Chapter 10
Similarity Measure for Obfuscated Malware Analysis

P. Vinod
SCMS School of Engineering and Technology, India

P. R. Rakesh
SCMS School of Engineering and Technology, India

G. Alphy
SCMS School of Engineering and Technology, India

ABSTRACT

The threats imposed by metamorphic malware (capable of generating new variants) can easily bypass a detector that uses pattern-matching techniques. Hence, the necessity is to develop a sophisticated signature or non-signature-based scanners that not only detect zero day malware but also actively train themselves to adapt to new malware threats. The authors propose a statistical malware scanner that is effective in discriminating metamorphic malware samples from a large collection of benign executables. Previous research articles pertaining to metamorphic malware demonstrated that Next Generation Virus Kit (NGVCK) exhibited enough code distortion in every new generation to defeat signature-based scanners. It is reported that the NGVCK-generated samples are 10% similar in code structure. In the authors' proposed methodology, frequencies of opcodes of files are analyzed. The opcodes features are transformed to new feature spaces represented by similarity measures (37 similarity measure). Thus, the aim is also to develop a non-signature-based scanner trained with small feature length to classify unseen malware and benign executables.

INTRODUCTION

Malware is generic term that refers to software which does undesired malicious activity in computer systems. As the amount of available data multiply, the problem of managing the information turn out to be more difficult. The increased use of internet file sharing has led to wide spread of malware. The consequence of this prevalence is that many computer systems are vulnerable and are infected with malicious programs. Zero day attacks cause great destruction to the computer world. Apart from the new attacks, existing malware threats are tansformed into new ones. Malware detectors have not evolved to mitigate sophisticated attacks.

DOI: 10.4018/978-1-4666-6158-5.ch010

Therefore, there is a need urgent need to develop robust detector which can identify not only the existing malwares but also unseen and obfuscated malwares. Static signature based detection technique has been a dominant within Antiviruses. Many writers make use of virus constructors for developing new malware. These malware kits allow hackers to generate new malware specimens with minimal knowledge. Some of the tools are Next Generation Virus Kits (NGVCK), G2, Mass Code Generation (MPCGEN), Virus Creation Lab (VCL 32) etc. We consider variants of a malware from various tools like NGVCK, G2 etc. Earlier studies have already shown that NGVCK tool provides enough obfuscation in subsequent malware generations. Thus, traditional signatures can prove to be ineffective when dealing with unknown variants. In this work, we propose static analyzing technique to extract relevant features from malware. Based on these relevant features we determine similarity amongst pair of files using 37 similarity measurement indices. In an effort to classify the samples, the objectives are as stated below:

1. To find whether various distance/similarity measures are effective in classification of malware and benign executables.
2. To evaluate the effectiveness of virus generation tool kits like NGVCK, G2 etc in the generation of strong metamorphic variants.
3. To evaluate the effects of feature reduction in classification accuracy.
4. To find which classifier produces better result for the test/train model.
5. To find actegory of features that contribute to the detection scheme.
6. To evaluate response of the proposed detector on imbalanced dataset.

In the remaining part of this chapter we briefly introduce different types of malware. Obfuscation methods adopted by metamorphic engine to generate strong metamorphic malware is also discussed. Subsequently, we discuss malware detection techniques proposed by researchers to identify metamorphic malware. Also, we introduce machine learning methods for the detection of malicious code. Later part of this chapter we discuss our proposed methodology based on similarity measurement indices. Finally experiments, results, inferences are covered. We close the chapter with pointers to future research in the challenging domain of desktop and mobile malware detection.

MALWARE AND DETECTION TECHNIQUES

Malicious programs have a long history and ever since the invention of malicious programs their detection have attracted the anti-malware community. Malicious programs aimed at detecting useful information confined to system and users, they remain dormant and undetected. Anti-detection mechanisms have evolved into complexity. However, sophisticated malicious program also known as *polymorphic* and *metamorphic* viruses have evolved in unprecedented rate. Detection of malcode that can obfuscate or morph future instances is a big challenge and needs to be addressed critically. In the following subsection, we discuss different types of malware with their detection techniques.

Malware

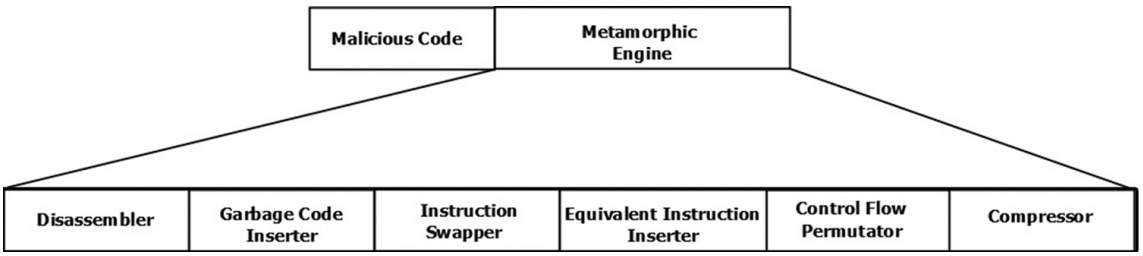
Malware or unwanted software's are the program created to compromise normal functionality of computer, gather sensitive information by gaining super user privilege and bypass access control. Malicious programs prevails itself in numerous forms such as codes, scripts, active contents and other software. Often, malware is confused as defective codes (also known as bug), whereas malware is skillfully implanted to disrupt the functioning of the host program.

Types

The most common types of malware are:

1. **Virus:** Virus is a piece of code that propagates from one system to other, infecting each system on its way. When an infected code is launched in a vulnerable machine it utlizes this machine to create a denial of service on the system.
2. **Worms:** Are *self-replicating* and independent codes that affects computer by exploiting operating systems vulnerability. In order to propagate in the network they take advantage of the information and file transport making it a network aware.
3. **Trojan Horses:** These malware are usually disguised as a legitimate applications. They appear in some eye catching program unknowingly downloaded by the users. The consequences range from deleting, stealing sensitive data or generating unwanted pop ups.
4. **Spyware:** Spy on user activity to collect keystrokes, harvest data, modify security settings of software or browser to interfere with unethical network connections.
5. **Adware:** Is a type of malware that is capable of delivering advertisements. These are often bundled with spyware that is capable of tracking user activity for stealing information.
6. **Encrypted Malware:** Encrypts the malware payload. The malicious code is encrypted using *block* or *stream ciphers* methods that thwart scanners employing pattern matching techniques. In this type of malicious code the decryption routine are constant and can be finally detected using signature based detectors.
7. **Polymorphic Malware:** Use a *polymorphic engine* to mutate syntactic structure of new variant while keeping the original algorithm intact. That is, the code changes itself each time it executes, but the function of the code (its semantics) does not change. More importantly, polymorphic malware have variable decryption routine. Thus, emulation based scanner can sometime be effective in their detection. However, limitation is to create a ideal emulator.
8. **Metamorphic Malware:** Metamorphic viruses can obfuscate their body by various techniques. The most applicable techniques are (refer Figure 1):
    a. **Register or Variable Exchange:** Use of different registers/variables in newly produced instance.
    b. **Instruction Substitution:** Replacement of instructions with their possible equivalents instructions blocks or subroutines.
    c. **Code Transposition:** Code and subroutines reorder or change of the flow control by using conditional or unconditional jumps.
    d. **Instruction Permutation:** Reorder independent instructions.

