Evolutionary Antivirus Signature Optimization

Evolutionary Antivirus Signature Optimization

Eliana Giovannitti
Politecnico di Torino
Torino, Italy
eliana.giovannitti@polito.it

Luca Mannella
LINKS Foundation
Torino, Italy
luca.mannella@linksfoundation.com

Andrea Marcelli
Hispasec Sistemas
Málaga, Spain
amarcelli@hispasec.com

Giovanni Squillero
Politecnico di Torino
Torino, Italy
giovanni.squillero@polito.it

***Abstract*—This work presents an automatic methodology able to improve machine-generated signatures for Android Malware detection. The technique relies on a population-less evolutionary algorithm and uses an unorthodox fitness function that incorporates unsystematic human experts knowledge in the form of a set of rules of thumb. The proposed optimization algorithm does not require to rank the individuals, as exploiting experts knowledge, the resulting population of candidate solutions is not a totally ordered set any more. Experimental results show that the resulting signatures are of good quality and more accurate than the original ones, lowering both false positives and negatives.**

***Index Terms*—Android Malware, Automatic Signature Optimization, Estimation of Probability Evolutionary Algorithms**

I. Introduction

The Android ecosystem offers an open market model, where millions of applications are downloaded by users every day. However, such popularity comes with a price: new malware are developed and spread every day by malicious actors. The prompt identification of new malicious applications is a critical issue and still an open problem. Although the official Google Play store applies a strict review process on new applications to confirm their compliance with Google policies [1], the same procedure is not applied on other third-party app-stores, which accidentally offer a channel for malware propagation.

One of the longest running practice for Android malware developers is to repack popular applications from Google Play by adding malicious features and then redistribute them to third-party app-stores [2]. Indeed, by exploiting the original app popularity, the malware gains a fast propagation. Moreover, the repackaging process is is simplified by the availability of specific tools [3] [4].

In this scenario, antivirus (AV) software are struggling to keep their signature database up-to-date and AV scanners are suffering from several false negatives detection [5]. It is a challenging task to create high quality signatures able to generalize the matching of new malware variants while avoiding false positives detection. Furthermore this generalization requires a big effort in terms of human experts' time.

Given the industrial interest in the automatic generation of new Android Malware Signatures (AMS), new tools are constantly proposed. One of the latest presented work is YaYaGen (Yet Another Yara Rule Generator) [6], an automated approach to generate YARA signatures to detect Android malware. It combines several heuristic measures with a dynamic greedy optimization algorithm specifically designed to solve a variant of the set covering problem.

One of the limitation that commonly affect automatic signature generation approaches, is that signatures are too specific to the reference malware samples, that is they hardly detect other malware variants too. In other words, automatically generated signatures produce a very low number of false positives, but a considerably higher number of false negatives.

The aim of this work is to optimize automatically-generated AMS by mean of an estimation off probability evolutionary algorithm (EDA), able to include the human experts knowledge in the optimization process. The efficacy of the proposed method has been proved by extensive experimental results, showing the ability of the proposed EDA of generating more generic rules, that lower the false negatives without affecting the global accuracy.

The main contributions presented in this paper are the following:

- The design of an optimization procedure for AMS which combines heuristics and evolutionary algorithms.
- The usage of expert knowledge as a set of generic *hints* that do not need to be globally coherent.
- The implementation of the proposed methodology as an extension of the original *YaYaGen* software.

The rest of the paper is organized as follows. Section II surveys the signature generation background, while Section III introduces the proposed workflow to generate and optimize a signature. Section IV presents the proposed methodology, and Section V presents the experimental setup and the results. Finally, Section VI concludes the paper.

II. Related Work

A. Automatic Anti-virus Rule Generation

In the early days of AV technology, a simple hash value of an application was enough to detect a malicious software. However, every modification, as tiny as one byte, was enough to escape the detection. Today's signatures are pattern-matching rules commonly defined on static or dynamic properties of the applications and still represent the most reliable (i.e., with the lowest rate of false positives) antivirus technology.

The automatic generation of network signatures has been explored in various previous work [7]–[11] where most of these studies focus mainly on worm fingerprinting.

