Volume 7, Issue 2, February 2017 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Short Review on Metamorphic Malware Detection in Hidden Markov Models

Yeong T. Ling*, Nor Fazlida Mohd Sani
Faculty of Computer Science & Information Technology, Universiti Putra Malaysia, 43400 UPM, Serdang, Selangor, Malaysia

DOI: 10.23956/ijarcsse/V7I2/01218

Abstract— Metamorphic malware is well known for evading signature-based detection. To cope up with numerous malware which can emerge easily by using open source malware generator, efficient detection in terms of accuracy and runtime performance shall be considered during analysis. Detection strategies such as data mining combine with machine learning have been used by researchers for heuristically detecting malware. In this paper, we present Hidden Markov Model as an efficient metamorphic malware detection tool by exploring the common obfuscation techniques used in malware while reviewing and comparing the different studies that adopt HMM as a detection tool.

Keywords— Metamorphic, Malware, Hidden markov models, Obfuscation, Heuristics

I. INTRODUCTION

Malware, stands for 'malicious software', invades the privacy and integrity of digital dataas well as performance and functionality of a computer system. There are a number of different ways malware can get onto your computer, such as downloading free or legal software from the internet that secretly contains malware, clicking a fake error message that starts a malware download, opening an email attachment which contains malware, visiting an infected website, downloading infected applications from trusted online store, etc. It is threatening today's computing world by its high complexity of growing. According to [1], it estimated a maximum of $65.05 million economic lost in U.S. due to cyber attack in the year of 2015. There was a 26% increased in 2014 and over half a billion personal records were stolen or lost in year 2015 [2][3]. The most popular malware detection technique used by commercial product today is signature-based detection, however it is not capable of detecting new, unknown malware that result in zero day attacks.

Metamorphic malware is capable of rewriting its own code with each infection, or mutate to new variant to disguise the malicious code without changing its functionality [4], the so called malware families. It adopts code obfuscation to disguise its malicious behavior and the same time generate various variants within the same malware family. Metamorphic malware can easily evade signature based detection since they can create variants by code morphing which does not have common signature among the variants.

The process of detecting malware, called malware analysis, is a process where the analyst collects all the required information from an executable file, study and extract the important characteristic that can represent the file, compare with existing known malware characteristic and determine whether it belongs to malware or benign file. Currently, number of malware variants are growing but number of malware families or classes remains constant.

II. TYPES OF MALWARE

Depending on their way of infection, malware can be classified into several classes, such as worm, virus, trojan horse, spyware, bot, rootkit, etc. [4]. Most commercial anti-virus software uses signature-based detection technique to scan for malware. Signature-based detection uses unique strings of an examined file to create a binary pattern of its machine code. A major limitation of signature-based detection is that it is unable to identify malicious files for which signatures have not yet been updated in the signature database. Therefore, malware writers frequently mutate their codings to retain malicious functionality by changing the file's signature. In order to evade traditional signature-based malware detection, malware writers create armored malware to dissuade the anti-malware tools. They apply concealed strategies by obscuring or encrypting a code to make it difficult to understand, analyze and be detected. These strategies can be divided into following three groups:.

A. Packing

Malware in this type uses compression and encryption mechanisms to hide its malicious code. During runtime, the decryptor that was carried in the body of packed code will restore the malicious code in memory and execute it. In order to detect them, unpacking needs to be done. Current anti-malware tools normally use entropy analysis to detect packing [5] files, to unpack a program, they must first know the packing algorithm used to pack the program. Fig. 1 illustrated the basic structure of a packed malware.

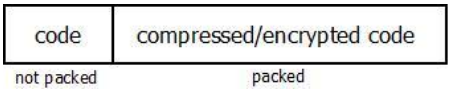

Fig. 1 Packed malware structure

B. Polymorphism

Polymorphic malware use encryption algorithm to encrypt itself. The first polymorphic malware that has this armored technique is Win95/Marburg. When an infected program executes, the malware will be decrypted and then written to system's memory for execution. In each execution of the infected program, a new version of the malware is encrypted and stored for next execution. This produces a different malware signature. The changed malware will keep the same functionality. It is possible for a signature-based technique to detect polymorphic malware during runtime. Fig. 2 illustrates the basic structure of polymorphic malware encrypt itself with endless number of new decryptors. The decryptor changes shape from generation to generation by creating different decryption keys so that different encryption algorithm can be built, but under the encryption, the malicious code is constant[6].