Figure 1. Metamorphic engine



e. **Garbage Code Insertion:** NOP instruction and other instruction that has no effects on the operation of the code are inserted in the original program instructions.

Malware Detector

Malware detectors protect the system against the malicious code infection. Malware scanners may be configured to reside on the same computer it is trying to protect or could exist in an external computer. These detectors contain the knowledge of the malware (in form of signature or behavior) stored in the repository to flag any monitored samples as suspicious. According to Kindsight Security Labs Malware survey Report 2012, in fixed broadband deployment, 13% of home networks were found to be infected of which 6% are infected by high threat level malware such as botnets, rootkits or banking trojans, 7% of households are infected with a moderate threat level malware such as spyware, browser hijackers or adware. The Plankton-Apperhand-Tonclank family, which was reported as malware in 2011, have now been downgraded by the mobile security industry to "*aggressive adware*". Android infection rate has increased to an alarming rate in the order of 1 in 1000 devices which include the top android malware like Trojan.GGTracker, Trojan. Pjapps3.A, Adware.SndApp.B.

Sophos Security Threat Report 2012 anticipates growing sophistication in web-borne attacks, even broader use of mobile and smart devices, and rapid adoption of cloud computing bringing new security challenges. It reported United States (11.43%), India (8.02%), Russian Federation (7.52%), Brazil (5.62%), Italy (3.37%), Vietnam (3.07%), Indonesia (2.88%), Taiwan (2.85%), Ukraine (2.82%), Romania 2.64%), France (2.25%) as the top 12 spam producing countries in the world. Consequently, malware detectors with very high detection and robust techniques rates need to be devised.

Malware Detection Techniques

- **Anomaly Based:** In this approach the valid behavior of the system is studied and approximated to build the repository that has to be compared with the potentially unwanted infectors.
- **Signature Based:** Characteristics of the viruses are investigated and they are stored in the form of MD5 hashes for detection. Mostly, useful to identify known malware attacks.
- **Heuristic Bases:** Identifies the maliciousness by monitoring the behavior of software. Such detectors have high false alarms if heuristics are not well defined.
- **Behavior Based:** This approach inspect at how software would behave to determine hostile performance.
- **Integrity Check:** Detects the changes in files by comparing them to their original check-sum stored in a list of check sum table prepared in non-infection mode.

Malware Detection Methods

As malwares evolved in complexity, detection techniques have emerged in its sophistication. Several detection methods/models is proposed to effectively identify and classify zero day malware. Following subsection introduce various detection methods.

Statical Analysis

Static analysis is a process of analyzing the code before the execution in a system to classify samples as benign or malware. This method use the syntax or structural properties of the program to determine maliciousness.

Approach use varied features of malware files such as opcode, portable executable metadata, strings, function code length and/or Hex patterns. Tools of the trade are decompilers, disassemblers, source code analyzers, and even basic utilities

such as *diff* or *grep* command. Detectors based on static analysis aim to develop a system by reverse engineering unknown binary executable code without the need for manual inspection of assembly code.

On successfully reversing malware, the analyst can understand malware infection life cycle. Static analysis has an advantage that it reveals how a program could behave under unusual conditions, as one can examine segment of the program that normally are not executed. Also, it is fast, as the detector fully predict the behavior of all but the smallest programs. However, it has the disadvantage of using disassemblers for analysis.

Dynamic Analysis

Dynamic analysis require the samples to be exeuted in an controlled environment to learn the behaviour of specimens. However, with dynamic analysis complete execution flow of a program is approximated to understand its nature. Certain malware that exhibit multiupersonality behaviour leads to increased levels of false alarms. As, the runtime behaviour of samples are used in dynamic analysis to determine maliciousness it incur high processing overhead. Therefore sometimes, static analysis is preferred, to understand in the initial phase of detection whether the sample is suspicious.

METAMORPHIC MALWARE DETECTION METHODS

Earlier research work in the area of metamorphic malware detection is introduced in (Vinod et. al, 2011a). They also discuss *longest common subsequence* techniques for detecting synthetic metamorphic malware where the extracted opcode sequence are used to create control flow graphs (CFG). In (Wong W, 2006) the author investigate the degree of metamorphism in different malware constructor like Generation 2 (G2), Virus Creation Lab (VCL32), MPCGEN (Mass Code Generator), Next Generation Virus Creation Kit (NGVCK) obtained from (http://vx.netlux.org). They demonstrated that NGVCK generated variants depicted average similarity of 10%. However, non-malware files exhibited similarity close to NGVCK. Thus, to detect metamorphic malware variants created with NGVCK tool, Hidden Markov Model (HMM) based scanner was developed. Samples which could not have been detected with commercial antiviruses where however identified by their proposed HMM detector.

Authors in (Vinod et al, 2010a), proposed a method for detecting unseen malware samples by executing programs using STraceNTx in an emulated environment. Samples were generated with different malware kits (VCL32, IL_SMG, PS-MPC, MPCGEN and NGVCK). Common base signature from each malware constructor was developed. It was observed that unseen samples were detected using the base signature. Later, they investigated the degree of metamorphism amongst different constructors. Similarity amongst malware constructors were determined by computing proximity index. *Inter constructor* and *intra constructor* proximity was also determined. The research exhibited that all constructor demonstrated high *intra constructor* proximity except NGVCK. Unlike, NGCVK constructor, code distortion with other malware kits were not significant, indicating minimal degree of metamorphism. Also, *inter constructor* similarity was determined. Most of the samples were found to have higher proximity with other malware constructor. However, NGVCK generated variants depicted less *intra constructor* proximity. This indicated that metamorphic engine of NGVCK virus kit was robust in comparison to other malware constructor.

Authors (Attaluri, S, 2009) used opcode alignment sequences obtained from morphed malware, to create stronger detector based on Profile Hidden Markov Models (PHMMs), for the classification of unseen synthetic malware specimens. PHMMs are robust in comparison to standard HMM as the former capture positional information. The

generated PHMM was tested on metamorphic malware samples generated from three malware constructors (VCL-32, PS-MPC and NGVCK). Detection rate of 100% was achieved for VCL-32 and PS-MPC samples, whereas, considering proper multiple aligned sequences of NGVCK malware, a better detection rate was obtained. The study also highlighted that PHMMs were not effective if code blocks were permuted. Under such scenario the standard HMM resulted in better accuracy. However, the primary disadvantage with PHMM is that they are computationally expensive for modeling malware.

Vinod et al, 2009, developed three malware samples and seven variants from the base malware by applying different code morphing technique (*equivalent instruction substitution*, *garbage insertion*, *code permutation using branch instruction*). Structure of variants were normalized with a custom built assembly code normalizer. Subsequently, normalized control flow graph (CFG) was constructed. Later, from this normalized CFG, sequence of opcode were extracted and compared with the opcode sequence of other variants using *Longest Common Subsequence (LCS)*. It was examined that variants of same base malware depicted higher similarity. Moreover, morphed malware could be differentiated from benign files. The only drawback of using LCS approach is that it has computational complexity of $O(m.n)$ where, m *and* n are the length of two opcode sequences extracted from malware/benign .asm files.

