided a good approximation of actual Linux malware binaries. This was based on a comparison between the static characteristics of Tigress-encoded binaries and actual Linux ELF malware. Analysis of 1,800 samples of Linux malware binaries obtained between 2020 and 2025, 179 benign binaries, and 18 Tigress-encoded binaries yielded the following results. First, Packing/obfuscation signatures were detected in 29.6% of malware samples, rising from 19.7% in 2020 to 39.0% in 2025, reflecting the evolution of obfuscation techniques for Linux malware over time. Second, Tigress-obfuscated binaries had significantly lower section entropy than packed malware (5.90 compared to 6.98). Tigress also maintained a significantly higher number of sections than packed malware, although rates of symbol stripping were quite similar. Third, signature-based detection and entropy-based heuristics showed distinct sets of packed samples, whereas Tigress did not conform to either trend. These results show that Tigress, at least within the examined configurations, does not sufficiently reflect

the static properties of modern Linux malware and thus constrains the degree of realism for Tigress-based benchmarking and detector evaluation. A realistic evaluation must involve a set of malware-inspired applications, entropic properties, hybrid datasets, time variability, and honest documentation of the dataset building procedure. In future work, we plan to further extend the Tigress dataset through complex programs, examine dynamic and semantic properties, and compare other obfuscation systems, namely O-LLVM and VMProtect.

References

- [1] You, Ilsun, and Kangbin Yim. "Malware Obfuscation Techniques: A Brief Survey." In 2010 International Conference on Broadband, Wireless Computing, Communication and Applications, IEEE, 2010, 297-300.
- [2] Roundy, Kevin A., and Barton P. Miller. "Hybrid Analysis and Control of Malware." In International Workshop on Recent Advances in Intrusion Detection, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, 317-338.
- [3] C. Collberg, "Tigress C Diversifier/Obfuscator," <http://tigress.cs.arizona.edu>, 2016, accessed: 2024.
- [4] He, Jingxuan, Pesho Ivanov, Petar Tsankov, Veselin Raychev, and Martin Vechev. "Debin: Predicting Debug Information in Stripped Binaries." In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, 2018, 1667-1680.
- [5] Luo, Lannan, Jiang Ming, Dinghao Wu, Peng Liu, and Sencun Zhu. "Semantics-based Obfuscation-Resilient Binary Code Similarity Comparison with Applications to Software Plagiarism Detection." In Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering, 2014, 389-400.
- [6] Xu, Xiaojun, Chang Liu, Qian Feng, Heng Yin, Le Song, and Dawn Song. "Neural Network-Based Graph Embedding for Cross-Platform Binary Code Similarity Detection." In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017, 363-376.
- [7] Banescu, Sebastian, Christian Collberg, Vijay Ganesh, Zack Newsham, and Alexander Pretschner. "Code Obfuscation Against Symbolic Execution Attacks." In Proceedings of the

- 32nd Annual Conference on Computer Security Applications, 2016, 189-200.
- [8] Schrittwieser, Sebastian, Stefan Katzenbeisser, Johannes Kinder, Georg Merzdovnik, and Edgar Weippl. "Protecting Software Through Obfuscation: Can It Keep Pace with Progress in Code Analysis?." *Acm Computing Surveys (csur)* 49, no. 1 (2016): 1-37.
 - [9] Cauligi, Sunjay, Craig Disselkoen, Daniel Moghimi, Gilles Barthe, and Deian Stefan. "SoK: Practical Foundations for Software Spectre Defenses." In *2022 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2022, 666-680.
 - [10] Ugarte-Pedrero, Xabier, Davide Balzarotti, Igor Santos, and Pablo G. Bringas. "SoK: Deep Packer Inspection: A Longitudinal Study of the Complexity of Run-Time Packers." In *2015 IEEE Symposium on Security and Privacy*, IEEE, 2015, 659-673.
 - [11] R. Perdisci, A. Lanzi, and W. Lee, "Classification of Packed Executables for Accurate Computer Virus Detection," *Pattern Recognition Letters*, vol. 29, no. 14, 2008, 1941–1946.
 - [12] Shafiq, M. Zubair, S. Momina Tabish, Fauzan Mirza, and Muddassar Farooq. "Pe-miner: Mining Structural Information to Detect Malicious Executables in Realtime." In *International Workshop on Recent Advances in Intrusion Detection*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, 121-141.
 - [13] Antonakakis, Manos, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric et al. "Understanding the Mirai Botnet." In *26th USENIX Security Symposium (USENIX Security 17)*, pp. 1093-1110. 2017.
 - [14] Kolias, Constantinos, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. "DDoS in the IoT: Mirai and Other Botnets." *computer* 50, no. 7 (2017): 80-84.
 - [15] Pa, Yin Minn Pa, Shogo Suzuki, Katsunari Yoshioka, Tsutomu Matsumoto, Takahiro Kasama, and Christian Rossow. "{IoT POT}: Analysing the Rise of {IoT} Compromises." In *9th USENIX Workshop on Offensive Technologies (WOOT 15)*. 2015.
 - [16] abuse.ch, "MalwareBazaar: Malware Sample Sharing Platform," <https://bazaar.abuse.ch>, 2024.
 - [17] horsicq, "Detect It Easy (DIE)," <https://github.com/horsicq/Detect-It-Easy>, 2024.
 - [18] Lyda, Robert, and James Hamrock. "Using entropy analysis to find encrypted and packed