Perdisci et al. [12] tackles the problem of automatically generate network signatures for cluster centroids, with the aim

of deploying them into an IDS at the edge of a network in order to detect malicious HTTP traffic. In order to avoid false positives detections, generated signatures are checked with a large dump of benign network traffic, discarding those that produce unwanted matches.

AndroSimilar [13] and DroidAnalytics [14] tackle the problem of generating malware signatures specifically for Android malware. The former uses a statistical approach to generate variable length signatures, while the latter relies on a opcode level analysis to extract the API call tracing information. However, detailed experiments show that both approaches are affected by a high false positives rate.

A part from the academic research, a series of tools have been proposed during the years from the industry too. The majority of them address the signature generation for the Windows malware, however the approaches are mostly general, and can be easily extended to the generation of AMS too. Among the others, yarGen [15], YaraGenerator [16], Yabin [17] and BASS [18] are the most well known.

yarGen generates signatures by extracting opcodes and strings, and after a white-list filtering, it uses an heuristic mechanism to select the most effective ones. On the other hand, YaraGenerator simply looks for common strings among the provided samples to generate a YARA rule. Differently, Yabin extracts opcodes from rare functions in the binary, and its effectiveness relies on a comprehensive dataset of goodware functions. Finally, BASS applies a more complex approach where samples are initially clustered according to their similarity, then it produces a ClamAV signature using the Longest-Common-Subsequences (LCS) algorithm.

B. Estimation of Distribution Evolutionary Algorithm

In canonical *evolutionary algorithms* (EAs) [19], an *individual* encodes a candidate solution and the set of all individuals that have a role in the evolutionary process is called *population*. In *estimation of distribution algorithms* (EDAs) [20], on the contrary, candidate solutions are not explicitly stored, but the population contains the distributions, and possibly the relationships, of the variables.

The selfish gene algorithm (SG) is an EDA proposed more than two decades ago [21], it was inspired by a tough experiment suggested by Richard Dawkins in his celebrated book [22]. It exploits a univariate discrete probabilistic model—a vector that stores the marginal probabilities of allele values for each gene position, independently of other gene positiions—and is akin to other algorithms, such as the equilibrium genetic algorithm (EGA) [23], the population-based incremental learning (PBIL) [24], the univariate marginal distribution algorithm (UMDA) [25], and remarkably similar to the compact genetic algorithm (cGA) published the same year [26].

Like the cGA, the SG algorithm iteratively samples two solutions at a time and conducts a competition between them. It then updates the probabilistic model by rewarding the allele values contained in the winner and penalizing the allele values contained in the loser of the competition. The strength of the reward/penalty for each gene position i is determined by a parameter ϵ_i, which usually depends on the number of different alleles that can occupy the i-th locus.

The original SG was quite simple to implement and efficient in finding optima, yet far more robust than pure hill climbing. It was exploited by practitioners in some real-world applications, such as CAD problems [27], by scholars for various test benches [28], and few new approaches derived from it [29]. In 1999 it was enhanced to tackle highly deceptive functions at the expense of a significant loss in performance [30].

III. Signature Generation Workflow

YaYaGen is a tool for the automatic generation of malware signatures. It relies on the use of report files obtained from the analysis of the APK (Android PaKage). Given the set of reports generated by malwares belonging to a same family the tool is able to automatically create a signature of the whole malware family. The signature generation process makes the intersection of all the report, so the created signature contains all the common features of the malware family members. The drawback is that the resulting signature could be over-specific and unable to match new malwares of the same family. For this reason it is essential to add an optimization phase that avoids the creation of such over-specific signatures.

The signature generation workflow used in this paper is a common strategy used in an industrial setting; it is composed by the following phases (Figure 1):

1) *Data Analysis*: a set of Android application is analyzed to extract common characteristics relevant to the clustering process.
2) *Clustering*: the applications are grouped to facilitate subsequent steps. This phase is optional and depends on the applications number and on how they have been collected. Moreover it may be either guided by a human expert or completely unsupervised.
3) *Signature Generation*: the information extracted from the Android applications is converted in a working signature.
4) *Signature Optimization*: once generated the signature gets a score. If the score of the signature is beyond a fixed threshold, the signature is optimized using the methodology of the proposed framework.
5) *Testing*: the effectiveness of the final optimized signature is then evaluated on a large Android applications dataset.