Authors in (Lin D., 2011) developed a metamorphic engine that could evade HMM based detector. The idea of the study was to investigate as to what extent the base malware could be morphed so as to fail HMM based scanners. The metamorphic engine morphed the NGVCK generated malware code using elementary code obfuscation techniques. More importantly, the basic idea was to generate morphed malware copies that are structurally similar to benign files. In order to carry the experiment Cygwin executables were considered as the benign samples (as they employ low level system functionality as malware specimens) and code fragment of variable proportion from benign subroutines were extracted, which were subsequently inserted in malware samples keeping in view that maliciousness is not disturbed. The experiment was conducted for different fraction of injected code (extracted from benign files) in range of 5% to 35%. It was observed that even 5% of block of benign fragment inserted as dead code in malware sample could thwart HMM detector. This research article opened a new direction for malware analyst to understand that the metamorphic engine employing basic obfuscation method as *insertion of dead code* (from subroutine of benign files) could defeat both pattern based as well as spectrum based scanners.

In (Runwal N, 2012), authors devised a method for determining similarity between the files using opcode graphs (directed graph). An opcode graph is constructed from opcode extracted from malware assembly files. Nodes are denoted as opcode and the directed edge between the nodes are represented as the probability of a successor opcode in a file with respect to a given opcode under consideration. The study showed that the opcode graph similarity method outperformed HMM based detector and yielded no false positives or negatives. As there existed good separation between the similarity scores of malware v/s malware and benign v/s benign. Hence, a threshold could be easily established. Using this threshold, unseen samples can be classified as malware/ benign. The authors also tested the effectiveness of the opcode graph model based on two attacks (a) removal of rare opcode and (b) transforming base malware into complex form (where junk code from benign samples are injected into malware samples so that malware and benign samples appear structurally similar). The investigation demonstrated that metamorphic malware could be discriminated from benign executable even after the elimination of uncommon opcode. Also, the detection rate of HMM scanners and opcode graph based detector are comparable if benign code is injected in malicious files as dead code.

Studies in (Lin D., 2011) depicted that HMM based detector failed to detect strong metamorphic malware invariants injected with benign opcode as dead code. Annie H, 2012, devised a method to improve the detection rate of HMM scanner by combining it with statistical methods such as *chi-squared test*. It was observed from the experiments that HMM detectors performed better if short sequence of benign opcode were added as junk code. Moreover, for a block of benign code added to malware file the detection rate of HMM based scanner degraded. Thus, *chi-square* based classifier was developed. This hybrid model devised by integrating HMM and *Chi-square* classifier resulted in improvement of classification accuracy.

Authors in (Donabelle B, 2013) proposed a method for identification of metamorphic malware based on byte features, using structural entropy and wavelet transform techniques. Image transformation methods were applied on malicious files to segment them using different scaling parameter. Subsequently, it was noticed that the scaling parameter with values 3 and 4 segmented malware files appropriately. The segmented files were then compared based on edit distance. The entropy score depends on the length of malware files. Hence, in some cases the detector did not perform well as the NGVCK generated malware differed largely with respect to its length.

Vinod et al, 2012b, created a probabilistic signature for the identification of metamorphic malware inspired by bio informatics multiple sequence alignment method (MSA). For each malware family, single, group and MSA signature was developed. Threshold were determined for 37 malware families. The results depicted promising outcome when compared to the detection rate achieved with 14 commercial antivirus scanners. Their study showed that the signatures generated using sequence alignment method were far superior in comparison to those generated by commercial AV. The proposed detector was found to have better detection rate (73.2% approximately) and was ranked third best compared to other commercial malware scanners.

DATA MINING TECHNIQUES FOR MALWARE DETECTION

The authors in (Md.Enamul, K, 2005) proposed a *"phylogeny"* model, particularly used in areas of bioinformatics, for extracting information in genes, proteins or nucleotide sequences. The *n-gram* feature extraction technique was proposed and fixed permutation was applied on the code to generate new sequence, called *n-perm*. Since new variants of malware evolve by incorporating permutations, the proposed *n-perm* model was developed to capture instruction and block permutations. The experiment was conducted on a limited data set consisting of 9 benign samples and 141 worms collected from VX Heavens (http://vx.netlux.org). The proposed method showed that similar variants appeared closer in the phylogenetic tree where node represented a malware variant. The method did not depict how the *n-perm* model would behave if the instructions in a block of code are replaced by equivalent instructions which could either expand or shrink the size of blocks (with respect to number of instructions in a block).

The authors (Schultz et al, 2001) introduced malware detection using data mining method. Three types of features were extracted: *PE Header*, *strings*, and *byte sequence*. To detect unknown computer viruses, classifiers like *decision tree*, *Naive Bayesian network* and *RIPPER* were used. The results demonstrated that the machine learning method outperformed signature based techniques. The *boosted decision trees* performed better with respect to classification and an area of 0.996 under the ROC curve was obtained.

Jeremy et al, 2004, extracted *n-grams* of byte from a collection of 1971 benign and 1651 malware samples. Top 500 *n-grams* were selected, and evaluated on various inductive methods like Naive Bayes, decision trees, support vector machines and boosted versions of these classifiers. The authors indicated that the results obtained by them was better compared to the results presented by Schultz et al, 2001. Better classification accuracy was obtained with boosted decision trees

with area an under ROC curve of 0.996. In an extension to their study, the authors evaluated the effectiveness of earlier trained malware and benign models with new test data. It was noticed that the trained model were not good enough to identify unknown samples collected from the point after the training model was prepared. This study suggested that the training models also require updation for identification of unknown malware samples.

Dima S. et al 2009, proposed a method for malware detection using the opcode sequence obtained by disassembling executables. Opcode *n-grams* were extracted for *n = 1, 2,…., 6* from a large collection of malware and benign samples. Top *n-gram* was considered using three feature selection methods: *Document Frequency, Fischer Score* and *Gain ratio,* which finds prominence in text categorization. The frequency of terms in documents were computed and vector space model was represented for each sample. The experiment was evaluated on different *Malicious File Percentage (MFP)* levels to relate to real scenario. It was shown that *2-gram* features outperformed all *n-grams* models. The feature selection method like *Fischer score* and *Document Frequency* were found to be better as compared to Gain Ratio. Top 300 opcodes *n-gram* was found to be effective in classification of the executables with an accuracy of 94.43%. The Boosted decision tree, Artificial Neural Network (ANN) and decision tree produced low false positives compared to Naive and Boosted Naive Bayes. Also, the test set with MFP level of 30% was found to be better compared to other MFP levels with accuracy of 99%.

(Menahem et al, 2009a) proposed a method for improving malware detection using the ensemble method. Each classifier use specific algorithm for classification and each classifier is suited for a particular domain. In order to achieve higher detection rate using machine learning technique, the authors combined the results of individual classifier employing methods like weighted voting, distribution summation, Naive-Bayes combination, Bayesian combination, stacking and Troika (Menahem et al, 2009b). The goal of this research was to evaluate whether ensemble based method would produce better classification accuracies as compared to individual classifier. Since ensemble methods require high processing time, the experiment was performed on a part of initially collected data set (33%). From their experiments, it was observed that the Troika and Distribution function were found to be the best methods with accuracy of 95% and 93% respectively but suffered with high execution time overheads. All ensemble methods that outperformed in accuracies suffered in execution time. PE file header information, byte *n-grams* and function based features were extracted and binary, multiclass classification was performed. Troika was found to perform better for multiclass classification followed by Bayesian-combination. The authors suggest that since execution time is of prime concern, Bayesian combination would be better choice for ensemble based classification of malware and benign instances.

