I.J. Mathematical Sciences and Computing, **2019, 3, 49-62**
Published Online July 2019 in MECS (http://www.mecs-press.net)
DOI: 10.5815/ijmsc.2019.03.05

Available online at http://www.mecs-press.net/ijmsc

Signature-Based Malware Detection Using Approximate Boyer Moore String Matching Algorithm

A. Ojugo[a,*], A. O. Eboka[b]

[a]*Department of Mathematics/Computer Science, Federal University of Petroleum Resources Effurun, Delta State, Nigeria*
[b]*Department of Computer Education, Federal College of Education (Technical), Asaba, Delta State, Nigeria.*

Received: 22 June 2018; Accepted: 13 June 2019; Published: 08 July 2019

Abstract

Adversaries to any system restlessly continues to sought effective, non-detectable means to aid them successful penetrate secure systems, either for fun or commercial gains. They achieve these feats easily through the use of malware, which keeps on the rise, an ever-growing and corresponding overpopulated malware zoo. As such, information technology industry will continue to encounter via these escapades, both monetary and prestigious losses. Malware by design aims to alter the behaviour of its host by self-replicating its genome or codes unto it. They are quite fascinating in that on execution, some malware change their own structure so that its copies have same functionality but differ in signature and syntax from the original or parent virus. This makes signature detection quite unreliable. Study investigates detection of metamorphic malware attacks using the Boyer Moore algorithm for string-based signature detection scheme.

Index Terms: String Matching Algorithm, Malware Detection, Metamorphics, Pattern Matching, Code Obfuscation.

1. Introduction

Today's society dependence on digitally transmitted data is drastically on the rise along with the growing need to dissuade adversaries from unauthorized access to such data. This has led to advances in security and forensics – with the use of firewalls, gateways and other mechanisms that helps to seek and ensure data security via integrity and privacy. This is quite a herculean task – as its trending direction focuses now more on

* Corresponding author.
E-mail address: ojugo.arnold@fupre.edu.ng

intrusion detection systems to monitor data traffic so as to identify resources misuse, unauthorized use and abuse on networks [37].

Viruses (or malware) are a fascinating creation – in that they are malicious programs that attach themselves to a host or a system, modifies its host's internal structure as it infects the system by attaching its signature (string or code) to its host's internal design, and consequently, alters the behaviour of its host's internal working(s). Thus, malware self-replicates its signature as codes that are attached onto a machine without the user's consent, and spreads by attaching a copy of itself to some part of program file. It attacks system resources and is designed to deliver a payload that aims to corrupt program, delete files, reformat disks, crash network, destroy critical data or embark on other damage to the host machine [12, 48, 7].

As they infect a host, they sometimes modify themselves to include better and possibly, an evolved copy of themselves as copied unto the host – so that as they spread from one host to another – they create variants of the original virus in other infected host systems [4, 5, 6]. The use of Internet for data transfer has become a soft target for their widespread to help wreak havoc, faster globally. Early detection of viruses is thus, imperative to minimize the damage caused [9, 48, 7].

As the field of data security continues to experience quite a huge growth resulting from threats and attacks constantly perpetuated by hacker and crackers via the use of malware – various studies exists that have also successfully implemented a plethora of tools to help combat this menace. The average losses per company due to computer intrusion detection have constantly been on the decrease since 2002 [3, 32]. Even with the daily increase with watchful eyes and systems in place – there is no need to relax, as new attacks are on the rise. These attacks have been observed to be targeted towards criminal and commercial purposes to include (and not limited to) espionage, blackmail, social engineering, fraud etc [48, 9, 31].

1.1. Statement of Problem

Studies have shown that AVs effectiveness has decreased against unknown or zero-day attacks. Evolutionary heuristics and modeling have been successfully applied to many research studies in malware detection with the following problems and caveats:

The changing intent of virus makers keeps the study of virus intrusion consequently on the rise and inevitable.

Independent testing on major virus scanners consistently shows none to yield a 100% detection, irrespective of the heuristics employed. Thus, all scanners must yield an acceptable rate of false-positive and true-negative results, identifying benign files as malware [19-20].

The chaotic and evolvable nature of metamorphics and its range of complications as it provides a backdoor for other crimes – makes it critical and imperative, to early and accurate detect and classify malware intrusion.

Carefully supervised diagnosis can be often mundane and time-wasting of man-hours. This, also in some instances yields inconclusive results for evolving signature strings – all of which will amount to increased false-positive and true-negative results. The understudied string matching algorithm has been successfully employed in effectively classifying malwares as in Section II.

Most evolutionary models employ hill-climbing methods, which are often limited by speed that traps their solution at local maxima – resolved via string-based signature model (good-rule suffix shift) as in Section 3.

Often, the search for optima in most evolutionary model is cumbersome as no one method yield better optima than hybrids. Also, with hybrid, there are the issues of data conflicts and conflicts arising from the statistical methods (combined) adopted. Also, the virus strings have to be encoded unto the adopted model. As resolved in Section II, malware strings are not encoded. Thus, no data pre-processing is required or needed – so that the model also completely is devoid of model overtraining, over-fitting and over-parameterization.