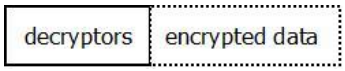

Fig. 2 Polymorphic malware structure

C. Metamorphism

The first metamorphic malware was released in 1989 called Win95/*Regswap*. Metamorphic malware behaves as in automatically reprogram itself each time it propagates or is distributed to a new host. In each propagation, it mutates its syntax or shape, to foil signature based detection, while maintaining its malicious functionality. It applies semantics preserving transformations to modify their own code so that one variant of the malware has little resemble to another variant even though semantically their functionality is the same. Therefore, a metamorphic malware is equipped with a 80% the morphing engine that carries the 20% malicious code as input and morphs it at runtime to a syntactically different but semantically equivalent variant, so that the same detection does not work on every mutated variant. The malware will first find its code, then disassemble the code into mnemonics by a disassembler. The disassembled code is then analyzed to gather useful information for code obfuscator. The obfuscator will then uses this to transform the binary code, so that new variant of malware has different structure but the same functionality. The code is then compressed and assembled into machine code through assembler. The code is now obfuscated and is attached. Fig. 3 illustrates a simple structure of the replication of a metamorphic malware in *n* generations, G, where V is the virus attached on file at the *i*th generation, for $i \leq n$. It can be seen that there is no constant behavior exists between different generations [6].

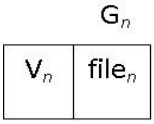

Fig. 3 Metamorphic malware structure

According to [7], there are two ways to group metamorphic malware, based on how they communicate and transform themselves. In terms of communication, it can be categorized into *open-world* or *closed-world* malware. *Open-world* malware can mutate through connection with other sites over Internet, just as *Conficker* worm did in 2008. As for *closed-world* malware, on the other hand, it reprograms itself by referring to a pseudo-code representation that it carries.

In terms of transformation, it has binary or alternate-representation transformer. Binary transformer is when during evolution, it mutates the binary executable itself, e.g. binary or post-compilation obfuscation. Whereas in alternate-representation transformer, it refers to a pseudo-code representation and mutates based on it, e.g. assembly level, source code or mnemonic level obfuscation. Metamorphic malware uses obfuscation techniques to hide itself by changing its structure on each infected program while keeping the same functionality. By obfuscating repeatedly, two copies from the same malware will be quite different and most of current detection methods fail to completely identify such every-increasingly stealth metamorphic malware. The obfuscation techniques can be divided in two types, data flow obfuscation (junk or dead code insertion, variable or register substitution, instruction replacement or permutation) and control flow obfuscation [8].

III. OBFUSCATION TECHNIQUES

To evade detection, metamorphic malware use several different techniques to evolve their codes into new generations, but have same malicious functionality. This section describes these techniques.

A. Garbage code insertion

Garbage or dead code insertion is to make the malware codes look different and thus possibly evade hexadecimal string matching. It does not change a program's functionality but to confuse and exhaust the emulator during code analysis [4]. Garbage code as such the instruction NOP does not have any functionality and garbage instructions may also be a branch of code which is never reached. Fig. 4 illustrates two generations of the Win32/*Evol* which appeared in July 2000 where garbage codes were inserted by the metamorphic engine[4].

Original		After		
C7060F000055	mov [esi], 5500000Fh	BF0F00055	mov edi, 5500000Fh	
C7460488EC5151	mov [esi+0004], 5151EC8Bh	893E	mov [esi], edi	
		5F	pop edi	; garbage
		52	push edx	; garbage
		B640	mov dh, 40	; garbage
		BA8BEC5151	mov edx, 5151EC8Bh	
		53	push ebx	; garbage
		8BDA	mov ebx, edx	
		895E04	mov [esi+0004], ebx	

Fig. 4 Garbage code insertion

B. Instruction substitution

In this obfuscating technique, an instruction is replaced with its equivalent instruction by keeping the same functionality. Types of instructions substitution include reversing of the branch conditions, register moves substitute by push/pop sequences, alternative opcode encoding and xor/sub and or/test interchanging. Fig. 5 shows an example of Win95/*Bistro* malware with different instructions but perform the same function - zeroing the content of the *eax* register.