Non-signature based method using Self-organizing maps (SOMs) was proposed in (Seon, Y., 2006). SOM is a feed-forward neural network for topological representation of the original data. This technique is used to identify files infected by malware. Using SOM, it was observed that the infected files projected a high density area. The experiment was conducted on 790 files infected with different malware. The proposed method was capable of detecting 84% of malware with false positive rate of 30%.

The authors in (Santos I, 2009) extracted *n-grams* from benign and malicious executables and used *k-nearest* neighbour algorithm to identify the unseen instances. In this method *n-grams* from malware and benign samples were extracted for *n = 2, 3, 4,…., 8* to form the training and test set. The number of coincidences of *n-grams* in the training set and test instance was determined. An unknown sample was considered as malware only if the difference of *k* most similar malware and benign instances was greater than a threshold *d*. The experiment was conducted on a data set collected from a software agency and the results

depict that the detection rate for 2-gram was poor, 91% detection rate was obtained for 4-gram model.

Authors in (Henchiri, O., 2006) presented a method based on byte *n-gram* feature on different families of viruses collected from VX Heavens. For each family of virus an inter-family support threshold was computed, and a maximum of 500 features that have higher frequency than the threshold was retained. Likewise, features which have higher value than the inter-family support were retained and others were eliminated. The results were evaluated on the proposed classifier and compared with traditional classifiers like decision trees, Naive Bayes, and Support Vector Machine. From their experiments, it was observed that shorter byte sequences produced better detection rates between 93% to 96%. The performance degraded with feature length less than 200 features.

A non-signature based method using byte level file content was proposed in (Tabish, S.M, 2009). This method compute diverse features in block-wise manner over the byte level content of the file. Since blocks of bytes of a file was used for analysis, prior information regarding the type of file was not required. Initially, each sample was divided into equal sized blocks and different information theoretic feature selection technique were applied. The feature extraction module extracted 13 different type of features for each *n-gram* (*n = 1, 2, 3, 4*). Thus, for each block 52 features were extracted. The experiments were conducted on malware data set collected from VX Heavens (http://vx.netlux.org) and benign samples consisting of different file types: DOC, JPG, EXE etc. Results depict that the proposed method was capable of achieving an accuracy above 90%.

The authors in (Igor, S., 2010) proposed a novel method for detecting variants of malware making use of opcode sequence. This approach was based on determining frequency of the opcode. The relevant opcodes were mined using mutual information and assigned with certain weights. The proposed method was tested on dataset downloaded from VX Heavens (http://vx.netlux.org) and was not tested for packed malware. The opcode sequence of fixed length *n = 1 or 2* was extracted for each base malware and its variants. Proximity between the variants was computed using *cosine similarity*. The similarity within malware data set was high as compared to benign set. Additionally, the similarity between malware and benign samples was low. Thus, it can be inferred that malware and benign samples are not as diverse as benign files; a characteristics that can be used for discriminating it from benign instances.

Analysis and detection of malware using structural information of PE was proposed by authors in (Merkel, R., 2010). PE header fields were extracted from executables and top 23 features were extracted. The analysis was performed on two test sets: (a) obfuscated and (b) clean malware samples. A hypothesis based classification model was developed and the results were compared with classifiers implemented in WEKA. The detection rate of 92% was reported with obfuscated malware executables.

In their study (Vinod et al, 2011b), different features from both malware and benign files like instruction opcode (consisting of opcode and addressing mode), portable executable header fields and mnemonic *n-grams* was extracted. The primary objective was to understand the degree of obfuscation introduced by malware writers either by modifying the PE header, substituting equivalent opcode or modifying the addressing modes. Features were processed with *scatter criteria* and classified with classification algorithm implemented with WEKA. The experiments revealed that *Random Forest* outperformed all classifier in terms of classification accuracy. Also, accuracy of 98.1% was obtained if classification was performed considering PE Header information. It was noticed that obfuscation was predominantly performed by modifying instruction either by replacing a given opcode with equivalent opcode or modifying the addressing mode of instruction. Therefore, the classification accuracy did not fair well when these features were used for the identification of malicious executables.

Authors in (Vinod et al, 2012a) devised a method for the discrimination of malware samples from large collection of benign executables. *Bi-gram* features were extracted after unpacking packed malware samples, provided by different user agencies making use of signature and dynamic unpackers. Reduced feature were obtained with *Principal Component Analysis* and *minimum redundancy and maximum relevance* techniques. Prominent features were extracted independently from malware and benign population. The objective of study was (a) to determine best feature (b) efficient classifier (c) appropriate feature selection methods and (d) optimal feature vector length that resulted in higher classification accuracy. They also investigated if the classification model performs well for imbalanced data set. Their results were also compared with previous literature and was found to produce detection rate of 97% with overall accuracy of 94.1%.

In (Vinod et al, 2013) proposed a novel method to create multicomponent feature (MCF features) composed of (a) opcode bi-gram (b) PE meta data (c) principal instruction code (opcode and addressing mode) and (d) prominent *uni-grams*. Features were reduced with minimum redundancy and maximum relevance, principal component analysis and extracting eigen vectors. They investigated different features and optimal feature vector length that yielded higher classification accuracy. Experiments were performed on total 2217 malware and 3307 benign PE files. It was observed that mRMR feature selection method produced higher classification accuracy. Overall accuracy of 96.1% was obtained with this proposed technique.

PROPOSED METHOD

Statistical analysis of malwares may be preferred over other methods due to advantages such as reduced memory usage, execution speed etc. Hence, statistical approach on portable executables is adopted in this study. Prior studies have exhibited that *bi-grams* resulted in better accuracy in comparison to other features (Vinod P, 2012a & Vinod P, 2013). Robust similarity measurement indices computed on histogram of opcodes also indicated better classification of unseen executables. Proposed methodology is summarized in Figure 2.

Malwares variants are generated using various tools like NGVCK, G2, VCL32 etc. Benign files are collected from various internet sources. Each benign executable is scanned using commercial anti-virus scanner before they are included in data set. Independent samples are preprocessed and *bi-gram*s are extracted. The two target classes of processed assembly files are then divided into test and train set. The opcode table or feature vector table is constructed and average distances are computed with various distance measures. This FVT is fed into the classifier to create malware and benign models which are subsequently used for predicting unseen samples.