2. Literature Revie

2.1. Types of Malware

Viruses are classified into [13, 14, 8, 33-34]:

1. Simple virus replicates itself if launched. It gains control of the system, attaches copy of itself to another program as it spreads. After which, it transfers control back to host program. It is easily detected via search/scan for a defined sequence of bytes, known as a signature to find the virus,
2. Encrypted Viruses scrambles its signature – making it unrecognizable at its execution. Its decryption routine transfers control to its decrypted virus body so that each time it infects a new program, it makes copy of both the decrypted body and its related decryption routine. It then encrypts a copy and attaches both to a target system. It uses an encryption key to encrypt its body. As the key changes, it scrambles its body so that virus appears different from one infection to another. Such virus is difficult to detect via signature. Thus, antivirus must scan for a constant decryption routine instead.
3. Polymorphics consists of a scrambled body, a mutation engine and decryption routine. Its decryption routine first, gains control of the system by decrypting its scrambled body and mutation engine. It then transfers control to the scrambled body to locate a new file in the host machine to infect via memory mapping. After which, it then copies its body and mutation engine into host machine's RAM, and invokes its mutation engine to randomly generate new decryption routine to decrypt its body with little or no semblance to the previous routine. It then appends this newly encrypted body, a mutation engine and decryption routine to the newly infected file. Thus, the encrypted body and the decryption routine, varies from one infection to another. With no fixed signature and decryption routine, no two infections are alike.
4. Metamorphics avoid detection by rewriting completely, its code each time it infects a new host and its file(s). Its engine accomplishes such features via code obfuscation and metamorphism. And, this in most cases – is 90% of its assembly language codes.

Malware (viruses in general) have 3-modules described as: (a) infect, an internal faculty of malware's design mechanism that depicts how to modify its host system, and replicating or copying itself unto the host, (b) trigger details how and when to deliver the malware's payload, and (c) payload details what damage and the extent of it holds once the host is infected by the malware. Even so, trigger and payload are often optional in some malware such as worms [36, 49].

2.2. Malware Intrusion Detection

Intrusion is any set of action that attempts to compromise integrity, confidentiality, privacy and availability of system (or network) resources. The intruder (or adversary) initiates such intrusive action. Malware often intrudes into a host machine's privacy. Thus, antiviruses are developed as means to monitors such data (over network connections) and determines if it is an intrusive activity via a malware, or not. If an intrusive action is detected, the antivirus performs one of these: (a) alerts such intrusion by flagging access to data by the malware, (b) logs in a message into the system and requires the administrator to analyze immediately, and (c) ends such adversary connections or access to the data, amongst other functions [9].

Threats via such intrusion can be initiated externally or internally [49-51, 37, 9] – resulting in 2-types: (a) external (where an adversary has no authorized access to resources; But, he then attacks a host through the means of malware penetration), and (b) internal (where such adversary has authorized access to resources). However, to ensure data security and integrity, many organizations implement use of antiviruses, firewalls, application gateways as their first step line of defense against intrusive actions.

2.3. Metamorphic Malware

Metamorphic viruses as noted above, changes their internal code structure and appearance while keeping its functionality. It does this via code obfuscation methods as in Fig 1. Its engine reads in a virus executable, locates the codes to be transformed using the method locate_own_code module. Each engine has its transformation rule that defines how a particular opcode or a sequence of opcodes is to be transformed. Decode module extracts the rules by disassembling. Analyze module analyzes current copy of virus and determines what transforms must be applied to generate the next morphed copy. Mutate module performs the actual transformations by replacing an instruction (set) with the other its equivalent code; While, Attach module attaches the mutated or transformed copy to a host [21, 48, 15, 22, 34, 8, 7].

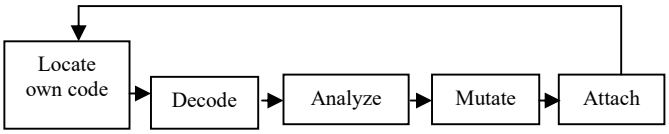

Fig. 1. Distinct Signature of Metamorphic Vırus

A typical metamorphic engine consists: (a) disassemble which disassembles binary codes, (b) shrinker replaces some codes with single equivalent, (c) expander replaces codes with other codes that perform same action, (d) swapper reorders codes by swapping unrelated codes, (e) relocator assigns and relocate relative references such as jumps and call, (f) garbager inserts whitespaces do-nothing codes, and (g) cleaner reverses the actions of garbager by removing whitespaces instructions [24]. Feats of metamorphic engine includes: (i) must handle assembly opcode, (ii) shrinker and swapper to process more than one instruction concurrently, (iii) garbager is moderately used so as not to affect actual instructions, and (iv) swapper analyzes each instruction so as not to affect next instructions' execution [15, 22, 25, 10].