Original	After
push ebp	push ebp
mov ebp, esp	**push** esp — mov replaced by push pop
mov esi, dword ptr[ebp+08]	**pop** ebp
test esi, esi	mov esi, dword ptr[ebp+08]
je 401045	**or** esi, esi — test/or interchange
mov edi, dword ptr[ebp+0c]	je 401045
or edi, edi	mov edi, dword ptr[ebp+0c]
je 401045	**test** esi, esi
xor edx, edx	je 401045
	sub edx, edx — xor/sub interchange

Fig.5 Example of Wind95/Bistro substitute instruction of codes

C. Register swapping

In this technique, the virus instruction operands are stored in different registers for each new infection. It replaces the use of a register in an instruction with another unused register as Fig. 6 shown. Even though this has no impact on a program behavior, it does evade signature-based detection.

Original		After	
5A	pop edx	58	pop eax
BF04000000	mov edi, 0004h	BB04000000	mov ebx, 0004h
8BF5	mov esi, ebp	8BD5	mov edx, ebp
B80C000000	mov eax, 00Ch	BF0C000000	mov edi, 000Ch
81C288000000	add edx, 008h	81C088000000	add eax, 0088h
8B1A	mov ebx, [edx]	8B30	mov esi, [eax]
899C8618110000	mov [esi+eax*4+00001118],ebx	89B4BA18110000	mov [edx+edi*4+00001118],esi

Fig.6 Register usage swapping

D. Code reordering

This is code permutation or subroutine permutation. It reorders its virus code by using jump instruction or complex transfer of control as the backbone for obfuscation. It basically creating different permutations of the code while keeping the functionality. As shown in Fig. 7, the malicious code divides itself into five modules with one of the possible module permutation in different generations. As for the subroutine permutation, the order is irrelevant and does not impact the functionality of the malware as the order in which a subroutine appears in the program is totally irrelevant and does not affect the execution of the program.

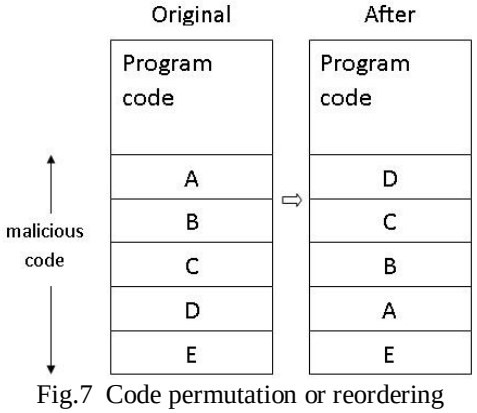

Fig.7 Code permutation or reordering

E. Host code mutation

To generate new variants, Win95/*Bistro* mutates its code as well as the original code of host file into which they are inserted through a randomly executed code-morphing routine. Bistro uses a random code block insertion engine to mutate its code. This code block can generate millions of iterations before any actual malicious instructions been carried out.

IV. MALWARE ANALYSIS

There are two major approaches in malware detection, namely static and dynamic analysis. Static analysis is a process of analyzing binary executables or source code without actually running it. The advantage of static analysis approaches is they can access to all of the software's possible behaviors, on the other hand, they required well enough software artifacts in source code or binary code level to execute a build. Furthermore, malware can easily use anti-analysis techniques to mislead disassembler and static analyser tools. Static analysis is very much restricted due to self-modifying code, binary packers, or encryption [9]. Moser et al. [10] investigated the limits of static analysis. In their work, they introduced a scheme based on code obfuscation revealing the fact that the static analysis by itself is not enough to detect malware and dynamic analysis is a necessary complement to static analysis as it is less vulnerable to code obfuscation.

Whereas in dynamic analysis approaches, there is no requirement to have access to source code or binary code and during operational deployment it can find infrastructure, configuration and patch errors that static analysis tool missed. It analyzes the behavior of a malicious code while it is being executed in a controlled environment. There are two types of dynamic malware analysis, namely debug execution (WindDbg, SoftICE and OllyDbg) and run-time execution (FileMon, RegMon, TcpView and Process Explorer) however, there is limited scope of actual instructions being executed [11] and it is time consuming which raise the issue of scalability. The virtual environment in which malware are executed is different from the real one, sometimes malware behavior is triggered only under certain conditions. Since there are many free obfuscation tools available, large number of malware samples can be generated everyday. Therefore, an automated approach to reduce close human analysis is needed. Due to this, heuristic-based detection has become the new study of malware detection.