The original framework, without the new evolutionary optimization phase that is presented here, is described in [6] and is in use on *Koodous*[1], the mobile AV platform from Hispasec, since January 2018.

A. Score of the generated signatures

High quality signatures require an optimal balance between generality and specificity. To find this balance is one of the main challenges in automatic signature generation. When a signature is too general, it can generate many false positives.

[1]https://koodous.com/

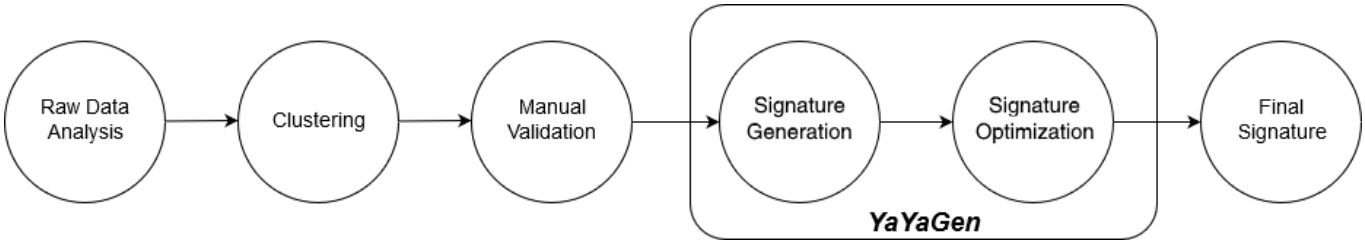

Fig. 1. Workflow of signatures generation.

On the other hand, a very specific one is unable to detect elements with small differences inside the same family.

To rank the discriminating power of a signatures we introduce a signature *score*. Since a signature is composed of one or more attributes, we can assign a score to each attribute. The score is given by a proportional rule: the greater the attribute discriminating power, the highest the score. Its value is computed by an empirical approach combined with the *Simplex Algorithm* [31].

A signature detection score is given by the type and the number of its attributes; this score is computed according to the following formula:

$$S^{\sigma} = \sum_{i=0}^{n} A_i^{\sigma}$$

where

S^{σ} = the score of the signature S
n = the number of attributes inside S
A_i = the i-th attribute of the signature, that is $S = \{A_0, A_1, ..., A_n\}$
A_i^{σ} = the score of the i-th attribute of the signature.

B. Signature Representation in the Disjunctive Normal Form

The attribute set of a malware family is the intersection of the attributes of its members, so it possible to express the signature of the family in a disjunctive normal form.

The *disjunctive normal form* (DNF) is a standardization of a logical formula which is a disjunction of conjunctive clauses; it is simply called "OR of ANDs". A DNF formula is made up of clauses which, in turn, contain one or more literals.

Aided by the fact that every signature is composed of one or more attributes extracted from an application analysis report (e.g., the access permission to a specific resource, the name of an activity, a URL accessed at execution time, etc...), it is quite easy to associate an attribute to a literal and an AMS to a clause. The final signature is then the disjunction of several AMS.

Based on the above, a signature (S) can be expressed as:

$$S = \bigvee_{i=1}^{n} (\bigwedge_{k=1}^{m_i} A_{i,k})$$

where

S = the final signature generated, a disjunction of several AMS
$A_{i,k}$ = the k-th attribute belonging to the i-th AMS
n = the number of generated AMS
m_i = the number of attributes inside the i-th AMS.

The score of the resulting signature can be easily calculated starting from the score of single attributes:

$$S^{\sigma} = \min_{i=1}^{n} (\sum_{k=1}^{m_i} A_{i,k}^{\sigma})$$

IV. Proposed Framework

Malware signatures can be naively defined by the strings that only appear in the malware samples not in the legitimate program. Quite differently, signatures defined in the proposed approach are accurate, descriptive and based on structural properties derived from static and dynamic analyses.