2.4. Metamorphic Code Obfuscation Methods

Metamorphic engine uses code obfuscation to yield morphed copies of original program. Obfuscated code is more difficult to understand and can generate different looking copies of a parent file as it operates on both control flow and data section of a program. Code obfuscation is achieved via [26, 27, 28]:

1. Register Usage Exchange/Rename modifies register data of an instruction without changing the codes itself, which remain constant across all morphed copies. Thus, only the operands changes.
2. Dead Code inserts whitespaces, which do not affect its code execution via a block or single instruction so as to change codes' appearance while retaining functionality.
3. Subroutine Permutation aims to reorder subroutines so that a program of many subroutines can generate (n-1)! varied permutations, whose addition will not affect its functionality as this is not important for its execution.
4. Equivalent Code Substitution replaces instruction with its equivalent instruction (or blocks). A task can be achieved in different ways. Same feat is used in equivalent code substitution.
5. Transposition/Permutation – modifies program execution order only if there is no dependency amongst instructions.
6. Code Reorder inserts unconditional and conditional branch after each instruction (or block), and defines branching instructions to be permuted so as to change the programs' control-flow. Conditional branch is always preceded by a test instruction which always forces the execution of the branching instruction.
7. Subroutine Inline/Outline is similar to dead code insertion in that subroutine call are replaced with its equivalent code as Inline inserts arbitrary dead code in a program; while outline converts block of code into

subroutine and replace the block with a call to the subroutine. It essentially does not preserve any logical code grouping.

2.5. Antivirus / Malware Detection Mechanisms

Antiviruses have been successfully implemented and used to detect, prevent and remove malware from host machines. Antiviruses (AV) have been broadly classified to use different strategies grouped into: heuristic search, cyclic redundancy check, logic search and spy on processes to scan for viruses. [32, 30] malware intrusion can be detected using the following schemes, basically classified into:

1. Signature (rule-based) maintains an attack database, of signatures using various heuristics or rule set that includes a list of the major feats about a process (or data packet). They are created by experts with the antivirus sniffing each process passing through the system, compares them against its database, and if a match is found, it generates an alert; Else, it proceeds [9]. Its demerits includes: (a) it requires previous knowledge of known attacks to generate accurate rules, (b) are completely blind to new unrecorded attack, (c) false negative classifications that result from feats not in the rules knowledgebase and not an attack are often on the rise, and (d) addition of new rules enlarges the knowledgebase, and processes comparison that often overwhelms the antivirus processing program so that it fails to detect attacks and network anomalies. Thus, it scans the memory for processes and thread with specific signatures. To evade detection and circumvent such detection techniques/schemes, virus makers create a new string of viruses, which alters their structure but keeps its functionality via code obfuscation (metamorphic) means.
2. Anomaly-based scheme simply creates a profile statistics of all unusual threads and processes executing within the system memory through their streams of control message protocol. It also scan's for sudden exponential growth in access to data or execution memory required via code emulation technique, which creates a virtual machine or sandbox (controlled environment, honey pot) unto which the virus is allowed to run and execute. Thus, host files are also allowed to execute within it and scanned for virus. Once the virus is detected, it is no longer a threat as it is run in a controlled environment that limits its damage to the host machine. It does not rely on prior knowledge to detect undocumented, new attacks; [16, 17, 48]. This scheme also poses continually a herculean task, which aims to effectively distinguish normal memory data and resource traffic from statistically-unusual traffic. Its only demerit however, is that it can often yield high false-positives classification rates [52].

Antivirus is been found to cripple system's performance. And, any incorrect classification by it often leads to security breach – as they run at the operating system's kernel. The success however, of antivirus that uses heuristics techniques, depends on a right balance between false-positives and true-negatives. Today, malware may no longer be executables. Macros can present security risk and antivirus heavily relies on signature-detection. Metamorphic and polymorphic viruses, evades and makes signature detection, quite ineffective [18].

3. Materials and Experimental Methodology

3.1. Virus Abstract Representation

Study uses real permutation metamorphic engine (as adopted for generation of Zmist., Zperm and Zmorph viruses). It uses substitution, transposition and trash (all permutation) methods to build viruses of the same functionality. The engine changes its opcode, generating new variants from old versions authored by Zombie and extracted from VX Heaven [29]. Zmist at the point of release, was one of the most complex binary viruses ever written. It uses Entry-Point Obscuring that supports a unique code integration scheme, and occasionally

inserts jumps after each instruction in a section, pointing to the next instruction. It extremely modifies files from one generation to next through extreme camouflage. This, makes Zmist better suited for the study [12, 11, 24, 9].

3.2. String Matching Algorithm

Many attacks that aim to exploit services often carry specific strings in their packet(s) payload. Attacks on web-servers for instance often try to invoke the perl or any shell interpreter to gain access to a root shell or arbitrary commands like wget. Other exploits that use buffer overruns also carry specific commands called signatures strings. Such attacks are detected by using the string matching algorithm that search through packet(s) payload for such signatures (or strings). The problem however, arises because often – these signatures or suspicious strings can be encodes anywhere in the payload and its key-space (i.e. the number of strings to find) is potentially large. Different methods can be used to find and address such issue.

3.3. Boyer Moore Algorithm (BMA)

Boyer Moore's algorithm is one of the most efficient string matching algorithms available as it actually does find matches in a sub-linear search time. It achieves this by simply scanning through the key string from left to right. On a miss, the key string is shifted a pre-computed number of characters to the right until a match of the current character occurs. Then, the next character not yet matched is considered. Since the length of the key string and the position of the current character is known, the number of characters on which the match of the key string can occur can be computed. Fig. 2 matched the characters depicted as upper-case letters. On inspecting it, we note that the algorithm can find the exact string matches in a sub-linear search time. Its demerit(s) is that: (a) in the search for multiple key strings in a payload – the algorithm is quite inefficient, and (b) each key string is stored in its entirety.