V. DETECTION TECHNIQUES

A. Signature-based

Signature-based approaches are the mainstream in commercial anti-virus systems due to their low false positive rate and low computational complexity. They work by referring to a virus database of known "signature" associated with specific files. These signatures are compiled when an anti-virus system records a static "fingerprint" of a file, which is a sequence of byte. When a file is scanned, its signature is compared with the current contents of the virus database to determine if there is match. However, this method can not detect malicious files for which no signatures have been recorded yet. It requires big disk space and a fair amount of processing power to grind through all the data. It has been shown that it is easily defeated by simple code obfuscation techniques and can not handle the zero-day attacks.

B. Behavior-based

Behavior-based detection watches processes for telltale signs of malware, which it compares to a list of known malicious behaviors. Malware, just like any other running program, use the services provided by the host system to sent inputs to processor, memory, programs and other operating resources [12]. It works by sandbox the execution of a file and observe how the file is actually run. The anti-virus tools seek to identify malware by watching for abnormal or suspicious behavior of the file. It utilizes the behavior information of the malware during its execution. Therefore, behavior-based detection no longer identifies a single byte of signature in a malware, but a entire class of malware. Therefore, it can detect unknown, encrypted, and obfuscated malware [13].

One major approach of behavior-based detection is anomaly-based detection, where it uses its knowledge of what is normal behavior of a program to decide the maliciousness of a inspect program. A major advantage of anomaly-based detection is its ability to detect zero-day attacks. However, the two fundamental limitations of this technique are its high false alarm rate and the complexity involved in determining what features should be learned in the training phase. This contributes to the high false positive rate commonly associated with anomaly-based detection techniques. To address this problem, specification-based detection emerge to approximates the system by leverage some specification or rule set of what is normal behavior in order to decide the maliciouness of a inspected program. However, it is difficult to specify accurately the complete behaviors of a system or program that been inspected [14], this turns out to cause problem of false positives (first alarms) that normal program is falsely considered as malware.

C. Heuristic-based

Heuristic-based detection uses heuristic analysis to examines files for characteristic that the system considers suspicious [15]. Heuristic analysis which relies on datasets uses data mining and machine learning techniques to learn the characteristic or behavior of an executable file. Just as first attempt by Schultz et al., where Na¨ıve Bayes and Multi Na¨ıve Bayes were used to classify malware and benign files [16]. Characteristic or behavior of an executable file are the features which can represent the file itself. These features include n-grams [17], [18], instruction sequences (opcodes) [19], [20], API call sequences [21], [13], control flow graph [22], [23], etc.

Currently, there are a number of works that used HM for metamorphic malware detection. The following section study the HMM methods that have been used in malware classification and detection.

VI. RELATED WORK

A. Hidden Markov model

Hidden Markov Model (HMM) is a powerful statistical tool to perform malware analysis [24]. Since malware behavior is commonly represented as a sequence of events, HMM becomes a good candidate tool for malware behavioral analysis. The biggest strengths of HMM is its ability to find the hidden state or underlying structure in a given sequential data. Given a sufficient number of observations, we can train a HMM to represent the data. We can then score a given sequence of observations to determine how well the model fits the observations. A generic HMM is illustrated in Fig. 8, where X_t represents the hidden state at time t, with state transition probability matrix A. Each observation O_i gives some information about the hidden state through probability distributions contained in B.

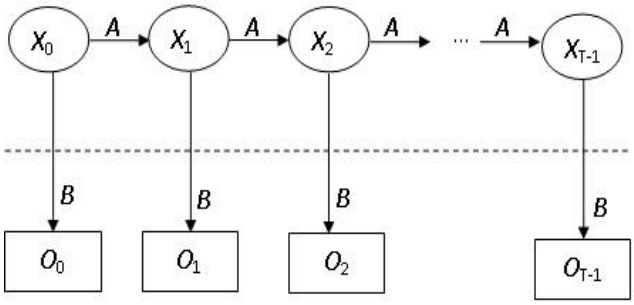

Fig.8 Hidden Markov model

B. Malware detection with HMM

Based on the work in [24], Wong et al. [25] proposed using HMMs to detect metamorphic virus variants using opcode sequences. From their result, it shown that NGVCK can generate the most morphed variants than other generators and HMM can be used to model virus families. Using a threshold approach, the detection rate is between 97.5% to 97.1% with false positive rate (FPR) 3.1%. However, it failed to detect viruses on specific designed metamorphic malware engine such as MetaPHOR or MWOR.

Desai et al. [26] later created a metamorphic engine on top of [25] using code obfuscation techniques, including benign floating point opcode insertion, and proved that HMM can detect highly metamorphic viruses.

