Taylor & Francis
Taylor & Francis Group

Journal of Experimental & Theoretical Artificial Intelligence

ISSN: 0952-813X (Print) 1362-3079 (Online) Journal homepage: https://www.tandfonline.com/loi/teta20

Can genetic algorithms help virus writers reshape their creations and avoid detection?

Iyad Abu Doush & Mohammed I. Al-Saleh

To cite this article: Iyad Abu Doush & Mohammed I. Al-Saleh (2017) Can genetic algorithms help virus writers reshape their creations and avoid detection?, Journal of Experimental & Theoretical Artificial Intelligence, 29:6, 1297-1310, DOI: 10.1080/0952813X.2017.1354078

To link to this article: https://doi.org/10.1080/0952813X.2017.1354078

JOURNAL OF EXPERIMENTAL & THEORETICAL ARTIFICIAL INTELLIGENCE, 2017
VOL. 29, NO. 6, 1297–1310
https://doi.org/10.1080/0952813X.2017.1354078

Taylor & Francis
Taylor & Francis Group

Can genetic algorithms help virus writers reshape their creations and avoid detection?

Iyad Abu Doush[a,b] and Mohammed I. Al-Saleh[c]

[a]Computer Science Department, Yarmouk University, Irbid, Jordan; [b]Computer Science and Information Systems Department, American University of Kuwait, Salmiya, Kuwait; [c]Computer Science Department, Jordan University of Science and Technology, Irbid, Jordan

ABSTRACT

Different attack and defence techniques have been evolved over time as actions and reactions between black-hat and white-hat communities. Encryption, polymorphism, metamorphism and obfuscation are among the techniques used by the attackers to bypass security controls. On the other hand, pattern matching, algorithmic scanning, emulation and heuristic are used by the defence team. The Antivirus (AV) is a vital security control that is used against a variety of threats. The AV mainly scans data against its database of virus signatures. Basically, it claims a virus if a match is found. This paper seeks to find the minimal possible changes that can be made on the virus so that it will appear normal when scanned by the AV. Brute-force search through all possible changes can be a computationally expensive task. Alternatively, this paper tries to apply a Genetic Algorithm in solving such a problem. Our proposed algorithm is tested on seven different malware instances. The results show that in all the tested malware instances only a small change in each instance was good enough to bypass the AV.

ARTICLE HISTORY
Received 28 September 2015
Accepted 25 May 2017

KEYWORDS
Antivirus; virus; genetic algorithm; optimisation; evolutionary algorithms; computer security

1. Introduction

Securing systems is one of the most important issues affect the cyber world. Maintaining the confidentiality, integrity and availability of computing systems is the ultimate goal of any security control system. On the other hand, attackers always try to break through security systems. Breaching systems, stealing sensitive information and credit card numbers, spamming, phishing, and extortion are among attackers' actions. They are motivated by money, fame, business, ideology, conflict or fun. In order to break into a system, an attacker needs to find a vulnerability that can be exploited. Vulnerabilities can be technical, such as a badly written software, or non-technical, such as those driven by social engineering. No matter how a system is breached, an attacker will have an unauthorised code that runs inside a victim's machine. To deal with this problem, security engineers have been developing controls to prevent, deter, detect or recover from attacks. Intrusion Detection Systems (IDS), Intrusion Prevention Systems (IPS), firewalls, authentication mechanisms, cryptography and Antiviruses (AVs) are among such controls.

The AV software is the mostly used security control among the end-users community. Computer Security Institute (Richardson, 2011) reported through a survey conducted on a number of security practitioners that 97% of the respondents were using an AV. This concludes how mature, secure, affordable the AV is. Intrinsically, the AV scans files or data against its database of virus signatures. A virus will be quarantined if it matches a signature. A well-known weakness in the AV is the fact

CONTACT Iyad Abu Doush iyad.doush@yu.edu.jo

that it has low-detection rate against new attacks (Baecher, Koetter, Holz, Dornseif, & Freiling, 2006; Mansoori, Welch, & Fu, 2014). This is because the AV maintains signatures of already-known viruses. If a new virus is developed then the AV cannot detect it because it has no signature for it. To approach this problem, the AV uses other techniques such as heuristics and behaviour-based detection (Rieck, Trinius, Willems, & Holz, 2011).

In order to evade the AV detection, virus writers develop ever-evolving techniques to conceal their creations. Such techniques include packing (kind of compression), encryption, polymorphism (many encryption algorithms and keys to encrypt/decrypt a virus) and metamorphism (changing the semantics of the virus) (Szor, 2005). The main goal of these techniques is to create a new appearance for the virus and consequently getting unrecognised by the AV. On the other hand, the defence team reacts to such concealment techniques. Unpacking, decryption, code emulation, algorithmic scanning, X-Ray scanning and heuristic scanning methods are used against the above-mentioned attacks techniques (Szor, 2005). In all cases, however, the AV needs to keep a good balance between security and performance. For example, code emulation in a controlled environment takes much time that end-users might not be willing to bear (Filiol, 2005).

Virus writers are interested in modifying their creations with the least possible effort. They want the modified version not to be recognised by the AV. Making such changes requires knowledge about the parts of the virus that is being used by the AV to detect the virus. A brute-force approach could try changing every possible set of bytes in the virus and report the smallest set (Apap, Honig, Hershkop, Eskin, & Stolfo, 2002; Charlier, Mounji, Swimmer, & Informatik, 1995). However, this approach is computationally expensive because we are looking at the power set of the different permutations of the virus bytes (see Section 3). The novelty of this paper is that it represents the first attempt to figure out the least possible changes that are required so that the virus appears innocent. Our approach uses Genetic Algorithm (GA) to achieve the goal. The GA helps in reaching the minimum number of bytes to be changed in a virus file while maintaining a low computational cost.

This paper is organised as follows. First, we present the related work and then we introduce the problem. This is followed by a background on GA. After that, we explain our methodology in Section 5. Then our results are shown in Section 6. A discussion and future work are in Section 7. Finally, we conclude the paper.

2. Related work

Little work has been conducted in the field of virus scanners. Part of this problem is that most AV products are basically closed-source. That being said, the research community has only a few options. To the best of our knowledge, the only open-source AV is *ClamAV* (Kojm, 2004). ClamAV is mainly used to scan attachments in mail servers. However, some researchers use it to conduct experiments and provide proof-of-concepts systems. We used ClamAV in this study.