In its naive model, the Boyer-Moore algorithm successively aligns the pattern P with the text T, and checks to see if the pattern string P matches the opposing characters of T. Further, after the check is complete, P is shifted right relative to T. The Boyer-Moore algorithm contains three (3) ideas not contained in its naive model as: (a) right to left scan, (b) bad character shift rule, and (c) good suffix shift rule. Thus, the method examines fewer than m + n characters (at an expected sub-linear-time), and that (with a certain extension) runs in linear worst case time [53] as in Fig 1 below.

In some scenarios, it is necessary to find a group of similar strings instead of an exact match. This is equal to a string search using wildcards. For example, instead of searching for the string perl.exe, one can search for p*rl.exe – which will also produce a result of perl.exe, parl.exe, pearl.exe etc. Basically, two mechanisms exists that are based on two different ideas on how to handle character substitution or insertions namely:

- Substitution Error allows the replacement of one or more characters in a key search string. So, instead of searching the whole key string, only some positions are considered. This method cannot find character insertions or deletions.
- Resemblance – here, the similarities of the two strings are measured by dividing the number of characters they have in common by the length of the longer string. Formally, this equals the division of the intersection by the union. Resemblance thus, can handle character insertion and deletion (where it exists). Its drawback is that the character positions are not taken into account. So the two strings perl.exe and lexe.rpe have resemblance one. Clearly, this is not what we want. To overcome this, the order or position numbers can be introduced.

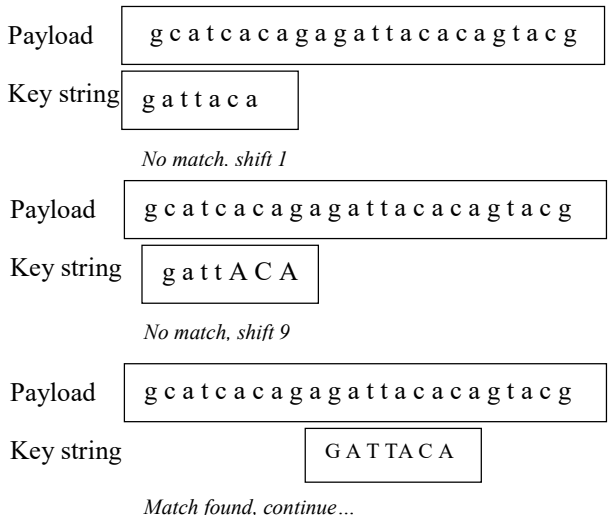

Fig. 2. Boyer Moore algorithm

3.4. The BMA: Experimental Model and Its Workings

The ideas that make BMA quite powerful and flexible are explained as thus:

1. Right to Left Scan – In any alignment given a set of text T and pattern string P, BMA first checks for the occurrence of P in the text by scanning characters from right to left (rather than from left to right as in naive model). For the alignment of P against T (as in fig. 3) – to check if P occurs in T, BMA starts at the right end of P by: (a) first compares T(9) with P(7). A match is found – so, it then compares T(8) with P(6), and so on and so forth, moving right to left until it finds a mismatch when comparing T(5) with P(3). At that point, the pattern string P is shifted right relative to T (the amount for the shift is discussed in the bad rule and good rule suffix). Then, the comparisons begin again at the right end of P. However, if the pattern string P is shifted one place to right after each mismatch, or after an occurrence of the pattern string P is found – then its worst-case run-time is given by O(nm) like its naive model (at which point it is unclear why we reverser engineered the model to compare characters from right to left). Cases where shifts of more than one position (large shifts) occurs, also abounds in many typical applications. These are resolved using the bad rule and good rule suffix shifts discussed below.