Austin et al. [27] further explored HMMs in more depth to better understand the meaning of the hidden states by using dueling HMM strategy. They examined HMMs for four different compilers, hand-written assembly code, three virus construction kits, and a metamorphic virus. Their results show that the dueling HMM strategy outperform the threshold based technique [24] and is effective at identifying viruses.

Kalbhor et al. expanded [27] by combining dueling HMM strategy [28] with threshold approach [24] to minimize the performance overhead at the same time provide flexibility to adjust the desired false positive or negative ratio.

Rezaei et al. [29] proposed a method based on detection circle using logarithm probabilities on string occurrence, specifically located character occurrence, and the amount of virus similarities. They claimed the proposed method can achieved more than 91% viruses which outperformed anti-viruses scanner. When applying encryption, anti-virus failed to detect the viruses but their method can detect 70% of them. This implies applying heuristic method is better than tradisional anti-virus scanner.

In order to better represent a program behavior, [30] proposed a malware classification method based on HMM using system calls as discriminating dynamic features. The result shown a weighted average of malware detection accuracy is 97% but below 50% for benign files detection when varying number of observation sequences are added. This indicates that maximum likelihood approach by itself is not sufficient to identify malware from benign files.

Thunga et al. [31] used HMM to identify viruses of metamorhpic family. They trained multiple HMMs for different metamorphic malware generators by relying on n-grams instead of term frequency or single opcode to define a HMM state. Term Frequency-Inverse Document Frequency (TFIDF) was used to eliminate unnecessary features before training the HMM with n-gram opcode sequence to detect real malware family. The result shown detection accuracy of about 90% for three different virus families; *zbot*, *winwebsec* and *zeroaccess*, which perform better than commercial antivirus scanners.

To improve [28] and make a few changes to [24], Gharached et al. [32] proposed a two layer HMMs to overcome issue of metamorphic malware evades traditional HMMs in terms of analysis speed. They used technique from [33] to extract opcode sequences. The results shown using the proposed technique is faster and more accurate than [28].

Prasad et al. [34] proposed a metamorphic malware classification based on HMM. They used one model for each malware family, which are Cygwin, G2, NGVCK, VCL32, MPGEN and collected more than 12,000 malware from various online resources including project malicia [35] to classify them into respective families. However, by using 100 sample size, NGVCK only can get 88% accuracy rate.

In the believed of able to achieve faster detection for unknown malware, Annachhatre et al. [36] proposed malware classification based on HMMs. They used variety malware generated from different generator as input to train HMMs. Unlike [24], [27], [32], [34], the authors employed k-mean clustering algorithm as a classifier to classify malware based on the resulting score from HMMs . Their results shown that HMMs are an effective tool for classifying malware. A summary of HMMs for malware detection is shown in Table I.

Table I: Comparison of HMMs in Malware Detection

Authors	File representation	Techniques	Pros	Cons
Wong et al. [24]	opcode sequence	Mishra algorithm, log-likehood	better than commercial virus scanner to detect highly morphed variants	only NGVCK family detection
Desai et al. [26]	opcode sequence	Mishra Algorithm	able to identify morphed copies	no malware family classification
Austin et al. [27]	Opcode	-	better result than threshold-based	Imbalance adjustment between FP and FN; performance overhead
Kalbhor et al. [28]	Opcode	multiple HMM	high detection rate for MetaPHOR,MWOR	Performance overhead
Rezaei et al. [29]	Opcode	Mishra Algorithm	better detection rate for encrypted virus	runtime performance unclear and procedure poorly described
Imran et al. [30]	system call	Maximum likelihood	Detection accuracy on real malware	not metamorphic malware; high FPR
Thunga et al. [31]	opcode n-gram	TFIDF feature Selection	High detection of real Metamorphic malware	Parallel comparison lack runtime evaluation
Prasad et al. [34]	Opcode n-gram	TF, TFIDF, multiple HMM	Metamorphic malware classification	Runtime performance unclear
Gharached et al. [32]	opcode sequence	sliding window approach	faster and higher detection rate than [24][27]; detection time is higher than threshold approach	-
Annachhatre et al. [36]	Opcode sequence	*k*-clustering, centroid mean	Improve detection with a classifier	Not metamorphic, runtime performance unclear

VII. DISCUSSIONS

Here we presented the malware detection in HMMs in most recent years. It can be seen that in order to detect metamorphic malware an suitable classifier embedded with HMM could be a possible way to detect this type of malware. The reason would be, metamorphic malware carry obfuscated code with malicious behavior. One could identify and eliminate the obfuscated part of the code before detection, or, study the characteristic of malware family based on its generator and perform detection. This incur how to determine appropriate feature which can represent the target file accurately. Either way of this, runtime performance shall also be concerned as time is critical during analysis for zero day attack.