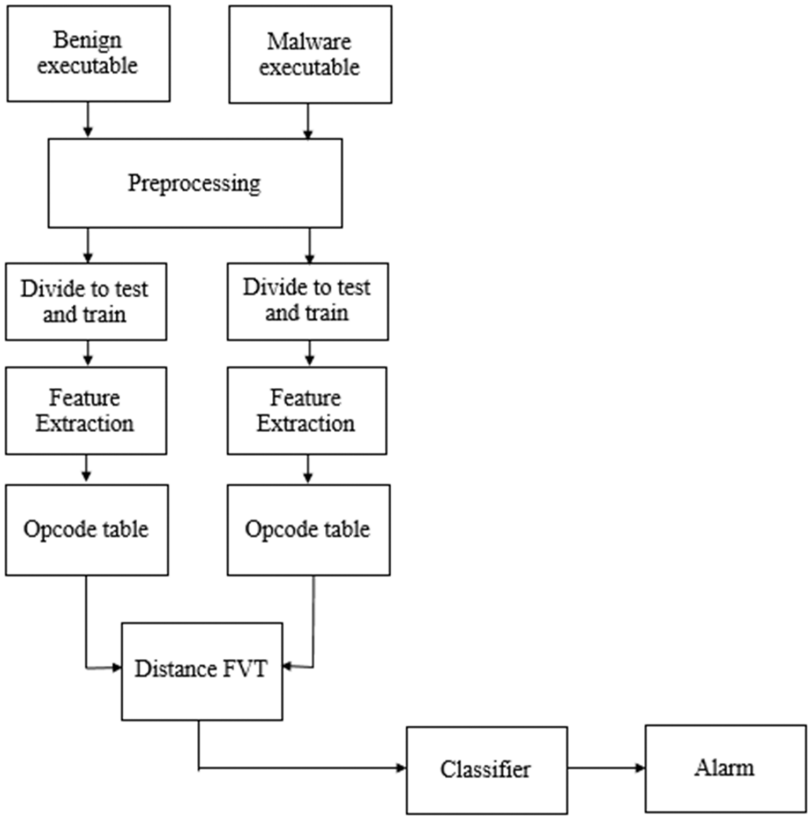
Data Preprocessing

IDA (http://datarescue.com) disassembler is used to generate assembly language code from executable files. IDA Pro is also used as a debugger for Windows PE, Mac OS X, Mach-O, and Linux ELF executable. Assembly codes that are generated with IDA pro disassembler are parsed to extract opcodes from them using a custom parser (refer Figure 3).

It has been argued by researchers that opcodes alone can become good representation of features for identifying malware. Even though, using the entire instruction may lead to precise result, it causes high overhead which can be reduced if opcodes alone are used. So mnemonics or operations are isolated from the operands using a customized parser. For example, MOV is extracted from MOV eax,edx .

Later, from processed assembly files *n-grams* are extracted. *n-gram* are combination opcodes extracted in a sliding window fashion. The *n-gram* having a size of sliding window 1 is called *unigram*, 2 is called *bi-gram* and a size of 3 is

Figure 2. Proposed methodology for malware detection



called as *tri-gram*. We use *n-grams* of size of 2 (*bi-grams*) in all our experiments. The following Table 1 depict *bi-grams* and *tri-gram* from generated from a *unigram*.

Experiment is performed in two phase. In the first phase included the extraction of prominent *bi-gram* features of malware files and subsequently determining the presence of these features in the benign samples. Second phase employed the extraction of features from the benign files and benign *bi-grams* in malware files was estimated.

Divide the Bigram Files into Test and Train

Entire data set is randomly divided into two set parts and train and test sets. Suppose that *x* % samples are considered in train set then *100-x* are used as test files (refer Figure 4).

Extraction of Prominent Features

Unique *bi-grams* appearing in malware/benign training files are determined. From the unique opcode list, a preprocessing operation is performed to extract the opcodes that appear predominantly in at least 40% of the files. This process is depicted in the Figure 5. We assume that this pruned unique list of opcodes acts as the footprint for malware/benign class. This also indicate that these opcodes may depict maliciousness in general. The pruned opcode list extracted from different proportion of training files (malware/benign), refer Table 2.

Opcode Table/Feature Vecttor Table

Entried of *Bigram* in the unique list are used to determine frequency of occurrence of the features in each file belonging to a specific target class.

Figure 3. Block diagram showing the preprocessing phase

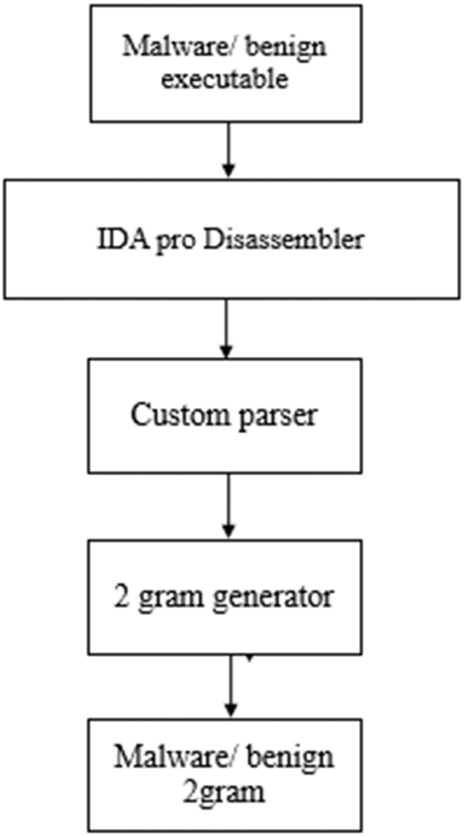
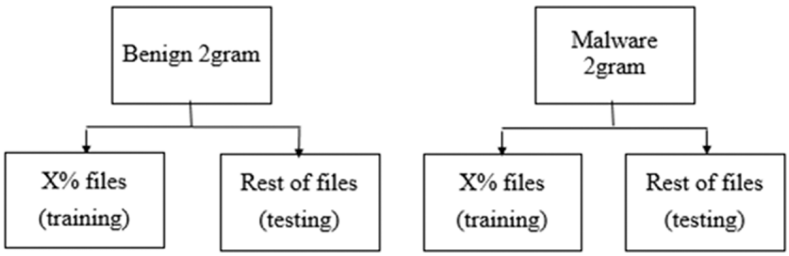


Table 1. Bi-gram tri-gram generation from unigram

Unigram	Bi-Gram	Tri-Gram
mov	movadd	movaddjmp
add	addjmp	addjmpmov
jmp	jmpmov	jmpmovinc
mov	movinc	movincsub
inc	incsub	incsubsub
sub	subsub	subsubjnc

Figure 4. Division of dataset into train and test set



This frequency is recorded in a table also referred to us as opcode table (refer Figure 6).

Subsequently feature vector table is used to compute average distance between each file. We have used 37 distance measures. The average of each distance measure is copied to another table referred to us as distance table. The distance table is shown in Figure 7.

The steps mentioned in section 5.3 and 5.4 is repeated using malware/benign unique list (prominent *bi-grams*) to estimate the frequencies of features in training set of both the target classes and finally a distance table is developed.

Preparation of Distance Feature Vector Table

A FVT of training files is created by copying the distance tables of both training files with features of malware/benign (refer Figure 8). The structure of the distance FVT is as shown in the Figure 9.

Unknown samples (malware/benign) are identified using malware and being features (refer Figure 10). The classification accuracy and values of different evaluation parameters are measured.

Second round of experiment include extraction of features from the training set of target class (benign/malware) to generate two FVTs each for training feature of a specific class. This is done by repeating the processes from feature extraction of to FVT generation. The basic principle behind this is if discriminant feature lead to better accuracy. Also to prune semantically equivalent opcode patterns.