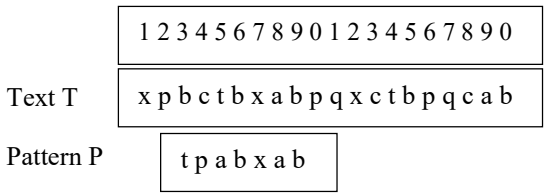

Fig. 3. Modified Boyer Moore algorithm

2. The Bad Character Rule – Suppose the last (rightmost) character string of the pattern P is y and the character in text T it aligns with is x ≠ y. When this initial mismatch occurs, if the rightmost position in P of character x – we can safely shift P to the right so that the rightmost x in P is below the mismatched x in T. Any shorter shift results in an immediate mismatch. Thus, the longer shift is correct (i.e. it will not shift past any occurrence of P in T). Again, if x never occurs in P, we can shift P completely past the point of mismatches in T. In these cases, some characters of T will never be examined and the method will actually run in sub-linear time. Again, if for a particular alignment of P against T, the rightmost n-i characters of P match their counterparts in T; But, next character to the left, P(i), mismatches with its corresponding text T(k) at position k. This rule notes that P be shifted right by max[1,i-R(T(k))] places (i.e. if the rightmost occurrence in P of character T(k) is in position j < i (including the possibility of j = 0), then we shift P so that character j of P is below character k of T. Otherwise, shift P by one position. The essence of this shift rule is to shift P by more than one character when possible. As in example above, T(5) = t mismatches with P(3) and R(t) = 1. So, P is shifted right by two positions. After the shift, the comparison of P and T begins again at the right end of P [1, 2].
3. The extended bad character rule: The bad character rule is useful for mismatches near the right end of P. But, it has no effect if the mismatching character from T occurs in P to the right of the mismatch point. This is common when the alphabet is small and the text contains many similar, but not exact substrings. The DNA is a typical situation with only 4-alphabets. Even, in proteins, we have an alphabet of size 20, which contains different regions of high similarity. In both cases, we use extended bad character rule as it is more robust. It notes that when a mismatch occurs at position i of P and the mismatched character in T is x; Then, shift P to the right so that the closest x to the left of position i in P is below mismatched x in T. The extended rule gives larger shifts, and the only reason to prefer the simpler rule is its additional cost in implementing the extended rule. The simpler rule uses only O(jj) space (is the alphabet) for array R, and one table lookup for each mismatch. But, the extended rule is implemented in only O(n) space, and at most one extra step per character comparison. Though, the amount of added space is not often a critical issue – the more critical question is if the longer shifts make up for the added time used by the extended rule. The original Boyer-Moore algorithm only uses the simpler bad character rule [1, 2].
4. The Good Rule Suffix – The bad character rule by itself is reputed to be highly effective in practice, especially for English text; But, less effective for small alphabets as it does not lead to a linear worst-case running time. Thus, we introduce the strong good suffix rule. The original preprocessing method, for the strong good suffix rule is quite difficult and somewhat mysterious (though a weaker version, is easier to understand). In fact, the preprocessing for the strong rule was given incorrectly [1, 2]. Suppose in a given alignment of P and T, a substring t of T matches a suffix of P; But, mismatch occurs at the next comparison to the left. Then find (if it exists) the rightmost copy t' of t in P such that t' is not a suffix of P and the character to the left of t' in P differs from the character to the left of t in P. Shift P to the right so that substring t' in P is below substring t' in T. If t' does not exist, then shift the left end of P past the left end of t in T by the least amount so that a prefix of the shifted pattern matches a suffix of t in T. If no such shift is possible, then shift P n-places to the right. If an occurrence of P is found, then shift P by the least amount so that a proper prefix of the shifted P matches a suffix of the occurrence of P in T. If no such shift is possible, then shift P by n-places (i.e. shift P past t in T). For example, consider the alignment of the pattern P in the text T in Fig. 4.

When a mismatch occurs at position 8 (i.e. P(8)) and T(10) as asterisked, t = ab, and t' occurs in P starting at position P(3). Thus, P is shifted right by six places resulting in the sequence string alignment as in Fig. 5 – with its listing as in Fig. 6 respectively. Note that the extended bad character rule would have only shifted P by only one place as in the example.

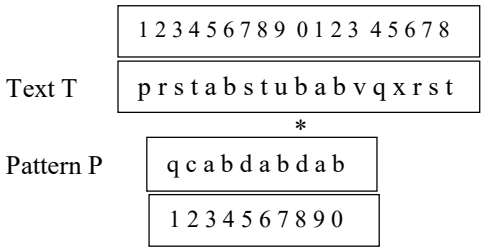

Fig. 4. Modified BMA Strong Goo

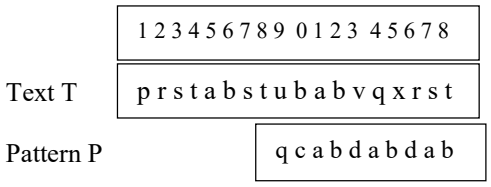

Fig. 5. Modified BMA Strong Good

```
{Preprocessing stage}
    Given the pattern P,
    Compute L0(i) and l0(i) for each position i of P, and compute R(x) for each character x 2 .
{Search stage}
    k := n;
    while k < m do
        begin
        i := n;
        h := k;
        while i > 0 and P(i) = T(h) do
            begin
            i := i - 1;
            h := h - 1;
        end;
        if i = 0 then
            begin
            report an occurrence of P in T ending at position k.
            k := k + n – l'(2);
        end
        else
            shift P (increase k) by maximum amount determined by
            the (extended) bad character rule and the good suffix rule
        end;
    end;
```

Fig. 6. Listing of Complete BMA Pseudo-code

4. Findings and Discussion

4.1. Findings

To measure its classification accuracy, we measure its rate of classification and improvement percentages with data sets from Section II as summarized in Tables 1 and 2 respectively:

$$Misclassification\ Rate(MR) = \frac{Number\ of\ Inccorect\ \Pr edict}{Number\ of\ Sample\ Set} \tag{1}$$

Table 1. Classification Accuracy of Each model

	Classification Precision	
Model	*Training Data*	*Testing Data*
Naïve BMA	89%	92%
BMA	99.1%	99.6%

Also, its improvement percentage is computed as thus:

$$\mathrm{Im}\ provement\ Percentage = \frac{MR(A) - MR(B)}{MR(A)} * 100 \tag{2}$$

Table 2. Improvement Percentage

	Improvement %	
Model	*Training Data*	*Testing Data*
Naïve BMA	4.52%	5.6%
BMA	9.54%	10.3%

Results obtained from tables 1 and 2 respectively shows that exact string matching algorithm outperforms their counterpart unsupervised evolutionary models [9, 48, 7]. BMA algorithm has a classification accuracy rate of 99.6% (i.e. with a 0.4% error rates in false-positives and true-negatives results) as opposed to counterpart Naïve BMA that offers a classification rate of 92%. Also, it promises an improvement rate of about 10.3% over the Naïve BMA that offers 5.6% improvement.