VIII. CONCLUSIONS

In this paper, we briefly surveyed the different techniques of malware detection in HMM. The future trend would be a more efficient of techniques to increase the accuracy of detection of metamorphic malware. Even though metamorphic malware is hard to write, with the support of many free metamorphic malware generator available online, one still can create abundant metamorphic malware easily. To detect metamorphic malware, according to the reviews aforementioned, malware classification shall be conducted first followed by detection phase. Different features selection can affect the analysis speed and detection rate in HMM. Therefore, comparison of choosing different features shall be investigated. Nevertheless, a combination of features shall be considered for better detection. From this study, we have seen that HMM is an effective model for malware detection. Further research shall concentrate on how to reduce overhead cost by HMM at the same time achieve low FPR and high detection rate.

REFERENCES

[1] (2016) Statistical portal website. [Online]. Available: http://www.statista.com/statistics/193444/financial-damage-caused-by-cyberattacks-in-the-us/

[2] K. Haley. (2016) Internet security threat report: Attackers are bigger, bolder, and faster. [Online]. Available: http://www.symantec.com/connect/blogs/2015-internetsecurity-threat-report-attackersare-bigger-bolder-and-faster

[3] (2016) Internet security threat report. mountain view: Symantec. [Online]. Available: https://www.symantec.com/content/dam/symantec/docs/reports/istr-21-2016n.pdf?aid=elq_9562&om_sem_kw=elq _16172640&om_ext_cid=biz_email_elq_

[4] P. Szor and P. Ferrie. (2003) Hunting for metamorphic, symantec security response. [Online]. https://www.symantec.com/avcenter/reference/hunting.for.metamorphic.pdf

[5] S. Alam, R. N. Horspool, I. Traore, and I. Sogukpinar, "A framework for metamorphic malware analysis and realtime detection," *Computers & Security*, vol. 48, pp. 212–233, 2015.

[6] E. Konstantinou and S. Wolthusen, "Metamorphic virus: Analysis and detection," *Royal Holloway University of London*, vol. 15, 2008.

[7] A. Walenstein, R. Mathur, M. R. Chouchane, and A. Lakhotia, "The design space of metamorphic malware," in *2nd International Conference on i-Warfare and Security (ICIW 2007)*, pp. 241–248.

[8] J.-M. Borello and L. M´e, "Code obfuscation techniques for metamorphic viruses," *Journal in Computer Virology*, vol. 4, no. 3, pp. 211–220, 2008.

[9] (2016) Malware analysis basics - part 2. [Online]. Available: http://resources.infosecinstitute.com/malware-analysis-basicdynamic-techniques/#article

[10] A. Moser, C. Kruegel, and E. Kirda,"Limits of static analysis for malware detection," in *ACSAC 2007.Twenty-third annual*, pp. 421–430.

[11] D. Cornel. (2008) Static analysis techniques for testing application security. [Online]. Available: http://www.denimgroup.com/media/pdfs/DenimGroup_StaticAnalysisTechniquesForTestingApplicationSecurity_OWASPSanAntonio_20080131.pdf.

[12] G. Jacob, H. Debar, and E. Filiol, "Behavioral detection of malware: from a survey towards an established taxonomy," *Journal in computer Virology*, vol. 4, no. 3, pp. 251–266, 2008.

[13] Y. Fukushima, A. Sakai, Y. Hori, and K. Sakurai, "A behavior based malware detection scheme for avoiding false positive," in *Secure Network Protocols (NPSec), 2010 6th IEEE Workshop* on, pp. 79–84.

[14] N. Idika and A. P. Mathur, "A survey of malware detection techniques," *Purdue University*, vol. 48, 2007.

[15] D. Harley and A. Lee. (2007) Heuristic analysis–detecting unknown viruses. [Online]. Available: http://www.eset.com/download/whitepapers/HeurAnalysis

[16] M. G. Schultz, E. Eskin, F. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables," in *Security and Privacy, 2001. S&P 2001. Proceedings.* 2001 IEEE Symposium on, pp. 38–49.