After the identification of a good set of clauses that matches all the target applications, the goal of this framework is to optimize every generated signature. To do this, it is necessary to select the proper literals that make up each clause. The aim is achieved using specific heuristic strategies based both on rules referable to expert knowledge and on the introduction of a population-less EA.

A. Double Threshold mechanism

The score of a signature is defined as the sum of the weight of its literals (Section III-A) and is inversely related to the signature generality. The lower the score, the more a signature is able to generalize, but also the more is prone to unwanted detections. On the other hand, a high score means a more specific signature and a lower number of false positives results.

In order to find a good balance and to build effective and well optimized signatures, we make use of two fixed thresholds (T_{min}, T_{max}). T_{min} represent the value below which a signature is considered too generic, T_{max} the limit for overly-specific signatures.

In the generation phase, a signature with a score smaller than T_{min} is considered invalid (i.e., too generic). In the extreme case where it's not possible to generate any "valid signature" (i.e., a signature with a score greater than the thresholds) the problem is proposed to the analyst.

In the optimization phase, a generated signature with a score greater than the upper threshold (T_{max}) is considered too

specific and it is managed by the optimization process that tries to reduce the score.

Experts' knowledge helped in establishing that $T_{min} = 400$ and $T_{max} = 650$.

A *Basic Optimizer* (BO) implemented this *double threshold mechanism*. The BO pseudo-randomly removes AMS attributes until the signature score is lower than T_{max}. At the same time it's continuously checked that the target signature still has a score grater than the lower threshold (T_{min}), so as to avoid the creation of a too generic signature.

B. The Evolutionary Optimization Phase

We developed a more efficient optimization phase that overcomes the BO performances based on the SG. Moreover the algorithm can be fully automated and can substitute a human expert.

The technique relies on an population-less estimation of probability evolutionary algorithm, with an unorthodox fitness function given by unsystematic human experts' knowledge coded as a set of rules. In the current application, the genome is the signature to optimize and loci are the attributes of the AMS. Each locus may contain two alleles (*true* and *false*) specifying whether the attribute is used in the final signature or not.

In more details, to compare two candidate solutions the following metrics are considered, in order of priority:

1) the number of Android application reports correctly detected by the candidate solution among the ones used to generate the candidate AMS. The main aim is to achieve 100% coverage;
2) a set of manually-defined heuristic rules, gathered from human experts;
3) the score of the candidate optimized AMS;
4) the number of attributes contained inside the candidate AMS.

The first value is to be maximized while the third and forth minimized.

The algorithm starts comparing the number of matches of two candidates. If a signature matches more reports than the other one (and so, probably even more than the AMS it was generated by) it will be considered as the most powerful one. Otherwise, if the number of matches is equal the algorithm continues using the heuristic rules. If the number of reports and the heuristic rules are still not sufficient to establish which signature is better the algotihm evluate the score and the number of attributes choosing the candidate signature with the lower values.

The score of the signatures are also constrained between two thresholds ($T_{\min} < S^{\sigma} < T_{\max}$). A candidate solution receives a huge penalty if the signature becomes too general going under $T_{\min}$, and a smaller penalty if the signature becomes too specific going beyond $T_{\max}$. Penalties are different because, in the malware detection environment, it is preferable to have too specific signatures and miss some samples rather than detect false positives.

Human experts have been interviewed, providing a set of *rules of thumb* that can be used to determine if a signature is better than another one:

- having URL is better than not having it;
- not having SSL is better than having it;
- more categories the better;
- more functionalities the better;

To make empirical rules even more similar to the human decision-making process we introduce a tolerance in the comparisons implemented by the last two rules. For example, signature A is assumed better than signature B if A is greater than B increased by 10%.

While the set of heuristic rules gathered from human experts, has been demonstrated able to simulate human way-of-thinking, such set of rules does not specify a totally ordered set of individuals. That is, if a signature i_a is preferable to i_b, and i_b is preferable to i_c, this doesn't imply that i_a is preferable to i_c:

$$(i_a \geq i_b) \wedge (i_b \geq i_c) \not\Rightarrow \ i_a \geq i_c$$

For example: Let's assume that the threshold on the functionalities rule is 3 and the threshold on the categories rule is 10. If

$$\begin{aligned} S1 &= 5categories; \\ S2 &= SSL + 10categories + 2functionalities; \\ S3 &= SSL + 15categories + 4functionalities; \end{aligned}$$

We obtain

$$S1 > S2; \quad S2 = S3; \quad S1 < S3.$$

If two candidate solutions are both composed only by URL or both have the SSL functionality, it is not possible to establish which AMS is better.