AVs are tested against performance and security. The AV performance is very important to the end-user community. Large performance degradation is not acceptable as it makes the system inconvenient to use. Several works have addressed the AV performance enhancement by either software or hardware solutions (Lin, Lin & Lai, 2011; Miretskiy, Das, Wright, & Zadok, 2004; Paul, 2008; Silberstein, 2004; Vasiliadis & Ioannidis, 2010; Uluski, Moffie, & Kaeli, 2005). Low-level reasoning about performance degradation caused by AVs is provided in Al-Saleh, Espinoza, and Crandall (2013).

Maintaining good performance is not the whole story. Consequently, the security of the AV has been studied by several researchers (Al-Saleh & Crandall, 2011; Bishop, Bloomfield, Gashi, & Stankovic, 2011; Christiansen, 2010; Christodorescu & Jha, 2004; Daryabar, Dehghantanha, & Udzir, 2011; Daryabar, Dehghantanha, & Broujerdi, 2012; Filiol, 2006a; Josse, 2006; Lagadec, 2008; Meert & Teirlinckx, 2012; Rad, Masrom, & Ibrahim, 2011; Ramilli & Bishop, 2010; Ramilli, Bishop, & Sun, 2011). For example, Al-Saleh and Crandall (2011) can predict how up-to-date the AV is through a side effect timing channel that is created remotely. In addition, Al-Saleh, AbuHjeela, and Al-Sharif (2014) show that some AVs can be bypassed under pressure because it cannot manage to react to different threats simultaneously.

Virus writers have developed several techniques over time to evade detection. These techniques include obfuscation, polymorphism, encryption and metamorphism (Aharoni, Peleg, Regev, & Salman, 2016; Biondi, Josse, & Legay, 2016; Blackthorne, Bulazel, Fasano, Biernat, & Yener, 2016; Ersan, Malka, & Kapron, 2016; Tasiopoulos & Katsikas, 2014). Even though pattern matching still the most technique used by the AV to detect malware, some other techniques are utilised by the AV. These techniques include algorithmic scanning Szor (2005), emulation Szor (2005) and heuristics Eisner (1997). Apart from the AV security, Al-Saleh (2013) shows that the AV can affect the digital evidence for that it interferes with many system operations.

Identifying the signature that the AV has for a virus (a process called signature extraction) has been approached by both Christodorescu and Jha (2004) and Filiol (2006b). However, they both assume that a single byte always contributes to the signature. In other words, adding a byte to the extracted signature solely depends on that single byte, which is not always the case. Sometimes, as we showed in Section 6, changing one byte alone is not enough to bypass AV. In this work, we do not make any assumptions about that and we want to identify the minimum set of bytes that can be part of a signature.

Heuristic techniques are utilised by AVs in order to detect viruses based on some suspicious symptoms or behaviours (Bazrafshan, Hashemi, Fard, & Hamzeh, 2013; Griffin, Schneider, Hu, & Chiueh, 2009; Schultz, Eskin, Zadok, & Stolfo, 2001; Wang & Wang, 2015). Heuristic AVs keep watching the behaviour of processes, looking for bad behaviours. The AV maintains a Maliciousness Indicator Variable (MIV) for the running processes. Whenever a process has its MIV bypasses a certain threshold then it can be considered a virus. Heuristic techniques are good in that they can detect never-seen-before viruses. However, they might suffer from high false positive rates, which many users are not ready to bear.

Yusoff and Jantan (2011) present malware classification method based on GA to optimise decision tree (DT) called Anti-Malware System (AMS) Classifier. The purpose is to accurately classify unique and new types of malwares. The evaluation results show that AMS classifier accuracy outperforms the DT classifier by 4.5% to 6.5%. The proposed technique is trained on 80% of the data and tested on the remaining 20%.

Kim and Moon (2010) proposed a GA detection technique by analysing dependency graph. They represent the script malware by dependency graph and the detection problem becomes finding the maximum subgraph isomorphism. The relationship between the lines of the semantic code is used to develop the dependency graph. The technique is used to determine if a new script malware contains the obtained subgraph. The purpose is to detect polymorphic malware which changes only the appearance of the script but the core part remains the same. The evaluation results show that the proposed approach outperforms existing AV softwares in detecting viruses accurately.

Edge, Lamont, and Raines (2006) present virus detection algorithm based on artificial immune system genetic algorithm (REALGO). The algorithm generates antibodies randomly and uses virus signatures (antigens) to perform training on antibodies using GA. The purpose is to immunise the computer from viruses and to learn new signatures for unknown viruses. The proposed approach can detect new viruses. As a result, it develops a new retrovirus signature. The results show that REALGO algorithm outperforms other evolutionary techniques in four (out of eight) test functions. The proposed approach can only detect viruses with fixed length signatures.

Finally, Noreen, Murtaza, Shafiq, and Farooq (2009) have shown that GA can be used in evolving malware. However, they first conduct a high-level analysis on the malware code and identify the code semantic. Then, they use GA to evolve the virus and a code generator to convert the evolved virus into a machine code. The proposed framework is used to evolve new Bagle viruses. Our approach is different in that we do not do any semantic analysis and we do not regenerate a virus. We utilise GA to identify the minimal set of bytes that need to be changed in the virus in order to escape AV detection.

$$\begin{bmatrix} X \\ b_2 \\ b_3 \\ \vdots \\ b_N \end{bmatrix}, \quad \begin{bmatrix} b_1 \\ X \\ b_3 \\ \vdots \\ b_N \end{bmatrix}, \quad \begin{bmatrix} b_1 \\ b_2 \\ X \\ \vdots \\ b_N \end{bmatrix}, \quad \ldots, \quad \begin{bmatrix} b_1 \\ b_2 \\ b_2 \\ \vdots \\ X \end{bmatrix}$$

Figure 1. Alternatives of changing a byte of length one (with X symbol).

3. Problem statement

If we have a virus file F that contains N bytes as follows:

$$\mathbf{F} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix} \tag{1}$$

We want to replace some unknown number of bytes in F with a special symbol so that the AV will not be able to recognise the updated F (called F′) as a virus. Our goal, though, is to find F′ with the minimum number of replaced bytes (we call this file F*).

$$\mathbf{F}' = \begin{bmatrix} b'_1 \\ b'_2 \\ \vdots \\ b'_N \end{bmatrix} \tag{2}$$

where each b'_i from the updated file $(b'_1, b'_2, \ldots, b'_N)$ can be equal to either the original b_i or the special symbol X.