[17] R. Moskovitch, C. Feher, N. Tzachar, E. Berger, M. Gitelman, S. Dolev, and Y. Elovici, "Unknown malcode detection using opcode representation," in *Intelligence and Security Informatics*, 2008, pp. 204–215.

[18] J. Kuriakose and P. Vinod, "Metamorphic virus detection using feature selection techniques," in *Computer and Communication Technology (ICCCT), 2014 International Conference on*, pp. 141–146.

[19] D. Bilar, "Opcodes as predictor for malware," *International Journal of Electronic Security and Digital Forensics*, vol. 1, no. 2, pp. 156–168, 2007.

[20] G. Canfora, F. Mercaldo, C. A. Visaggio, and P. Di Notte, "Metamorphic malware detection using code metrics," *Information Security Journal: A Global Perspective*, vol. 23, no. 3, pp. 57–67, 2014.

[21] J. Xue, C. Hu, K. Wang, R. Ma, and J. Zou, "Metamorphic malware detection technology based on aggregating emerging patterns," in *Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human*, 2009, pp. 1293–1296.

[22] J. Lee, K. Jeong, and H. Lee, "Detecting metamorphic malwares using code graphs," in *Proceedings of the 2010 ACM symposium on applied computing*, pp. 1970–1977.

[23] P. Vinod, V. Laxmi, M. S. Gaur, G. Kumar, and Y. S. Chundawat, "Static cfg analyzer for metamorphic malware code," in *Proceedings of the 2nd international conference on Security of information and networks*, 2009, pp. 225–228.

[24] W. Wong and M. Stamp, "Hunting for metamorphic engines," *Journal in Computer Virology*, vol. 2, no. 3, pp. 211–229, 2006.

[25] W.Wong, "Analysis and detection of metamorphic computer viruses," M Sci. thesis, San Jose State University, California, USA, May 2006.

[26] P. Desai, "Towards an undetectable computer virus," M Sci. thesis, San Jose State University, California, USA, December 2008.

[27] T. H. Austin, E. Filiol, S. Josse, and M. Stamp, "Exploring hidden markov models for virus analysis: a semantic approach," in *System Sciences (HICSS), 2013 46th Hawaii International Conference* on, 2013, pp. 5039–5048.

[28] A. Kalbhor, T. H. Austin, E. Filiol, S. Josse, and M. Stamp, "Dueling hidden markov models for virus analysis," *Journal of Computer Virology and Hacking Techniques*, vol. 11, no. 2, pp. 103–118, 2015.

[29] F. Rezaei, M. K. Nezhad, S. Rezaei, and A. Payandeh, "Detecting encrypted metamorphic viruses by hidden markov models," in *Fuzzy Systems and Knowledge Discovery (FSKD), 2014 11th International Conference* on, pp. 973–977.

[30] M. Imran, M. T. Afzal, and M. A. Qadir, "Using hidden markov model for dynamic malware analysis: First impressions," in Fuzzy *Systems and Knowledge Discovery (FSKD), 2015 12th International Conference* on, pp. 816–821.

[31] S. P. Thunga and R. K. Neelisetti, "Identifying metamorphic virus using n-grams and hidden markov model," in *Advances in Computing, Communications and Informatics (ICACCI), 2015 International Conference on*, pp. 2016–2022.

[32] M. Gharacheh, V. Derhami, S. Hashemi, and S. M. H. Fard, "Detection of metamorphic malware based on hmm: A hierarchical approach.," *International Journal of Intelligent Systems & Applications*, vol. 8, no. 4, 2016.

[33] S. Alam, I. Sogukpinar, I. Traore, and R. N. Horspool, "Sliding window and control flow weight for metamorphic malware detection," *Journal of Computer Virology and Hacking Techniques*, vol. 11, no. 2, pp. 75–88, 2015.

[34] T. S. Prasad and N. R. Kisore, "Application of hidden markov model for classifying metamorphic virus," in *Advance Computing Conference (IACC), 2015 IEEE International*, pp. 1201–1206, IEEE, 2015.

[35] (2016) Malware in cybercrime. [Online]. Available: http://http://malicia-project.com/

[36] C. Annachhatre, T. H. Austin, and M. Stamp, "Hidden markov models for malware classification," *Journal of Computer Virology and Hacking Techniques*, vol. 11, no. 2, pp. 59–73, 2015.