C. Archive

In traditional EDA, when two candidate solutions are generated and compared, they have to compete with a set of solutions that are considered equally good. This pool of individuals is called *archive* [32]. All the individuals contained inside the archive are candidates to be returned as the final solution. If the fitness value of a candidate is greater or equal to all the solutions included in the archive, that candidate solution becomes part of the archive. To make this mechanism work it is necessary to have an absolute order among the solutions.

By contrast, in the proposed framework the transitive property is lost, so is the absolute order. This is due to the introduction of the heuristic rules described in Section II-A. It is therefore necessary to change the way the optimizer selects the individuals to be kept in the archive.

We propose a tournament-based approach. After every comparison, each individual inside the archive receives:

- 3 points, if it is better than the other one;

- 0 points, if it is worst than the other one;
- 1 point, if it is not possible to establish which individual is better.

This mechanism is executed in a round-trip way, so that each pair of individuals is compared twice. At the end, the individual that is stored in the archive is the one with the highest score. If there is more than one individual with the highest score, then they are all kept in the archive.

V. Case of Study

A. Experimental Setup

As a case study we used a dataset of 1.5 million Android applications collected over the year 2016. The dataset is up-to-date and heterogeneous in the set of attack vectors it represents: in order to have the same ratio between detected and undetected applications as in Koodous, we sampled a subset of 1 million apps. As a result, the dataset under analysis is composed by 65% undetected applications, 31% detected by signatures, and 4% detected through triage only.

Note, all the tests performed during this research activity were executed on a server equipped with a 4-core Intel i5 processor (i5-2500 CPU @3.30 GHz), 8GB of RAM, and running Ubuntu 16.04.3 LTS.

B. Experimental Results

Experimental results show that the new version of YaYaGen gains the ability to generate more accurate rules, lowering both false positives and negatives. The following table compares the score of same signatures generated by the original version of YaYaGen (*Not Optimized* column) with the score of signatures generated by the optimized versions (*Basic Optimizer* and *SGX Optimizer* columns). Since the score of a rule is directly related to the accuracy of the malware detection, the results show that both the optimizers succeed in reducing the score within the rage ($T_{min} = 400$, $T_{max} = 650$) that was defined as optimal. Moreover, it is noted that the SGX-RO reaches the best results using a combination of sub-optimal literals defined through heuristic rules[2].

After several experiments we observed that the *steady state mechanism* is able to substantially improve the execution speed unless the initial configuration was to much distant from an optimal solution.

VI. Conclusions and Future Works

The main idea behind this paper is to improve the capabilities of generating effective Android Malware Signature using *heuristic and evolutionary techniques*.

A heuristic mechanisms was introduced to establish the goodness of the generated signatures: a value was assigned to each possible literal and, thanks to this, it was possible to estimate the goodness of each generated signature. Two thresholds (T_{min} and T_{max}) were defined to establish if a signature has to be regarded as to much generic or to much specific.

[2]Results are computed on the average of 10 independent tests

To improve the generated rules, an *optimization phase* was developed inside an already developed tool (YaYaGen). Experimental results show that the signature optimization phase gives to YaYaGen the ability to generate more accurate rules, lowering both false positives and negatives.

Our proposed approach has gained great interest within the malware research team of Koodous, for this reason and after a deep testing phase, it will be integrated inside *Koodous Brain*. Koodous Brain is an artificial intelligence platform developed to assist Android malware detection in the *Koodous project*, an open community antivirus from Hispasec Sistemas.

Even if this research activity has reached several satisfactory results, it is far from being considered concluded. YaYaGen has already gained new instruments and potentialities but has to be expanded and empowered in the following months to give the malware analysts a powerful tool to assist their research activities.