A brute-force approach to solve this problem might start by trying to change all the possible sets of bytes. Starting with changing all the sets of bytes with the length of one. Then all the sets of bytes with the length of two, and so on (Apap et al., 2002; Charlier et al., 1995). For example, in the case of making one byte change (with X symbol), we will have the sequence shown in Figure 1.

After every new one byte change, the new generated F′ file is scanned by the AV to check if it is still recognised as a virus or not. For example, if we replace only the second byte, then F′ becomes $\mathbf{b} = (b_1, X, \ldots, b_N)$. If changing one byte is not enough to bypass the AV detection, then the process goes on with changing a set of two bytes, three bytes, and so on. Trying all possible sets, however, is exponential because the length of all possible sets that can be made of a given set **A** of length **N** is 2^N. Such combinations can be formalised as $\sum_{i=1}^{N} \binom{N}{i}$, where $\binom{\mathbf{N}}{i}$ is all the combination of all sets of length ***i***.

4. Background on genetic algorithms

GA was invented by Holland (1975). The purpose of the algorithm is to utilise the evolution techniques such as selection, crossover and mutation in solving optimisation problems (Goldberg, 1989). It randomly generates an initial population, and then computes and saves the fitness for each individual in the population. Evolutionarily, it selects, recombines (crossover) and mutates the current population to come up with a new (hopefully better) population.

GA exploits the current population using selection and crossover operators. It also explores the search space using mutation. This is necessary to prevent getting stuck in the local optima and to

increase the chance of finding the global optima. It repeats these steps until the optimal solution is found or the maximum number of generations is reached. The GA is selected because its operators result in a good balance between exploration and exploitation when compared to other optimisation algorithms (Deep & Thakur, 2007).

Several techniques can be used to select the mating individuals (Al-Betar, Doush, Khader, & Awadallah, 2012). In our work, we used tournament selection scheme, which initially proposed by Goldberg, Deb, and Korb (1989). The tournament selection provides a balancing approach between exploration and exploitation and it gives a better result when compared to other selection techniques (Al-Betar et al., 2012). This selection scheme randomly samples k solutions from the entire solutions stored in the population and then selects the solution with the best fitness from that sample.

The mutation method is an essential operator in GA (Deep & Thakur, 2007). It normally provides a mechanism to explore unvisited regions in the search space. Several types of mutation can be used, but in our proposed algorithm we used random mutation. The random mutation provides a good exploration by looking into several points in the search space (Deep & Thakur, 2007). In this type of mutation, if the mutation rate is met we generate an offspring with a randomly changed genes (Hasan, Doush, Maghayreh, Alkhateeb, & Hamdan, 2014).

Crossover is the operation in which two randomly selected chromosomes are mixed to generate a pair of new chromosomes. This operation is applied with a certain probability in GA.

In our approach, we randomly select one individual from the whole population and another individual from the best 50% as in (Abu Doush, 2012). The first individual is selected using the random selection scheme in which all the solutions stored in the population have an equal probability to be selected. The second individual is selected using a variation of the proportionate (or Roulette wheel) selection scheme proposed by Holand (Holland, 1975). In this case, the second individual is sampled from the 50% best population.

5. Methodology

Our methodology is built around the following questions:

- Can we use GA to find F* rather than trying the brute-force approach?
- Can the AV be misled by applying a minor change on the virus file?

Our proposed solution is represented in the algorithm shown in Figure 2. Simply, we need to use the principles of GA to solve the stated problem.

First, we create a population that comprises a set (*popSize*) of different F′ individuals/chromosomes (*line 5* of Figure 2). Each member in the population is created in three steps:

- Randomly select the number of bytes to be changed in F, call them x.
- Randomly choose the locations of the x bytes in F.
- Replace the bytes at the chosen locations with a special symbol.

Next, we sort (*line 6* of Figure 2) the created population according to the fitness function (*line 4* of Figure 2). The fitness function considers the fittest individual (changed file) to be the one with the fewest number of changed bytes and at the same time the file is not detected to be a virus.

If the modified virus file is F_i', then we can define the function:

$$\text{isVirus}(F_i') \leftarrow \begin{cases} \text{Yes} & \text{the file is a virus} \\ \text{No} & \text{the file is not a virus.} \end{cases}$$

and another function, NumOfBytesChanged(F_i') that returns the number of bytes changed for the given virus file compared with the original virus file.

Set crossoverRate, mutationRate, populationSize, numberOfIterations, specialCharacter.
VF = Read virus File
VF-Size = numberOfBytes(VF)
Fitness function = number of bytes changed in the file and is the file still a virus or not {*The fitness evaluation is based on two metrics: 1-Number of changed bytes, the fewer the better. 2-Is the new file after change a virus or not, the non-virus the better.*}
Produce population randomly of size populationSize of nominated individuals to be the new virus {*Randomly select for the virus file: 1- How many bytes to be changed. 2- The locations of the bytes to be changed.* }
Sort the population according to the fitness function {*Higher rank here is considered as the minimum number of bytes replaced in the virus file and the file become not a virus.*}
for $i = 1, \cdots, iterations$ **do**
    newIndividual = emptyString {*At the end of each iteration we will have a new individual (i.e., new variant of the original virus file).*}
    for $j = 1, \cdots, VFSize$ **do**
        individualOne = Select random individual from the best $populationSize/2$ individuals {*The individuals are sorted based on the fitness function. The best individual appears first.*}
        individualTwo = Select a random individual from all the population
        if $(U(0,1) \leq$ crossoverRate) **then**
            Swap the bytes at location j in both individualOne and individualTwo
            if fitnessFunction(individualOne) $<$ fitnessFunction(individualTwo)) **then**
                newChar = individualOne[j]
            else
                newChar = individualTwo[j]
            end if
        else
            newChar = the j^{th} byte from better fitness individual (i.e., individualOne or individualTwo)
        end if
        if $(U(0,1) \leq$ mutationRate) **then**
            if newChar == specialCharacter **then**
                newChar = VF[j]
            else
                newChar = specialCharacter
            end if
        end if
        newIndividual = newIndividual + newChar
    end for
    Insert newIndividual at the proper place in the sorted population according to its fitness
    Remove the worst individual in the population
end for

Figure 2. The proposed genetic algorithm for finding F*.