- malware." *IEEE security & privacy* 5, no. 2 (2007): 40-45.
- [19] Marzano, Artur, David Alexander, Osvaldo Fonseca, Elvertton Fazzion, Cristine Hoepers, Klaus Steding-Jessen, Marcelo HPC Chaves, Ítalo Cunha, Dorgival Guedes, and Wagner Meira. "The evolution of bashlite and mirai iot botnets." In 2018 IEEE Symposium on Computers and Communications (ISCC), pp. 00813-00818. IEEE, 2018.
- [20] De Donno, Michele, Nicola Dragoni, Alberto Giaretta, and Angelo Spognardi. "DDoS-capable IoT malwares: comparative analysis and Mirai investigation." *Security and Communication Networks* 2018, no. 1 (2018): 7178164.
- [21] UPX Team, "UPX: The Ultimate Packer for eXecutables," <https://upx.github.io>, 2024.
- [22] Cozzi, Emanuele, Mariano Graziano, Yanick Fratantonio, and Davide Balzarotti. "Understanding Linux Malware." In 2018 IEEE Symposium on Security and Privacy (SP), IEEE, 2018, 161-175.
- [23] Bekerman, Dmitri, Bracha Shapira, Lior Rokach, and Ariel Bar. "Unknown Malware Detection Using Network Traffic Classification." In 2015 IEEE Conference on Communications and Network Security (CNS), IEEE, 2015, 134-142.
- [24] CRossow, Christian, Dennis Andriesse, Tillmann Werner, Brett Stone-Gross, Daniel Plohmann, Christian J. Dietrich, and Herbert Bos. "Sok: P2pwned-Modeling and Evaluating the Resilience of Peer-to-Peer Botnets." In 2013 IEEE Symposium on Security and Privacy, IEEE, 2013, 97-111.
- [25] Wang, Chenxi, Jonathan Hill, John Knight, and Jack Davidson. *Software Tamper Resistance: Obstructing Static Analysis of Programs*. Technical Report CS-2000-12, University of Virginia, 12 2000, 2000.
- [26] Sharif, Monirul Islam, Andrea Lanzi, Jonathon T. Giffin, and Wenke Lee. "Impeding Malware Analysis Using Conditional Code Obfuscation." In NDSS. 2008.
- [27] Collberg, Christian, Clark Thomborson, and Douglas Low. *A Taxonomy of Obfuscating Transformations*. 1997.
- [28] Junod, Pascal, Julien Rinaldini, Johan Wehrli, and Julie Michielin. "Obfuscator-LLVM--Software Protection for the Masses." In 2015 ieee/acm 1st international workshop on software

- protection, IEEE, 2015, 3-9.
- [29] VMProtect Software, "VMProtect Software Protection," <https://vmpsoft.com>, 2024.
- [30] PEiD Team, "PEiD: PE Identifier," <https://www.aldeid.com/wiki/PEiD>, 2024.
- [31] Dahl, George E., Jack W. Stokes, Li Deng, and Dong Yu. "Large-scale Malware Classification Using Random Projections and Neural Networks." In 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, IEEE, 2013, 3422-3426.
- [32] Raff, E., J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. Nicholas. "Malware Detection by Eating a Whole Executable." In Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, 2018, 268-276.
- [33] Anderson, Hyrum S., Anant Kharkar, Bobby Filar, David Evans, and Phil Roth. "Learning to Evade Static PE Machine Learning Malware Models via Reinforcement Learning." arXiv preprint arXiv:1801.08917 (2018).
- [34] Rossow, Christian, Christian J. Dietrich, Chris Grier, Christian Kreibich, Vern Paxson, Norbert Pohlmann, Herbert Bos, and Maarten Van Steen. "Prudent Practices for Designing Malware Experiments: Status Quo and Outlook." In 2012 IEEE Symposium on Security and privacy, IEEE, 2012, 65-79.
- [35] Thomas, Romain. "Lief-library to Instrument Executable Formats (2017)." URL: <https://lief.quarkslab.com>.
- [36] Zhou, Dongchao, Lingyun Ying, Huajun Chai, and Dongbin Wang. "From Obfuscated to Obvious: A Comprehensive JavaScript Deobfuscation Tool for Security Analysis." arXiv preprint arXiv:2512.14070 (2025).
- [37] Zhou, Yongxin, Alec Main, Yuan X. Gu, and Harold Johnson. "Information Hiding In Software with Mixed Boolean-Arithmetic Transforms." In International Workshop on Information Security Applications, Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, 61-75.
- [38] J. Cohen, Statistical Power Analysis for the Behavioral Sciences, 2nd ed. Hillsdale, NJ: Lawrence Erlbaum Associates, 1988.