Possible improvements on this research can be focused on: make a distinction between IP and URL, improve the Automatic Initial Probability Computation, develop a new heuristic optimizer.

The first improvement was proposed during the testing phase. In this period it was observed that several URLs are sometimes matched on the same IP address. Implement a mechanism that takes advantage of this peculiarity could be helpful and very useful.

It was also noticed that the initial probability of 0 is a key element for the execution time and for the quality of the experimental results. Finding a precise mathematical function able to take advantage of this characteristic could further improve the computational time and the achieved results .

Finally, heuristic strategies allows to create rules similar to those that a human expert will write. Developing a third optimizer, entirely based on heuristic strategies, could give to YaYaGen a new powerful tool to optimize its signatures.

ACKNOWLEDGMENT

This article is based upon work from COST Action CA15140 "Improving Applicability of Nature-Inspired Optimisation by Joining Theory and Practice" (ImAppNIO) supported by European Cooperation in Science and Technology.

References

[1] "Google_android_security_2016_report_final.pdf," https://source.android.com/security/reports/Google_Android_Security_2016_Report_Final.pdf, Mar 2017, (Accessed on 11/12/2017).

[2] F. Zhang, H. Huang, S. Zhu, D. Wu, and P. Liu, "ViewDroid: Towards obfuscation-resilient mobile application repackaging detection," in *Proceedings of the 2014 ACM conference on Security and privacy in wireless & mobile networks*. ACM, 2014, pp. 25–36.

[3] R. Winsniewski, "Android–apktool: A tool for reverse engineering Android apk files," 2012.

[4] J. Freke, "Smali, an assembler/disassembler for Android's dex format," *Google Project Hosting [online] http://code. google. com/p/smali*, 2013.

[5] J. Oberheide, E. Cooke, and F. Jahanian, "Cloudav: N-version antivirus in the network cloud." in *USENIX Security Symposium*, 2008, pp. 91–106.

[6] A. Atzeni, F. Díaz, A. Marcelli, A. Sánchez, G. Squillero, and A. Tonda, "Countering android malware: A scalable semi-supervised approach for family-signature generation," *IEEE Access*, vol. 6, pp. 59 540–59 556, 2018.

Cluster	Algorithm	Clause	Match	Not Optimized		Basic Optimizer		SGX Optimizer	
				Literals	Score	Literals	Score	Literals	Score
Cluster10	Greedy		4/4	260	17614	9.00	625.00	11.50	401.70
	Clot		4/4	260	17614	9.20	597.00	12.60	401.80
Cluster20	Greedy	Clause 1	3/4	64	2073	20.20	612.30	39.20	555.00
		Clause 2	1/4	59	1907	19.80	620.40	39.30	550.50
	Clot	Clause 1	1/4	59	1907	20.80	615.40	38.30	536.20
		Clause 2	3/4	64	2073	18.20	620.50	40.90	583.90

TABLE I
RESULTS SAMPLES TAKEN FROM OPTIMIZERS EXECUTION

[7] Z. Li, M. Sanghi, Y. Chen, M.-Y. Kao, and B. Chavez, "Hamsa: Fast signature generation for zero-day polymorphic worms with provable attack resilience," in *Security and Privacy, 2006 IEEE Symposium on*. IEEE, 2006, pp. 15–pp.

[8] J. Newsome, B. Karp, and D. Song, "Polygraph: Automatically generating signatures for polymorphic worms," in *Security and privacy, 2005 IEEE symposium on*. IEEE, 2005, pp. 226–241.

[9] S. Singh, C. Estan, G. Varghese, and S. Savage, "Automated worm fingerprinting." in *OSDI*, vol. 4, 2004, pp. 4–4.

[10] Y. Xie, F. Yu, K. Achan, R. Panigrahy, G. Hulten, and I. Osipkov, "Spamming botnets: signatures and characteristics," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, pp. 171–182, 2008.

[11] V. Yegneswaran, J. T. Giffin, P. Barford, and S. Jha, "An architecture for generating semantic aware signatures." in *USENIX Security Symposium*, 2005, pp. 97–112.