In this case, the fitness function can be defined as

$$fittnessValue_i \leftarrow \begin{cases} \text{NumOfBytesChanged}(F_i') * W & \text{w.p. isVirus}(F_i') = \text{Yes} \\ \text{NumOfBytesChanged}(F_i') & \text{w.p. isVirus}(F_i') = \text{No.} \end{cases}$$

where W is a fixed large number (for example, W = 1000).

In our problem the best modified virus file is the one with minimum fitnessValue. The function isVirus() will call the AV to scan the file to identify if it is a virus or not. The complexity of this function depends on the implementation of the called AV. In this paper, we used ClamAV scanning engine. ClamAV uses very efficient string matching techniques such as Boyer–Moore and Aho–Corasick algorithms.

To clarify the concept of the best modified virus file, assume the original virus file F is modified using GA. The result is three modified versions of the virus file F:

- F1, which have one byte modified, but the file it is still detected as a virus.
- F2, which have two bytes modified, but the file is not detected as a virus.
- F3, which have three bytes modified, but the file is not detected as a virus.

In this case, the fitness values are: 1000, 2 and 3 for F1, F2 and F3, respectively. In this example, F2 is the file with the best fitness value. Because, it has the minimum number of bytes changed and the file is not detected as a virus.

After that, we go through a certain number of iterations to create a new individual that is inserted into the population while removing the worst one instead (*lines 31 and 32* of Figure 2). The process of creating a new individual is incremental; one byte is added at a time (*line 9* of Figure 2). To select a byte, we first select two individuals from the population. The first is randomly chosen from the top *popSize/2* individuals and the second individual is randomly chosen from all individuals (*lines 10 and 11* of Figure 2). If a crossover is applied, then the *j*th byte in the two individuals is swapped and the *j*th byte of the fitter will be taken. Otherwise, the *j*th byte of the fitter will be taken directly (*lines 12–20* of Figure 2). If a mutation is applied (*lines 22–27* of Figure 2), then the chosen byte will be tested and flipped. If it equals to the *j*th byte of VF, it is changed to the special symbol. Otherwise, change it to be the *j*th byte of VF.

The proposed algorithm has reduced the computational time when compared with the brute-force techniques (see Section 2), especially if the signature of the virus file is generated from different locations of the file. The algorithm starts with generating a modified version of the virus file by changing random bytes (i.e. generating a population of size (P)). Then we sort these modified versions according to the fitness function. Actually, the mating between the best 50% of individuals in the population will need one time sorting, and after that an insertion for the new generated individual in the correct place is applied. This means that the sorting will be needed only for the first time. The sort time, (S) equals $P * \log (P)$. The AV Scan time, (AV) depends on the AV engine speed. The complexity for this part equals $([P * \log P] + [P * AV])$.

The proposed GA has an outer loop which represents the generations (G), and an inner loop with a number of steps equal to the number of bytes in the virus file (F). The inner loop has five main steps: selection, crossover, mutation, AV scanning of the new individual (AV) and the insertion of the new individual (INS). The insert time equals $\log P$. The overall complexity of the proposed algorithm is $O([G * F * AV * \log (P)] + [P* \log (P) + P * AV])$.

6. Results

Our GA approach is adaptive and interactive in that the virus file is evolved adaptively by continuously scanning new forms of the virus using the AV. The proposed approach is not affected by the AV detection techniques (such as Unpacking, decryption, code emulation, algorithm scanning, etc.) because we utilised the AV scanning engine itself as a blackbox to shape the new virus.

Table 1. The names of the tested malware instances along with their original sizes.

Malware	Original search space (in bytes)
Worm.CodeRed.2	3,818
Worm.Stuxnet-2	30,601
Worm.Bagle.Gen-dll-2	17,408
Zeus-1	512
Trojan.Spyeye-499	46,464
Trojan.Peed-zippwd	30,222
Worm.Koobface-14	13,312

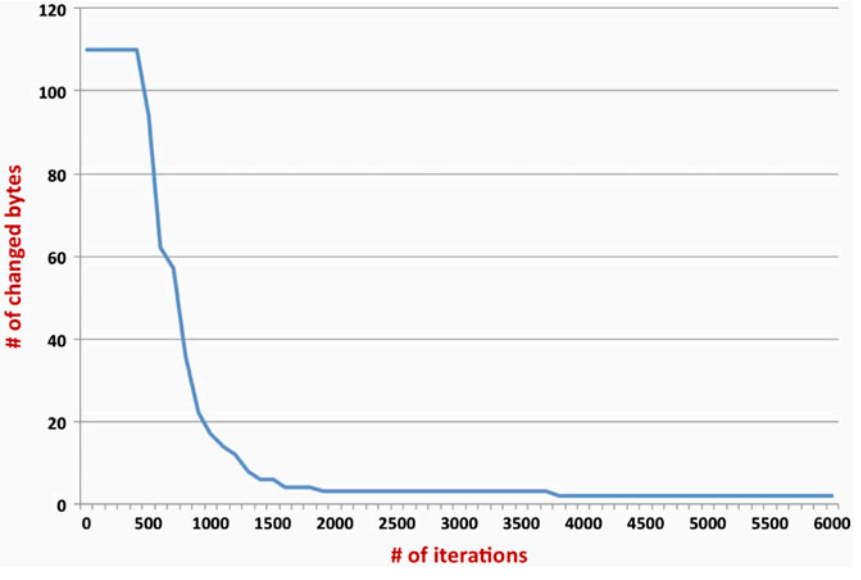

Figure 3. Search space convergence over iterations for Worm.CodeRed.2.

To show the effectiveness of the proposed algorithm, it is tested against several well-known malware instances.[1] Table 1 shows the malware instances we tested. It also shows the sizes of the instances. Finally, the table also shows the number of bytes that needs to be explored to find the minimal set after applying our algorithm. This number will be significantly reduced as our approach utilises GA.

The proposed approach is implemented using Python 2.7. Then we run the application on a computer with Linux Ubuntu 12.04, Intel(R) Core(TM) i5 CPU and 8 GB RAM.

Figure 3 presents the results for Worm.CodeRed.2. The size of the set randomly starts from 110 bytes. This size gradually converges over iterations until it reaches two bytes. The two bytes (which are NOT consecutive) together are required to be changed in order to avoid detection. This finding invalidates the claim of previous studies that depend on the premise that one byte is enough to change a malware instance (see the Related work Section).