4.2. Structure

Today, the appropriate content (signature string or code) for malware detection in minimal time is a critical factor. String algorithms play vital role thus. Researches by different groups in furtherance of string algorithms implemented at the various software abstractions aims to make pattern searching faster. In applying these to various applications, the best algorithm for the different scenarios can be determined. The recommended BMA algorithm gives reduced complexity with also reduced computation time [3, 53, 40]. The BMA algorithms assigned to various applications may not be best optimal; But, they are often better than most general algorithms. It is noted that most applications uses BMA algorithm for their effective and efficient functionality, as it has lesser time complexity. Also, Other algorithms depends upon the type of input and is efficient for certain or particular application [3, 41-42].

Metamorphics transform its codes as they propagate to avoid detection by using obfuscation methods to

alters its behaviour when it detects its execution within virtual machine (sandbox) as means to challenge a deeper analysis [46-47]. Virus writer use weaknesses of antiviruses which are limited to static and dynamic analysis. Thus, they attack these feats in system: (a) data flow, (b) control flow, (c) procedure abstract, (d) property verification, and (e) disassembly means to counter scans and identify such metamorphic viruses [24, 45]. To mutate its code generation, metamorphics analyze their codes and re-evaluates the evolved, mutated codes generated (since complexity of transformation in the previous generation has a direct impact on its current state, how a virus analyses and transforms code in its current generation). Thus, they employ code conversion algorithm that helps them detect their own obfuscation and reordering [43-45].

Acknowledgements

We acknowledge gratefully the Federal University of Petroleum Resources Effurun (FUPRE), the Tertiary Education Trust Fund (TETFund) for funding intervention, and we also appreciate the Vice Chancellor Prof. Akii O.A. Ibhadode who has made research a way of life in the Federal University of Petroleum Resources, Effurun. Thank you for the vision. We acknowledge the efforts and cooperation of the all staff of the Federal University of Petroleum Resources Effurun especially from the ICT Unit for providing server log files and other materials necessary for this research.

References

[1] Grossi, R and Luccio, F., (1989). Simple and efficient string matching with k mismatches, Information Processing Letters, Vol. 33, pp113–120.

[2] Ukkonen, E and Wood, D., (1990). Fast approximate string matching with suffix automata, Report A-1990-4, Department of Computer Science, University of Helsinki.

[3] Zink, T., (2009). Network security algorithms, Konstanzer Online Publikationss-System, www.nbn-resolving.de/urn:nbn:de:bsz:352-175988

[4] Daoud, E and Jebril, I., (2008). Computer Virus Strategies and Detection Methods, International Journal of Open Problems Computational Mathematics, 1(2), [web]: www.emis.de/journals/IJOPCM/files/IJOPCM1.2.8.pdf

[5] Dawkins, R., "The selfish gene", Oxford University Press, Second Edition, 1989.

[6] Zakorzhevsky, E.R., "Monthly malware statistics", 2011,[online]:www.securelist.com/en/analysis/204792182/Monthly_Malware_Statistics_June_2011.

[7] Allenotor, D., "An Evolvable Framework for Metamorphics". Computing, Information Systems, Development Informatics and Allied Research Journal, 2016, Vol 7 No 2. Pp 33-40 Available online at www.cisdijournal.net

[8] Ojugo, A.A., "Computer virus evolution: polymorphics analysis and detection", Journal of Academic Research, 2010, Vol. 15, No. 8, p34 – 46.

[9] Ojugo, A.A., R.E. Yoro., A. Eboka., M.O. Yerokun., C.N. Anujeonye and F.N. Efozia (2014). Evolutionary model for virus propagation on networks, Automation, Control and Intelligent Systems, 3(4): 56-62, doi: 10.11648/j.acis.20150304.12.