Figure 5. Block diagram showing the feature extraction

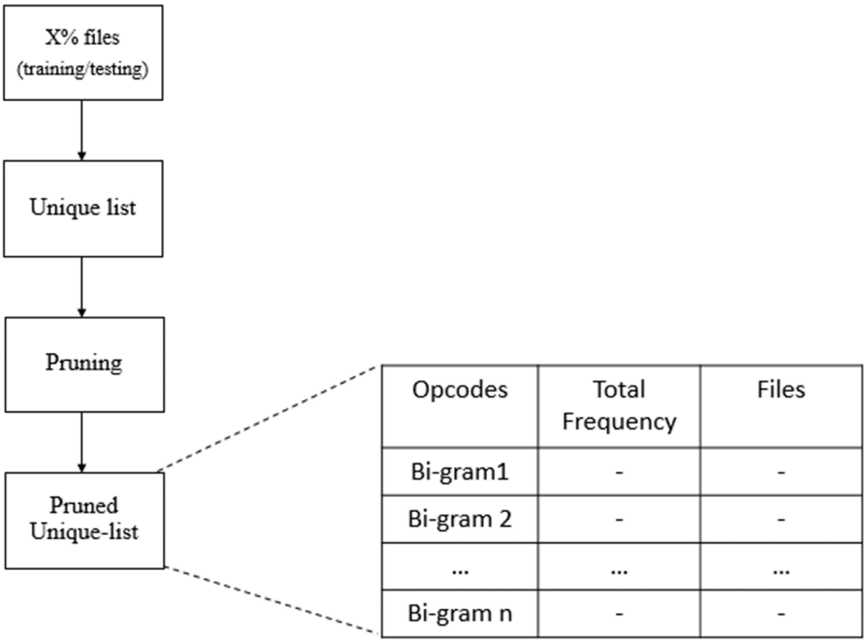
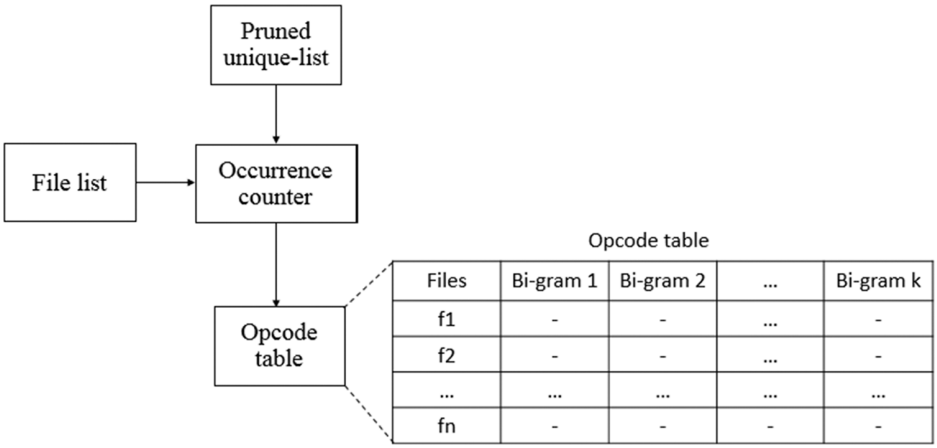


Figure 6. Block diagram showing the opcode table preparation



Classification

The train and test FVT is fed to the various classification algorithms implemented in WEKA to evaluate the performance of trained model. We assume that conventional distance measure cannot detect minor variations in each file, hence we consider 37 distance measures. These distance measures are effective in finding dissimilarities between files.

SIMILARITY MEASUREMENT INDICES

Distance or similarity measures are essential to solve many pattern recognition problems. There are a substantial number of distance/similarity measures encountered in many different fields.

From the scientific and mathematical points of view, distance is defined as a quantitative degree of how far apart two objects are. Synonyms for

Figure 7. Distance table

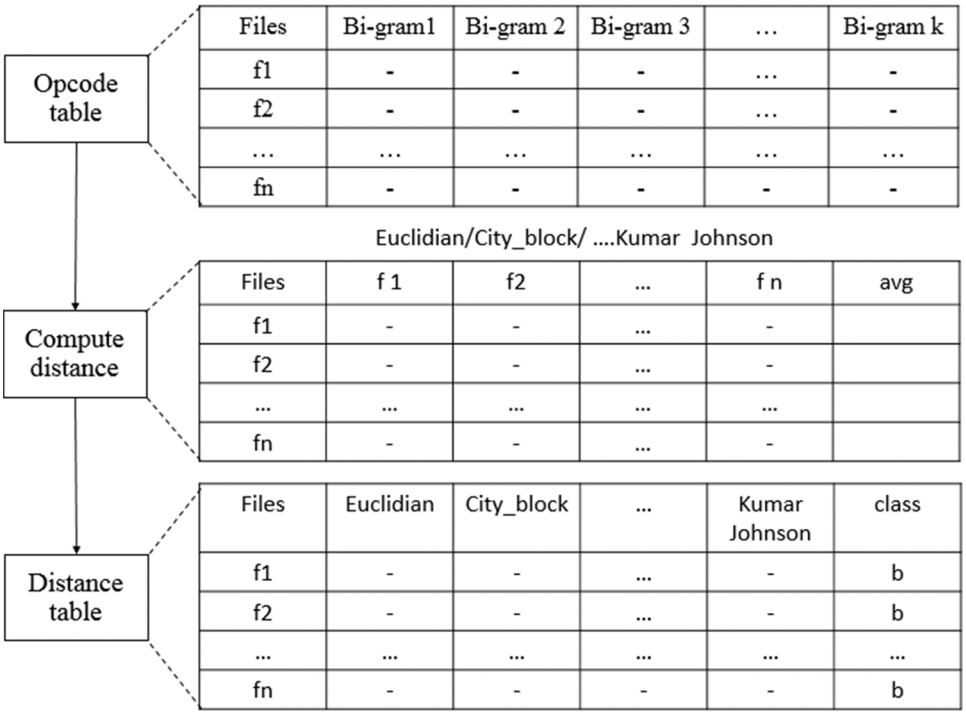


Table 2. Table shows pruned feature count

Percentage Division (train/test)	Class	Number of Entries in Uniquelist	Number of Entries in Pruned Uniquelist
80/20	Benign	10925	325
80/20	Malware	807	101
70/30	Benign	10789	309
70/30	Malware	782	98
60/40	Benign	9327	306
60/40	Malware	757	136
50/50	Benign	10525	349
50/50	Malware	734	139
40/60	Benign	7014	330
40/60	Malware	662	99

distance include dissimilarity and synonyms for similarity include proximity. The distance measures are classified into several families as shown in Figure 11. We have computed 37 distance measure which act as 37 features (S.-H. CHA (2007)) in our experiment (refer Table 3).

EXPERIMENTS AND RESULTS

The NGVCK (Next Generation Virus Creation Kit), G2 (Second Generation virus generator), MPCGEN (Mass Code Generator), VCL32 (Virus Creation Lab for Win32) (http://vx.netlux.org) tools were used to generate obfuscated malware executable from a single malware. These malware files are disassembled using the IDA disassembler to obtain the assembly code. This code is further parsed to obtain the opcode list. This constituted the malware data set for the experiment. The benign files are collected from various internet sources. Dataset constituted of 150 metamorphic malware variants and 800 benign files. The system on which the experiments were carried has 4GB RAM, i5 processor and 500GB hard disk installed on it.

The experiments are carried out using two different way (a) *cross validation* and (b) testing and training method. The cross validation is more efficient for smaller data set as each subset of data is iteratively included in train and test set.