Figure 4 shows that the number of bytes that needs to be changed for Worm.Stuxnet-2 starts randomly at 82 bytes. Then, this number decreases to 21 bytes after 100,000 iterations. In the case of Worm.Bagle.Gen-dll-2 (Figure 5), the number of bytes drops quickly from 253 to 26 at 10,000 iterations. Then, it slowly converges to 13 bytes after 100,000 iterations. Only 600 iterations were needed to converge the number of bytes to 1 for Zeus-1 (Figure 6). This low number of iterations is because the size of this malware is small.

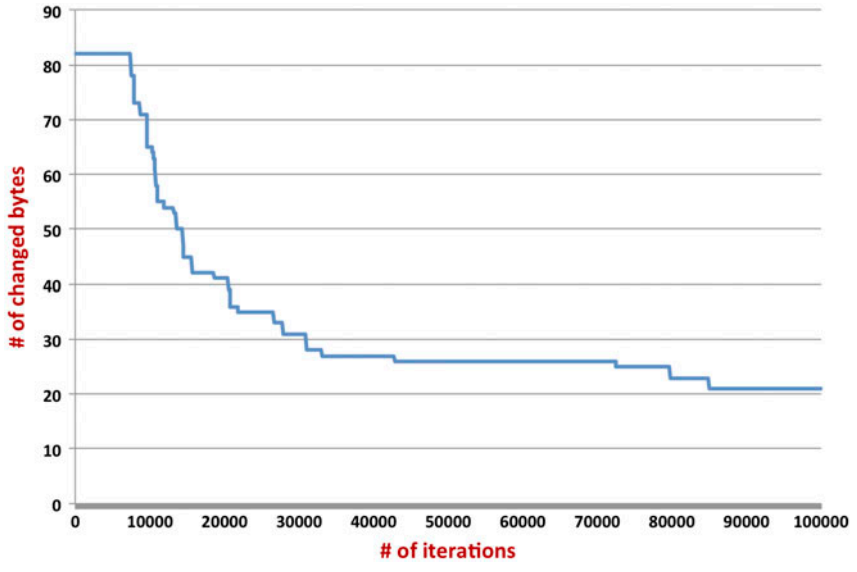

Figure 4. Search space convergence over iterations for Worm.Stuxnet-2.

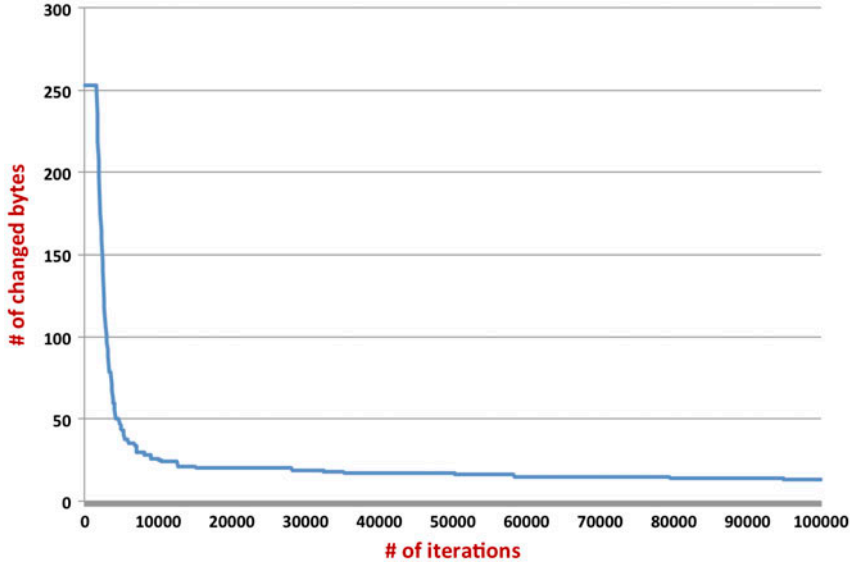

Figure 5. Search space convergence over iterations for Worm.Bagle.Gen-dll-2.

Figures 7 and 8 show similar behaviour as they both start with a high number of changed bytes (765 and 725, respectively). At 100,000 they reach 105 and 42 bytes, respectively. Finally, Figure 9 shows that the number of changed bytes effectively converges from 201 to 2 bytes after 100,000 iterations.

The parameters setting for the proposed algorithm are presented in Table 2. These settings were selected experimentally and they proved to provide good results for the proposed GA. Other parameter settings have been tested and do not provide better results. For example, the settings in Table 2 enables

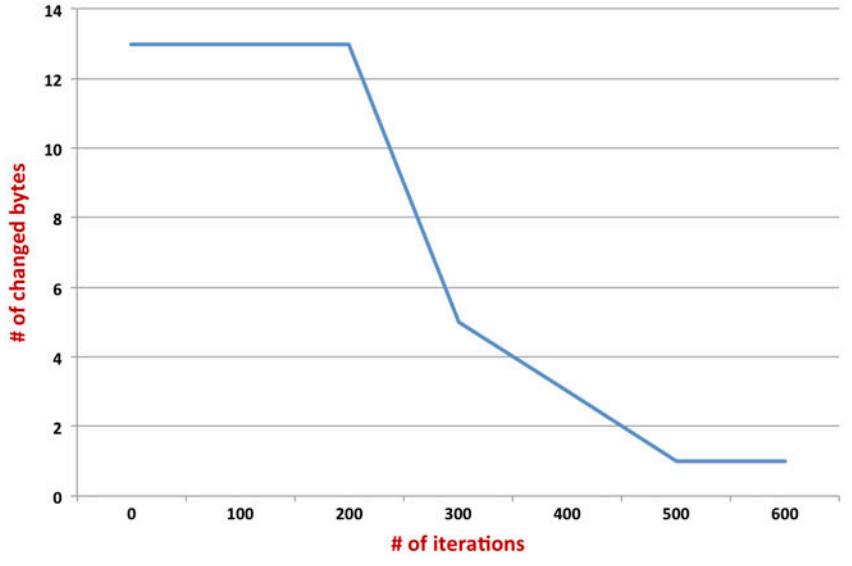

Figure 6. Search space convergence over iterations for Zeus-1.

