Evolutionary Computation for Improving Malware Analysis

Kevin Leach
University of Michigan
kjleach@umich.edu

Ryan Dougherty
Arizona State University
redoughe@asu.edu

Chad Spensky
UC Santa Barbara
cspensky@cs.ucsb.edu

Stephanie Forrest
Arizona State University
stephanie.forrest@asu.edu

Westley Weimer
University of Michigan
weimerw@umich.edu

ABSTRACT

Research in genetic improvement (GI) conventionally focuses on the improvement of software, including the automated repair of bugs and vulnerabilities as well as the refinement of software to increase performance. Eliminating or reducing vulnerabilities using GI has improved the security of benign software, but the growing volume and complexity of malicious software necessitates better analysis techniques that may benefit from a GI-based approach. Rather than focus on the use of GI to improve individual software artifacts, we believe GI can be applied to the tools used to analyze malicious code for its behavior. First, malware analysis is critical to understanding the damage caused by an attacker, which GI-based bug repair does not currently address. Second, modern malware samples leverage complex vectors for infection that cannot currently be addressed by GI. In this paper, we discuss an application of genetic improvement to the realm of automated malware analysis through the use of variable-strength covering arrays.

1 INTRODUCTION

Malicious software (malware) has proliferated in the past few years, significantly eroding user and corporate privacy and trust in computer systems [6, 11]. A combination of manual and automated analyses are required for understanding new malware samples [4, 12]. Unfortunately, a growing number of new malware samples employ evasive or stealthy techniques to avoid or subvert automated analysis [2, 9, 10]. These stealthy samples operate by detecting features or *artifacts* of the analysis environment in which it executes (e.g., virtual machines may expose virtual devices named "VMWare Hard Disk" or may not fully implement all CPU instructions [13, Table V]; malware may check for mouse movement or keystrokes during execution [9]). When a sample detects an artifact, the malware can decide not to execute, thus hiding its behavior from a well-intentioned analyst or automated analysis tool. These stealthy samples require additional steps, effort, and computational resources to *mitigate* the presence of artifacts so that stealthy samples cannot determine whether they are under analysis. As millions of new samples are created each year [6], and manual analysis effort is burdensome, higher-throughput automated analysis solutions are needed. Given the cost (e.g., runtime overhead, implementation time, etc.) associated with artifact mitigation, there is an opportunity to improve analysis efficiency by determining which artifacts should be mitigated in which combinations to maximize success against a large corpus of such stealthy malware.

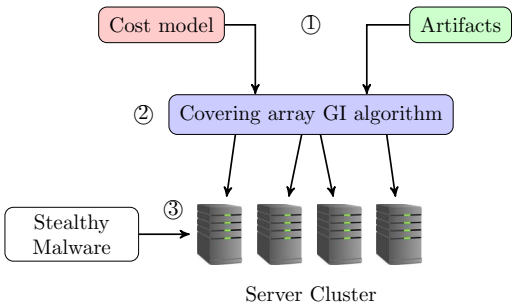

Figure 1: Proposed workflow for automated malware analysis. Given a cluster of analysis servers, a corpus of stealthy malware, a cost model of analysis configurations, and a set of artifacts used to achieve stealthy behavior, we propose using genetic improvement to find a low-cost (e.g., high efficiency) set of configurations for each analysis server that covers all artifacts used by malware in the corpus.

We propose extending state-of-the-art automated malware analysis techniques with a consideration of the *cost* and *coverage* of artifact mitigation strategies. With the increasing prevalence of stealthy malware and corresponding anti-stealth techniques, we must consider *which set* of artifacts should be mitigated during analysis. Deciding which mitigation strategy to take during analysis is not simple as many artifacts exist [9, 13] and are used to evade detection [10]. While all samples could be defeated with a high-cost analysis technique, we observe that each sample could individually be defeated by at least one low-cost (and efficient) analysis tool: we seek to identify a small, efficient set of *covering* mitigation strategies. Alternatively, we can control which artifacts to expose to each sample, and accept that we risk analysis failure for some samples in exchange for gaining overall analysis throughput. Our insight is that this problem formulation is related to a theoretical application of *covering array* algorithms from the domain of software testing, where each row in the array represents a test, and each column represents a software component [5]. By constructing arrays that denote which artifacts are mitigated as well as a model of the cost associated with each mitigation, we can use GI to explore the trade-off space between *cost* and *mitigation coverage*. This approach can allow us to improve the efficiency of automated malware analysis while retaining analysis fidelity.

GI'19, May 26, 2019, Montreal, Canada
2019. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Table 1: Example of configurations of a 2-server analysis cluster with 2 artifacts and associated (approximate) costs.

Config	Server	Screen	Debugger	Cost
1	1	✗	✗	0
	2	✗	✗	0
2	1	✓	✓	2
	2	✓	✓	2
3	1	✓	✗	1
	2	✗	✓	1

2 MOTIVATING EXAMPLE

We seek to lower the cost of the automated analysis of stealthy malware samples by using GI to explore automated malware analysis systems configurations. Consider a scenario in which an enterprise seeks to analyze a large corpus of stealthy malware samples with a fixed set of analysis servers. Each server can be configured to automatically analyze samples using tools with various levels of performance overhead (cost) and mitigated artifacts (coverage). Each server's configuration can be represented using an array of binary values that correspond to which artifact is mitigated by which tool.