Figure 8. Block diagram depicting FVT generation phase using benign features (training data)

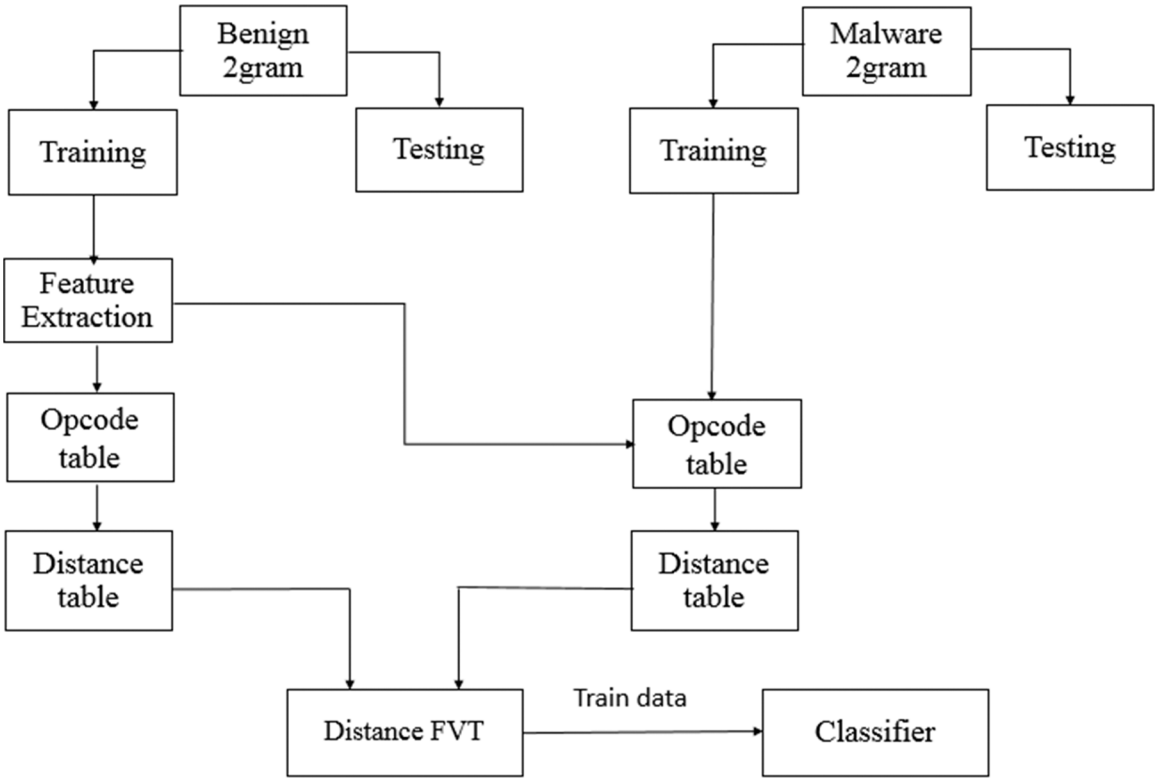
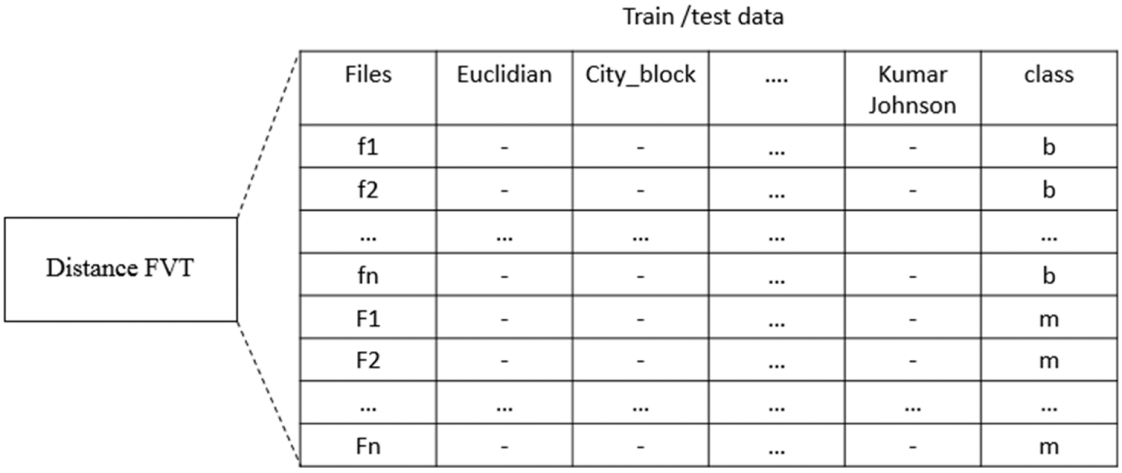


Figure 9. Block diagram showing the structure of distance FVT



The second is the *testing and training* method. In this method features are extracted from training set and unseen samples are classified into either of the target class.

Cross Validation

Cross-validation evaluates classification performance and compares algorithms. The process is executed by dividing data into two parts. One is

Figure 10. Block diagram depicting testing phase (features extracted using benign samples)