[12] R. Perdisci, W. Lee, and N. Feamster, "Behavioral clustering of http-based malware and signature generation using malicious network traces." in *NSDI*, vol. 10, 2010, p. 14.

[13] P. Faruki, V. Ganmoor, V. Laxmi, M. S. Gaur, and A. Bharmal, "Androsimilar: robust statistical feature signature for Android malware detection," in *Proceedings of the 6th International Conference on Security of Information and Networks*. ACM, 2013, pp. 152–159.

[14] M. Zheng, M. Sun, and J. C. Lui, "Droid analytics: A signature based analytic system to collect, extract, analyze and associate android malware," in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference on*. IEEE, 2013, pp. 163–171.

[15] R. F., "yargen," https://github.com/Neo23x0/yarGen, 2018.

[16] C. C., "yaragenerator," https://github.com/Xen0ph0n/YaraGenerator, 2013.

[17] A. OTX, "Yabin," https://github.com/AlienVault-OTX/yabin, 2018.

[18] "Bass - bass automated signature synthesizer," https://github.com/Cisco-Talos/BASS.

[19] A. Eiben and J. Smith, *Introduction to Evolutionary Computing*. Springer Berlin Heidelberg, 2015. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-44874-8

[20] M. Pelikan, M. W. Hauschild, and F. G. Lobo, "Estimation of distribution algorithms," in *Springer Handbook of Computational Intelligence*. Springer, 2015, pp. 899–928.

[21] F. Corno, M. S. Reorda, and G. Squillero, "The selfish gene algorithm: a new evolutionary optimization strategy," in *Proceedings of the 1998 ACM symposium on Applied Computing*. ACM, 1998, pp. 349–355.

[22] R. Dawkins, "The selfish gene," *New York*, 1976.

[23] A. Juels, S. Baluja, and A. Sinclair, "The equilibrium genetic algorithm and the role of crossover," *Unpublished manuscript*, 1993.

[24] S. Baluja, "Population-based incremental learning. A method for integrating genetic search based function optimization and competitive learning," Carnegie-Mellon Univ Pittsburgh Pa Dept Of Computer Science, Tech. Rep., 1994.

[25] H. Mühlenbein and G. Paass, "From recombination of genes to the estimation of distributions i. binary parameters," in *International conference on parallel problem solving from nature*. Springer, 1996, pp. 178–187.

[26] G. R. Harik, F. G. Lobo, and D. E. Goldberg, "The compact genetic algorithm," *IEEE transactions on evolutionary computation*, vol. 3, no. 4, pp. 287–297, 1999.

[27] J. Zhang, M. L. Bushnell, and V. D. Agrawal, "On random pattern generation with the selfish gene algorithm for testing digital sequential circuits," in *Test Conference, 2004. Proceedings. ITC 2004. International*. IEEE, 2004, pp. 617–626.

[28] R. Tavares, A. Teófilo, P. Silva, and A. C. Rosa, "Infected genes evolutionary algorithm," in *Proceedings of the 1999 ACM symposium on Applied computing*. ACM, 1999, pp. 333–338.

[29] N. E. A. Khalid, N. M. Ariff, S. Yahya, and N. M. Noor, "A review of bio-inspired algorithms as image processing techniques," in *International Conference on Software Engineering and Computer Systems*. Springer, 2011, pp. 660–673.

[30] F. Corno, M. Sonza Reorda, and G. Squillero, "Optimizing deceptive functions with the SG-clans algorithm," in *Congress on Evolutionary Computation*. IEEE, 1999, pp. 2190–2195.

[31] A. L. Brearley, G. Mitra, and H. P. Williams, "Analysis of mathematical programming problems prior to applying the simplex algorithm," *Mathematical Programming*, vol. 8, no. 1, pp. 54–83, Dec 1975. [Online]. Available: https://doi.org/10.1007/BF01580428

[32] L. Wang, C. Fang, C.-D. Mu, and M. Liu, "A pareto-archived estimation-of-distribution algorithm for multiobjective resource-constrained project scheduling problem," *IEEE Transactions on Engineering Management*, vol. 60, no. 3, pp. 617–626, 2013.