[10] Ojugo, A.A., A.O. Eboka., R.E. Yoro., M.O. Yerokun and F.N. Efozia., (2015). Framework design for statistical fraud detection, Mathematics and Computers in Science and Industry (Mathematics and Computers in Science and Engineering Series), 50: 176-182, ISBN: 978-1-61804-327-6, ISSN: 2227-4588

[11] Ye, Y., Wang, D., Li, T and Ye, D., "Intelligent malware detection based on association mining", Journal of Computer Virology, 2008, Vol. 4, No. 4, p323–334, doi: 10.1007/s11416-008-0082-4.
[12] Szor, P., "The Art of Computer Virus Research and Defense", Addison Wesley Symantec Press. 2005, ISBN-10: 0321304543, New Jersey.
[13] Mishra, P., "Taxonomy of software unique transformations", 2003, www.cs.sjsu.edu/faculty/stamp/students/FinalReport.doc
[14] Orr, "The viral Darwinism of W32.Evol: an in-depth analysis of a metamorphic engine", 2006, [online]: available at http://www.antilife.org/files/Evol.pdf
[15] Orr, "The molecular virology of Lexotan32: Metamorphism illustrated", 2007, [online]: www.antilife.org/files/Lexo32.pdf
[16] Singhal, P and Raul, N., "Malware detection module using machine learning algorithm to assist centralized security in enterprise network", International Journal of Network Security and Applications, 2012, 4(1), doi: 10.5121/ijnsa.2012.4106, p61
[17] Rabek, J., Khazan, R., Lewandowski, S., Cunningham, R., "Detection of injected, dynamic generated and obfuscated malicious code", In Proceedings of ACM Workshop on Rapid Malcode, 2003, p76.
[18] Filiol, E., "Computer Viruses: from Theory to Applications", New York, Springer, 2005, ISBN 10: 2287-23939-1.
[19] Hashemi,S., Yang, Y., Zabihzadeh, D and Kangavari, M., "Detecting intrusion transactions in databases using data item dependencies and anomaly analysis", Expert Systems, 2008, Vol. 25, No. 5, p460, doi:10.1111/j.1468-0394.2008.00467.x
[20] Grimes, R., "Malicious Mobile Code: Virus Protection for Windows", O'Reilly and Associates, Inc., Sebastopol, CA, USA, 2001.
[21] Cohen, F., "Computer viruses: theory and experiments", Computer Security, 1987, 6(1), p22-35.
[22] Sung, A., Xu, J., Chavez, P., Mukkamala, S., "Static analyzer of vicious executables", Proceedings of 20th Annual Computer Security Applications Conference, IEEE Computer Society, 2004, p326-334.
[23] Venkatesan, A., "Code obfuscation and metamorphic Virus Detection", Master thesis, San Jose State University, 2006, www.cs.sjsu.edu/faculty/students/ashwini_venkatesan_cs/report.doc
[24] Konstantinou, E., "Metamorphic virus: analysis and detection", Technical report (RHUL-MA-2008-02), Dept. of Mathematics, Royal Holloway, University of London, 2008.
[25] Walenstein, R., Mathur, M., Chouchane R., and Lakhotia, A., "The design space of metamorphic malware", In Proceedings of 2nd Int. Conference on Information Warfare, 2007, p243.
[26] Wong, W., "Analysis and Detection of Metamorphic Computer Viruses", Master's thesis, San Jose State University, 2006, http://www.cs.sjsu.edu/faculty/students/Report.pdf
[27] Borello, J and Me, L., "Code obfuscation techniques for Metamorphics, 2008, [online]: available at www.springerlink.com/content/233883w3r2652537
[28] Aycock, J., "Computer Viruses and malware", Springer Science and Business Media, 2006.
[29] VX Heavens Virus Collection, [online]: http://vx.netlux.org/
[30] Ojugo, A.A and Yoro, R.E., "Computational intelligence in stochastic solution for Toroidal Queen", Progress in Intelligence Computing Applications, 2013a, Vol. 2, No. 1, doi: 10.4156/pica.vol2.issue1.4, p46
[31] Ojugo, A.A., Emudianughe, J., Yoro, R.E., Okonta, E.O and Eboka, A.O., "Hybrid artificial neural network gravitational search algorithm for rainfall runoff", Progress in Intelligence Computing and Applications, 2013b, Vol. 2, No. 1, doi: 10.4156/pica.vol2.issue1.2, p22.
[32] Ojugo, A.A., Oyemade, D.A., Allenotor, D., Longe, O.B and Anujeonye, C.N., "Comparative Stochastic Study for Credit-Card Fraud Detection Models,. African Journal of Computing and ICT, 2015, Vol 8, No. 1, Issue 2. pp 15-24.
[33] Ojugo, A.A., Eboka, A.O., Yoro, R.E., Yerokun, M.O and Efozia, F.N., Hybrid model for early diabetes diagnosis, Mathematics and Computers in Science and Industry (A Mathematics and Computers in Science and Engineering Series), 2015, 50: 176-182, ISBN: 978-1-61804-327-6, ISSN: 2227-4588