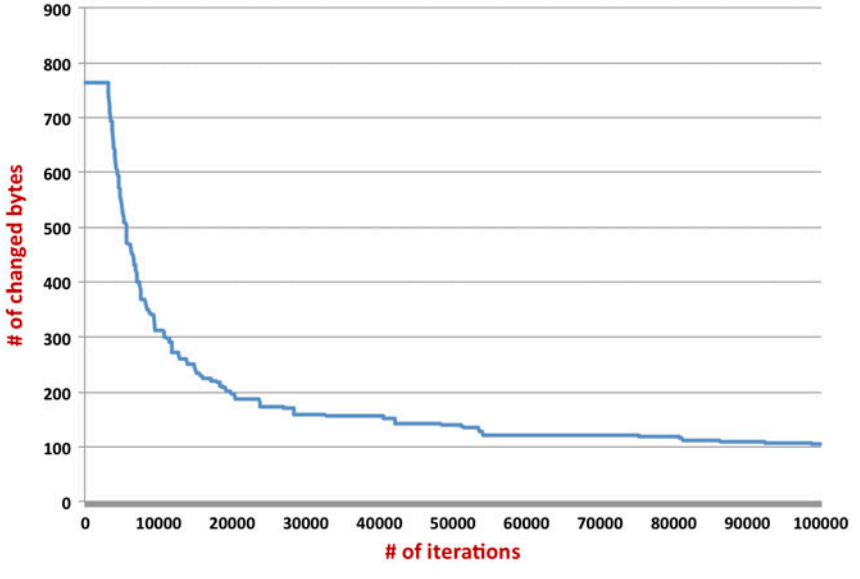

Figure 7. Search space convergence over iterations for Trojan.Spyeye-499.

Table 2. The parameters setting used for the proposed algorithm.

Paramter	Values
Crossover rate	0.7
Mutation rate	0.001
Population size	50
Number of iterations	100000

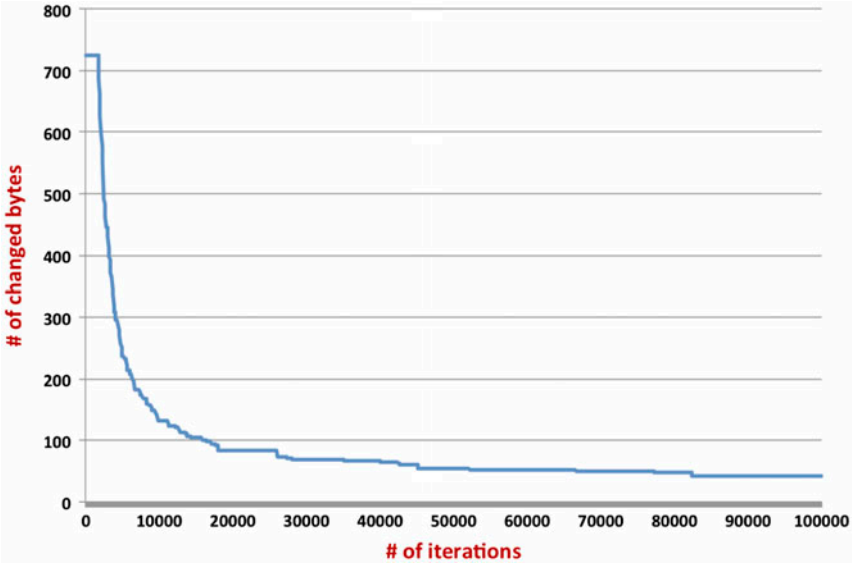

Figure 8. Search space convergence over iterations for Trojan.Peed-zippwd.

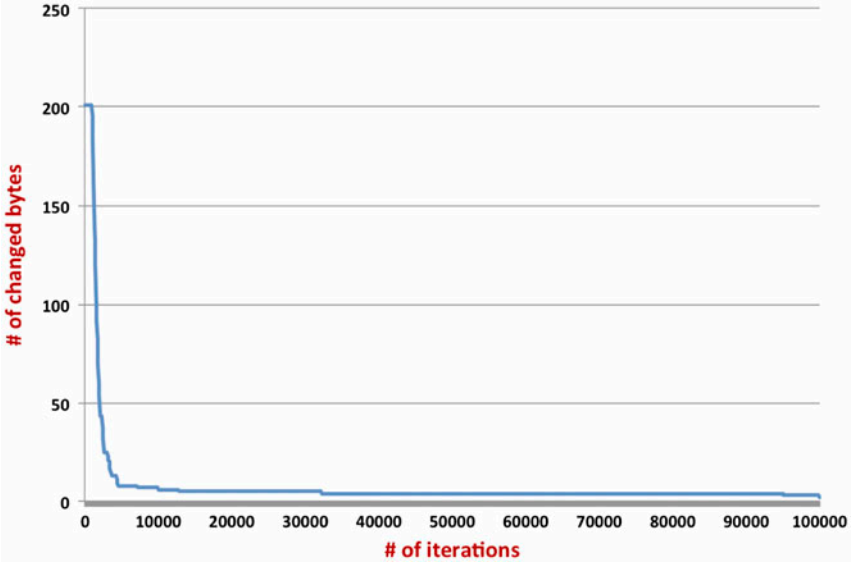

Figure 9. Search space convergence over iterations for Worm.Koobface-14.

the algorithm to change only 2 bytes for the CodeRed virus. On the other hand, Table 3 shows how the results are changed dramatically when changing each GA parameter when applied on the CodeRed virus. In all other settings, we did not reach to the minimum, which is 2. Therefore, it is obvious that our settings are empirically good.

In conclusion, we show that our approach is efficient and effective in converging the search space from the original sizes of the malware instances into much reduced space, which then can be further explored.

Table 3. The results after varying GA parameters for CodeRed.

Paramter	Values	Number of Bytes to be Changed
Crossover rate	0.1	5
Mutation rate	0.01	36
Population size	10	32
Number of iterations	1000	17

To improve the security of the AV, our results suggest choosing a hard-to-extract virus signature by the AV vendors. One of the very important features a signature should have is an almost full malware coverage. In other words, the signatures should be chosen in such a way that makes finding the signature-contributing bytes an unsolvable problem. One technique can be done by creating several sub-signatures from different sets of bytes so that many bytes contribute to the whole virus signature but in a disjunctive manner (i.e. a separate set bytes for each sub-signature).