Table 1 illustrates an example of our proposed approach in which we configure a 2-server automated analysis cluster over two artifacts, screen resolution and debugger presence. First, a stealthy malware sample could detect an analysis environment that is 'headless' by measuring screen resolution or monitor connectivity. Second, a sample could detect the presence of an attached debugger seeking to trace malware execution. Both artifacts can be mitigated, but for a cost. In the Table, we show three potential configurations of two servers. First (row 1), we could mitigate neither artifact on either server, incurring the lowest cost analysis (i.e., 0), but would not cover samples using those two artifacts (they would detect the analysis, avoid malicious behavior, and appear benign). Second (row 2), we could configure both servers to mitigate both artifacts. Doing so would cover all samples, but also incurs high cost. Finally (row 3), we could find a balanced configuration in which each server mitigates one artifact. Collectively, both servers incur lower cost while retaining coverage of both artifacts. We propose to use genetic improvement algorithms to explore this tradeoff space. Figure 1 illustrates a proposed workflow incorporating this concept.

3 VARIABLE-STRENGTH COVERING ARRAYS

A *covering array* (CA) is an array of integers where each column is a *factor* of the system being tested, and the rows represent individual tests performed on the system [3]. Each CA has an associated *strength*, which is the maximum size of any interaction of components being tested. For example, if we want to test any combination of three factors possibly interacting, then the strength is also three. The CA guarantees that no matter which set of factors are tested, then all possible ways of setting values to those factors are tested in some row. However, suppose that this is not always needed, in that not all combinations of factors need to be tested, or that different combinations of different sizes need to be tested. For example, if we have five factors a, b, c, d, e, we may only need to guarantee coverage for $\{a, b\}, \{b, c, d\}, \{a, d, e\}, \{b, c, e\}$, instead of all three-way interactions as would be in the general model. This is known as a *variable-strength* CA, written VSCA [7]. In contrast to the traditional use of VSCAs for software testing, we instead construct a VSCA to mitigate artifacts to increase the efficiency of automated stealthy malware analysis. Each artifact corresponds to a column of the VSCA, and each entry in the VSCA is a 0 or 1, to indicate whether or not the artifact is to be mitigated. The VSCA guarantees that for any choice of artifacts according to the model, then some row mitigates against all (or a subset) of them. We have a set of *mitigation strategies*, in which each has an associated *coverage*. For each row in the VSCA, we seek to find a minimum-cost choice of mitigation strategies such that the row is covered by the strategy.

We propose to use genetic algorithms both for generation of the VSCA and for finding the minimum-cost set of mitigation strategies. Genetic algorithms have been used for CAs [8], as well as for weighted set cover [1], but (1) they have not been used for VSCAs, and (2) since some mitigation strategies cannot (or should not) be paired together, we desire to find a *conflict-free* minimum-cost set cover. The cost here is thus a more general function that depends on (1) the individual costs of the tools themselves, (2) the number of tools chosen, and (3) which tools were chosen. Given such a function, we propose the use of genetic algorithms to search the space of mitigation strategies for a low-cost solution.

4 CONCLUSION

Stealthy malware is a growing concern. Lightweight automated malware analysis techniques must be balanced with heavier-duty analysis tools to fully analyze and understand larger corpora of stealthy malware. We suggest a collaboration between the GI and security communities to investigate approaches to explore tradeoffs between analysis cost and stealthy malware coverage.

REFERENCES

[1] J. E. Beasley and P. C. Chu. A genetic algorithm for the set covering problem. *European journal of operational research*, 94(2):392–404, 1996.

[2] X. Chen, J. Andersen, Z. Mao, M. Bailey, and J. Nazario. Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware. In *Proceedings of the 38th Annual IEEE International Conference on Dependable Systems and Networks (DSN '08)*, 2008.

[3] C. J. Colbourn. Combinatorial aspects of covering arrays. *Le Matematiche (Catania)*, 58(121-167):0–10, 2004.

[4] D. Farmer and W. Venema. *Forensic Discover*. Addison-Wesley, 2005.

[5] A. Hartman and L. Raskin. Problems and algorithms for covering arrays. *Discrete Mathematics*, 284:149–156, 2004.

[6] Kaspersky Lab. Kaspersky Security Bulletin 2017. https://media.kaspersky.com/jp/pdf/pr/Kaspersky_KSB2017_Statistics-PR-1045.pdf.

[7] S. Raaphorst, L. Moura, and B. Stevens. Variable strength covering arrays. *Journal of Combinatorial Designs*, page to appear, 2018.

[8] S. Sabharwal, P. Bansal, and N. Mittal. Construction of t-way covering arrays using genetic algorithm. *International Journal of System Assurance Engineering and Management*, 8(2):264–274, 2017.

[9] C. Spensky, H. Hu, and K. Leach. LO-PHI: Low observable physical host instrumentation. In *Networks and Distributed Systems Security Symposium 2016 (NDSS 2016)*, San Diego, CA, February 2016.

[10] S. Stefnisson. Evasive malware now a commodity. https://www.securityweek.com/evasive-malware-now-commodity, 2018.

[11] Symantec. Internet security threat report. https://www.symantec.com/content/dam/symantec/docs/reports/istr-22-2017-en.pdf, 2017.

[12] L. Zelster. Mastering 4 stages of malware analysis. https://zeltser.com/mastering-4-stages-of-malware-analysis/, February 2015.

[13] F. Zhang, K. Leach, H. Wang, A. Stavrou, and K. Sun. Using Hardware Features to Increase Debugging Transparency. In *Proceedings of the 36th IEEE Symposium on Security and Privacy*, 2015.