[34] Ojugo, A.A., "A profile hidden markov model for forecasting energy spread options direction and volatility, Technical Report for Dynamic High Performance Computing Research Group of the Federal University of Petroleum Resources Effurun, 2013, FUPRE-TR-DHCP-08, Pp 10-24.
[35] Ramage, D., "Hidden markov model fundamentals", Lecture notes in Computer Science, [online source]: www.springerlink.com/content/lecturen_notes/cs/235483w3r2652537
[36] Noreen, S., Ashraf, J and Svrenahak, K., "Malware detection using evolutionary models", International Journal of Virology, 2008, Vol. 23, No. 2, p123-132.
[37] Ojugo, A., A.O. Eboka., E.O. Okonta., E.R. Yoro and F.O. Aghware., "Genetic algorithm trained rule-based intrusion detection system", Journal of Emerging Trends in Computing and Information Systems, Vol. 3, No. 8, 2012, Pp 1182-1194
[38] Ursem, R., Krink, T., Jensen, M.and Michalewicz, Z., "Analysis and modeling of controls in dynamic systems", IEEE Transaction on Evolutionary Computing, 2002, 6(4), p378-389.
[39] Clerc, M., "The .Aswarm and the queen: towards a deterministic and adaptive particle swarm optimization", In Proceedings of Evolutionary Computation (IEEE), 1999, 5, p123-132.
[40] Gray, J and Klefstad, R., "Adaptive and evolvable software systems: techniques, tools and applications", 38th Annual Hawaii Int. Conf. on System Sciences, 2005, p274, IEEE Press.
[41] Hassan, R and Crosswley, W., "Variable population-based sampling for probabilistic design optimization and with a genetic algorithm", Proceedings of 42nd Aerospace Science, p32, Reno: NV, 2004.
[42] Hassan, R., Cohanin, B., De Wec and Venter, G., "Comparison of particle swarm optimization and genetic algorithm", In Proceeding of 44th Aerospace Science, 2004, Washington, p56.
[43] Homaifar, A.A., Turner, J and Ali, S., "N-queens problem and genetic algorithms", In Proceedings of the IEEE Southeast conference, 1992, p262.
[44] Hu, X., Eberhart, R.C and Kennedy, J., "Solving constrained nonlinear optimization problems with PSO, In Proceedings of the Multi-conference on Systems, Cybernetics and Informatics, 2005a, p234.
[45] Hu, X., Eberhart, R.C and Shi, Y., "Swarm intelligence for permutation optimization: study of n-queens", Proceedings of IEEE Genetic Evolutionary Computing on Memetic Algorithm, 2005b, p243
[46] Kennedy, J and Mendes, R., "Population structure and particle swarm performance", In Proceedings of the IEEE Congress on Evolutionary Computation, 2002, p1671, Honolulu
[47] Lakhotia, A., Kapoor, A and Kumar, E.U., "Are metamorphic computer viruses really invisible?", 2004, Part 1, Virus bulletin, p5-7.
[48] Ojugo, A.A and Eboka, A.O,, "An intelligent hunting profile for evolvable metamorphic malware", IEEE African Journal of Computing and ICT, 2015, 8(1-2), p181.
[49] Desai, P., "Towards an undetectable computer virus, Masters Thesis, Department of Computer Science, San Jose State University, 2008
[50] Gong, R.H.., Zulkernine, M & Abolmaesumi, P., "A software implementation of GA based to network intrusion detection", 2005, cse.msu.edu/~cse848/2011/Student_papers/Tavon_Pourboghrat.pdf
[51] Kandeeban, S. S. and Rajesh, R. S., (2007): GA for framing rules for intrusion detection, J. Comp. Sci and Security., 7(11), ISSN:1738-7906, PP.285-290.
[52] Kurose, J.F and Ross, K.N., "Computer network a top down approach", Pearson publisher, 2010, ISBN-10: 0-13-136548-7.
[53] Ojugo, A.A and Allenotor, D., "Text mining identification and detection using the exact string matching algorithm: a comparative analysis", Journal of Digital Innvations & Contemporary Research in Science, Engineering and Technology, 2018, 6(1), p169-180.

Authors' Profiles

Arnold Adimabua Ojugo received BSc in Computer Science from the Imo State University Owerri in 2000, MSc in Computer Science from the Nnamdi Azikiwe University Awka in 2005, and PhD in Computer Science from Ebonyi State University Abakiliki in 2013. He currently lectures with the Department of Mathematics/Computer Science of the Federal University of Petroleum Resources Effurun, Delta State, Nigeria. His research interests: Intelligent Systems and Control, High Performance and Dynamic Computing, Machine-Learning, Computer Vision, Ubiquitous Computing, Data Security and Forensics. He is an Editor with the Progress for Intelligent Computation and Application, Advancement for Scientific and Engineering Research and SciencePG (ACIS, AJNC, NET and WCMC Journals). He is a member of: The Nigerian Computer Society, Computer Professionals of Nigeria and International Association of Engineers (IAENG).

Andrew O. Eboka `received his HND in Computer Science from Akanu Ibiam Federal Polytechnic in the year 1998, Ebonyi State, PGD from Ebonyi State University in 2013, BSc/Ed in Computer Science Education from the Enugu State University of Science and Technology, Enugu in 2013. He received his MSc in Network Computing from Coventry University, United Kingdom. He currently lectures with the Department of Computer Science Education at Federal College of Education (Technical), Asaba, Delta State, Nigeria. His research interests include: Network Security and Management, Cyber-Security, Ubiquitous Computing, Data Security and Forensics. He is a member of: The British Computer Society and Association of Computer Machinery.

How to cite this paper: A. Ojugo, A. O. Eboka,"Signature-Based Malware Detection Using Approximate Boyer Moore String Matching Algorithm", International Journal of Mathematical Sciences and Computing(IJMSC), Vol.5, No.3, pp.49-62, 2019. DOI: 10.5815/ijmsc.2019.03.05