7. Discussion and future work

This paper introduces a novel approach for finding the minimal number of changes that needs to be applied to a malware file in order to evade detection. A brute-force approach incurs exponential behaviour (Apap et al., 2002; Zhou, Xu, Qi, & Li, 2008). However, this paper utilises GA to approach the problem, which can be solved now in a polynomial time. To the best of our knowledge, no previous research has tackled such problem. The purpose of this work is to prove that it is possible to use GA to make minimal changes to the virus file and evade detection.

In this work, we evaluate the proposed algorithm by changing several well-known malware instances and scan them using ClamAV. Experimenting with more commercial AVs and more malware instances is an interesting future direction.

Different parameters of GA such as mutation and crossover rates, contribute to its precision and convergence. Even though the values of such parameters are not thoroughly explained in this paper, their values were chosen after intensive tests of trial cycles. Explaining the effects of varying the parameters is a future direction.

Malware writers can abuse technology. This paper sheds light on such misuse in which GA is utilised in simplify the process of malware changing. This approach can help malware writers in evolving new malware instances from existing instances. Exactly as in the natural evolution, malware instances can get the best from each other using selection, mutation and crossover. The efficiency of using other techniques can be investigated in the future.

8. Conclusion

The malware writers will continue to improve their techniques. However, being a step ahead can prevent attacks. The end-users heavily rely on the AV software to detect malware. The AV scans data collected from the suspicious file against its own database of virus signatures. It is, though, vulnerable to new attacks. This paper proposes an efficient GA technique to find the minimum number of changes that can be applied to a virus file to evade detection. The minimal change can be typically approached using the standards of GA: selection, crossover and mutation. This paper shows that finding the minimal set of bytes that has to be changed in order to bypass the AV detection is possible. As a result, we recommend the AV vendors to put more efforts in the signature extraction process to avoid the problem presented in this paper.

Note

1. The tested malware can be downloaded from this website: http://openmalware.org/. The naming convention used for malware is the one used by ClamAV.

Disclosure statement

No potential conflict of interest was reported by the authors.

References

Abu Doush, I. (2012). Harmony search with multi-parent crossover for solving ieee-cec2011 competition problems. In T. Huang, Z. Zeng, C. Li, & C. Leung (Eds.), *Neural information processing*. Lecture notes in computer science (Vol. 7666, pp. 108–114). Springer: Berlin Heidelberg.

Aharoni, E., Peleg, R., Regev, S., & Salman, T. (2016). Identifying malicious activities from system execution traces. *IBM Journal of Research and Development, 60*, 1–5.

Al-Betar, M. A., Doush, I. A., Khader, A. T., & Awadallah, M. A. (2012). Novel selection schemes for harmony search. *Applied Mathematics and Computation, 218*, 6095–6117.

Al-Saleh, M., AbuHjeela, F., & Al-Sharif, Z. (2014). Investigating the detection capabilities of antiviruses under concurrent attacks. *International Journal of Information Security*, 1–10.

Al-Saleh, M., Espinoza, A., & Crandall, J. (2013). Antivirus performance characterisation: System-wide view. *Information Security, IET*, *7*, 126–133.

Al-Saleh, M. I. (2013). The impact of the antivirus on the digital evidence. *International Journal of Electronic Security and Digital Forensics, 5*, 229–240.

Al-Saleh, M. I., & Crandall, J. R. (2011). Application-level reconnaissance: Timing channel attacks against antivirus software. *Proceedings of the 4th USENIX Conference on Large-scale Exploits and Emergent Threats, LEET'11* (pp. 9–9). Berkeley, CA: ACM. USENIX Association.

Apap, F., Honig, A., Hershkop, S., Eskin, E., & Stolfo, S., (2002). *Detecting Malicious software by monitoring anomalous windows registry accesses*. Proceedings of the Recent advances in intrusion detection: 5th International Symposium, RAID 2002, Zurich, Oct 16–18, 2002 (pp. 36-53).Berlin Heidelberg: Springer.

Baecher, P., Koetter, M., Holz, T., Dornseif, M., & Freiling, F. (2006). *chapter The Nepenthes platform: An efficient approach to collect Malware*. Proceedings of the Recent Advances in Intrusion Detection: 9th International Symposium, RAID 2006 Hamburg, Sept 20—22, 2006 (pp. 165-184). Berlin Heidelberg: Springer.

Bazrafshan, Z., Hashemi, H., Fard, S. M. H., & Hamzeh, A. (2013). *A survey on heuristic malware detection techniques*. 2013 5th Conference on Information and Knowledge Technology (IKT) (pp. 113–120). Shiraz, Iran: IEEE.

Biondi, F., Josse, S., & Legay, A. (2016). Bypassing malware obfuscation with dynamic synthesis. *ERCIM News, 206*, 37–45.

Bishop, P., Bloomfield, R., Gashi, I., & Stankovic, V. (2011). *Diversity for security: A study with off-the-shelf antivirus engines*. 2011 IEEE 22nd International Symposium on Software Reliability Engineering (ISSRE) (pp. 11–19). IEEE.

Blackthorne, J., Bulazel, A., Fasano, A., Biernat, P., & Yener, B. (2016). *Avleak: Fingerprinting antivirus emulators through black-box testing*. Proceedings of the 10th USENIX Conference on Offensive Technologies (pp. 91–105). USENIX Association.

Charlier, B. L., Mounji, A., Swimmer, M., & Informatik, F. (1995). *Dynamic detection and classification of computer viruses using general behaviour patterns*. Proceedings of the Virus Bulletin Conference.

Christiansen, M. (2010). Bypassing malware defenses? *SANS Institute InfoSec Reading Room* (pp. 3–4).

Christodorescu, M., & Jha, S. (2004). Testing malware detectors. *SIGSOFT International Symposium on Software Testing and Analysis, 29*, 34–44.

Daryabar, F., Dehghantanha, A., & Broujerdi, H. G. (2012). Investigation of malware defence and detection techniques. *International Journal of Digital Information and Wireless Communications (IJDIWC), 1*, 645–650.

Daryabar, F., Dehghantanha, A., & Udzir, N. I. (2011). *Investigation of bypassing malware defences and malware detections*. 2011 7th International Conference on Information Assurance and Security (IAS) (pp. 173–178). Melaka, Malaysia: IEEE.

Deep, K., & Thakur, M. (2007). A new mutation operator for real coded genetic algorithms. *Applied Mathematics and Computation, 193*, 211–230.

Edge, K. S., Lamont, G. B., & Raines, R. A. (2006). *A retrovirus inspired algorithm for virus detection & optimization*. Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, GECCO '06 (pp. 103–110). New York, NY: ACM.

Eisner, J. (1997). Understanding Heuristics: Symantecn++s bloodhound technology symantec. *White paper series*. (Vol. XXXIV). Cupertino, CA: Symantec.

Ersan, E., Malka, L., & Kapron, B. M. (2016). Semantically non-preserving transformations for antivirus evaluation. *International Symposium on Foundations and Practice of Security* (pp. 273–281). Québec, Canada: Springer.

Filiol, E. (2005). Computer viruses: From theory to applications. *Fighting against viruses*. Paris: Springer.

Filiol, E. (2006a). Malware pattern scanning schemes secure against black-box analysis. *Journal in Computer Virology*, *2*, 35–50.

Filiol, E. (2006b). Malware pattern scanning schemes secure against black-box analysis. *Journal in Computer Virology*, *2*, 35–50.

Goldberg, D., Deb, K., & Korb, B. (1989). Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems, 3*, 493–530.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning*. Reading: Addison Wesley.

Griffin, K., Schneider, S., Hu, X., & Chiueh, T.-C. (2009). Automatic generation of string signatures for malware detection. *International workshop on recent advances in intrusion detection* (pp. 101–120). Berlin Heidelberg: Springer.

Hasan, B. H. F., Doush, I. A., Maghayreh, E. A., Alkhateeb, F., & Hamdan, M. (2014). Hybridizing harmony search algorithm with different mutation operators for continuous problems. *Applied Mathematics and Computation, 232*, 1166–1182.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor: The University of Michigan Press.

Josse, S. (2006). How to assess the effectiveness of your anti-virus? *Journal in Computer Virology, 2*, 51–65.

Kim, K., & Moon, B.-R. (2010). *Malware detection based on dependency graph using hybrid genetic algorithm*. Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, GECCO '10 (pp. 1211–1218). New York, NY: ACM.

Kojm, T. (2004). Clamav. Retrieved from http://www.clamav.net

Lagadec, P. (2008). Opendocument and open xml security (openoffice. org and ms office 2007). *Journal in Computer Virology, 4*, 115–125.

Lin, P.-C., Lin, Y.-D., & Lai, Y.-C. (2011). A hybrid algorithm of backward hashing and automaton tracking for virus scanning. *IEEE Transactions on Computers, 60*, 594–601.

Mansoori, M., Welch, I., & Fu, Q. (2014). Yalih, yet another low interaction honeyclient. Proceedings of the Twelfth Australasian Information Security Conference - AISC '14 (Vol. 149, pp. 7–15). Darlinghurst: Australian Computer Society.

Meert, D., & Teirlinckx, N. (2012). Malware, from theory to practice ossec mini-project. Technical report, Vrije Universiteit Brussel.

Miretskiy, Y., Das, A., Wright, C. P., & Zadok, E. (2004). *AVFS: An on-access anti-virus file system*. Proceedings of the 13th USENIX Security Symposium (Security 2004) (pp. 73–88). IEEE. USENIX Association.

Noreen, S., Murtaza, S., Shafiq, M. Z., & Farooq, M. (2009). *Evolvable malware*. Proceedings of the 11th Annual conference on Genetic and evolutionary computation (pp. 1569–1576). Montreal, Québec, Canada: ACM.

Paul, N. R. (2008). Disk-level behavioral malware detection. Ph.D. thesis, University of Virginia, Charlottesville, VA.

Rad, B.B., Masrom, M., & Ibrahim, S. (2011). Evolution of computer virus concealment and anti-virus techniques: A short survey, arXiv preprint arXiv:1104.1070.

Ramilli, M., & Bishop, M. (2010). *Multi-stage delivery of malware*. 2010 5th International Conference on Malicious and Unwanted Software (MALWARE), (pp. 91–97). Nancy, France: IEEE.

Ramilli, M., Bishop, M., & Sun, S. (2011). *Multiprocess malware*. 2011 6th International Conference on Malicious and Unwanted Software (MALWARE), (pp. 8–13). Fajardo, Puerto Rico: IEEE.

Richardson, R. (2011). 15th annual 2010/2011 computer crime and security survey. *Computer Security Institute,* 1–44.

Rieck, K., Trinius, P., Willems, C., & Holz, T. (2011). Automatic analysis of malware behavior using machine learning. *Journal of Computer Security, 19*, 639–668.

Schultz, M. G., Eskin, E., Zadok, F., & Stolfo, S. J. (2001). *Data mining methods for detection of new malicious executables*. Proceedings of the 2001 IEEE Symposium on Security and Privacy, SP 2001 (pp. 38–49), Oakland, CA.

Silberstein, M. (2004). Designing a cam-based coprocessor for boosting performance of antivirus software. Technical report.

Szor, P. (2005). *The art of computer virus research and defense*. Reading: Addison-Wesley Professional.

Tasiopoulos, V. G., & Katsikas, S. K. (2014). *Bypassing antivirus detection with encryption*. Proceedings of the 18th Panhellenic Conference on Informatics (pp. 1–2). Athens, Greece: ACM.

Uluski, D., Moffie, M., & Kaeli, D. (2005). Characterizing antivirus workload execution. *SIGARCH Computer Architecture News, 33*, 90–98.

Vasiliadis, G., & Ioannidis, S. (2010). *Gravity: A massively parallel antivirus engine*. Proceedings of the 13th International Conference on Recent Advances in Intrusion Detection, RAID'10 (pp. 79–96). Berlin Heidelberg: Springer-Verlag.

Wang, P., & Wang, Y.-S. (2015). Malware behavioural detection and vaccine development by using a support vector model classifier. *Journal of Computer and Systems Sciences, 81*, 1012–1026.

Yusoff, M. N., & Jantan, A. (2011). Optimizing Decision Tree in Malware Classification System by using Genetic Algorithm. *International Journal of New Computer Architectures and their Applications (IJNCAA), 3*, 694–713.

Zhou, X., Xu, B., Qi, Y., & Li, J. (2008). *MRSI: A fast pattern matching algorithm for anti-virus applications*. Seventh International Conference on Networking, ICN (Vol. 2008, pp. 256–261), Cancun, Mexico.