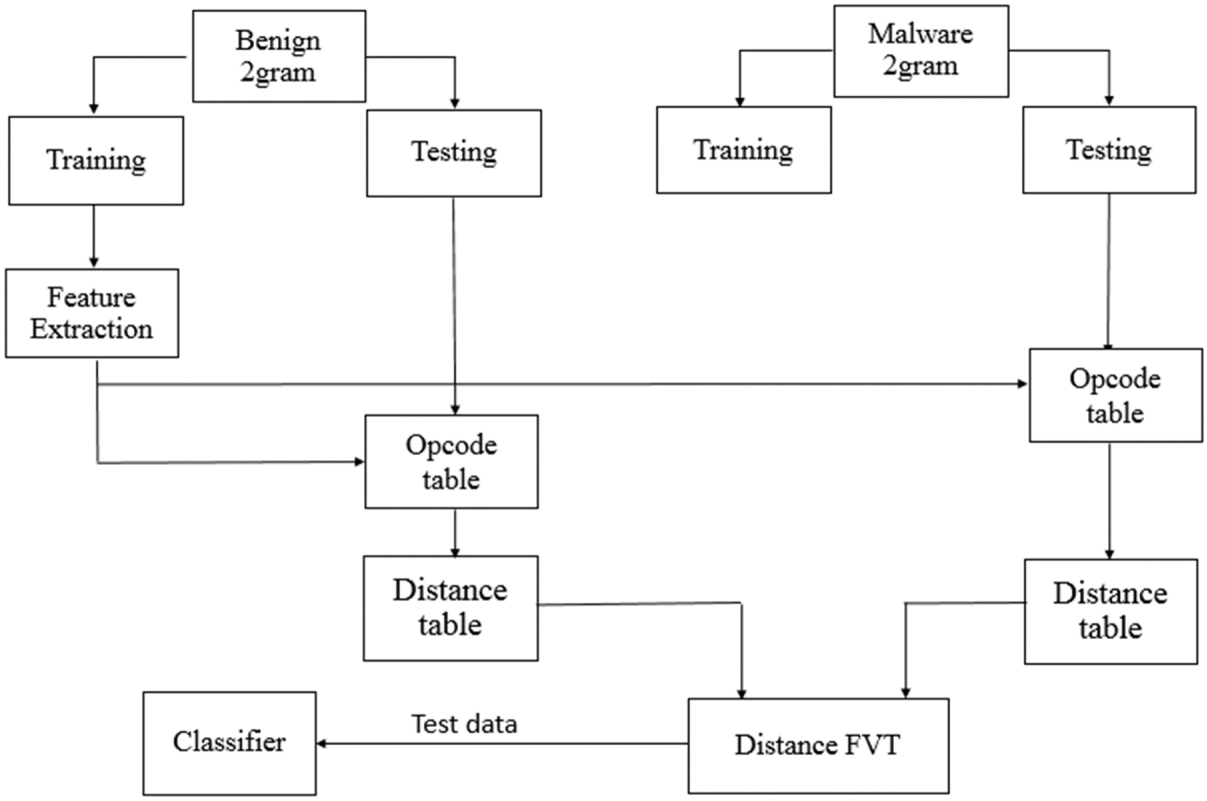


Figure 11. Taxonomy of similarity/distance measurements

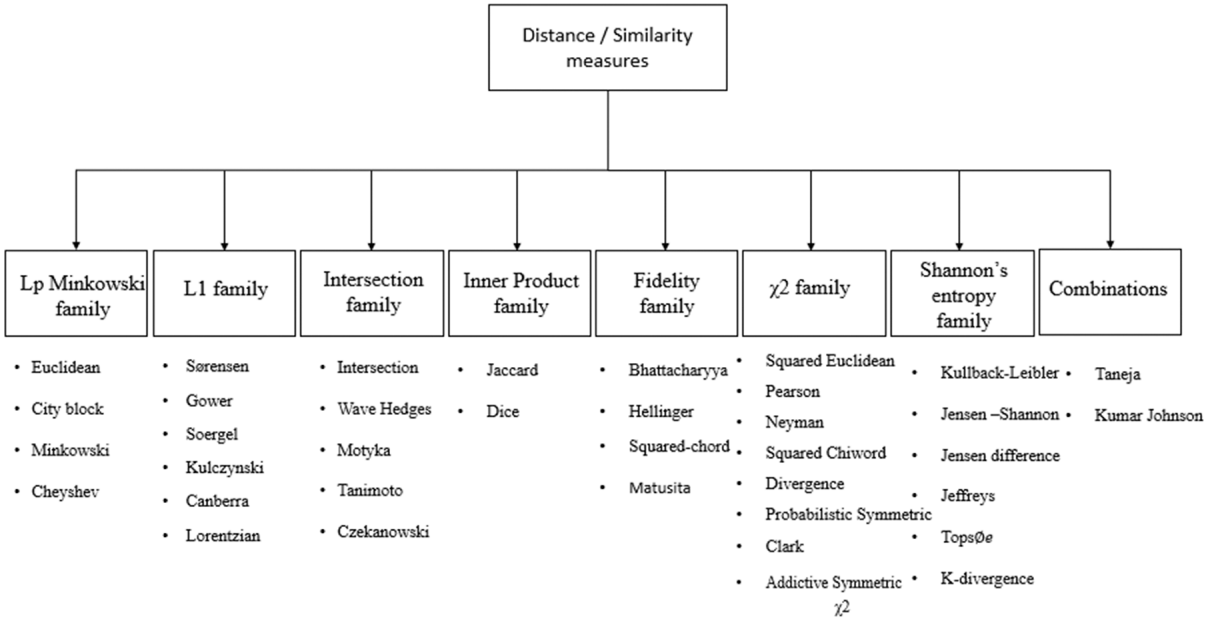


Table 3. Similarity measurement indices

S.No	Distance/Formula	S.No	Distance/Formula
1	Euclidean: $d_{Euc} = \sqrt{\sum_{i=1}^{d} \left\|P_i - Q_i\right\|^2}$	20	Matusita: $d_{M} = \sqrt{\sum_{i=1}^{d} \left(\sqrt{P_i} - \sqrt{Q_i}\right)^2}$
2	City block: $d_{CB} = \sqrt{\sum_{i=1}^{d} \left\|P_i - Q_i\right\|}$	21	Squared-chord: $d_{sqc} = \sum_{i=1}^{d} \left(\sqrt{P_i} - \sqrt{Q_i}\right)^2$
3	Minkowski: $d_{Mk} = \sqrt[P]{\sum_{i=1}^{d} \left\|P_i - Q_i\right\|^p}$	22	Squared Euclidean: $d_{sqe(P,Q)} = \sum_{i=1}^{d} \left(P_j - Q_j\right)^2$
4	Cheyshev: $d_{Cheb} = max\left\|P_i - Q_i\right\|$	23	Pearson $\chi 2$: $d_{p(P,Q)} = \sum_{j=1}^{n} \frac{\left(P_j - Q_j\right)^2}{Q_j}$
5	Sørensen: $d_{sor} = \frac{\sum_{i=1}^{d} \left\|P_i - Q_i\right\|}{\sum_{i=1}^{d} \left(P_i + Q_i\right)}$	24	Neyman $\chi 2$: $d_{n(P,Q)} = \sum_{i=1}^{d} \frac{\left(P_i - Q_i\right)^2}{P_i}$
6	Gower: $d_{gow} = \frac{1}{d} \sum_{i=1}^{d} \left\|P_i - Q_i\right\|$	25	Squared $\chi 2$: $d_{Schi}(P,Q) = \sum_{j=1}^{n} \left(\frac{P_j - Q_j^{\,2}}{P_j + Q_j} \right)$ $d_{Schi}(P,Q) = \sum_{j=1}^{n} \left(\frac{P_j - Q_j^{\,2}}{P_j + Q_j} \right)$
7	Soergel: $d_{sg} = \frac{\sum_{i=1}^{d} \left\|P_i - Q_i\right\|}{\sum_{i=1}^{d} \max\left(P_i, Q_i\right)}$	26	Probabilistic Symmetric $\chi 2$: $d_{pchi} = 2\sum_{i=1}^{d} \frac{(P_i - Q_i)^2}{P_i + Q_i}$
8	Kulczynski: $d_{kul} = \frac{\sum_{i=1}^{d} \left\|P_i - Q_i\right\|}{\sum_{i=1}^{d} \min\left(P_i, Q_i\right)}$	27	Divergence: $d_{clk} = \sqrt{\sum_{i=1}^{d} \left(\frac{\left\|P_i - Q_i\right\|}{P_i + Q_i} \right)^2}$
9	Canberra: $d_{can} = \sum_{i=1}^{d} \frac{\left\|P_i - Q_i\right\|}{P_i + Q_i}$	28	Clark: $d_{clk} = \sqrt{\sum_{i=1}^{d} \left(\frac{\left\|P_i - Q_i\right\|}{P_i + Q_i} \right)^2}$

continued on the following page

Table 3. Continued

S.No	Distance/Formula	S.No	Distance/Formula
10	Lorentzian: $d_{Lor} = \sum_{i=1}^{d} \ln\left(1+\left\|P_i - Q_i\right\|\right)$	29	Addictive Symmetric $\chi 2$: $d_{Adchi} \sum_{j=1}^{b} \frac{\left(P_i - Q_i\right)^2 \left(P_i + Q_i\right)}{P_i Q_i}$
11	Intersection: $d_{Is} = \frac{1}{2} \sum_{i=1}^{d} \left\|P_i - Q_i\right\|$	30	Kullback-Leibler: $d_{KL} = \sum_{i=1}^{d} P_i ln \frac{P_i}{Q_i}$
12	Wave Hedges: $d_{WH} = \sum_{i=1}^{d} \frac{\left\|P_i - Q_i\right\|}{max\left(P_i, Q_i\right)}$	31	Jeffreys: $d_J = \sum_{i=1}^{d} \left(P_i - Q_i\right) \ln \frac{P_i}{Q_i}$
13	Czekanowski: $d_{Cze} = \frac{\sum_{i=1}^{d} \left\|P_i - Q_i\right\|}{\sum_{i=1}^{d} \left\|P_i + Q_i\right\|}$	32	K-divergence: $d_{Kdiv} = \sum_{i=1}^{d} P_i \ln \frac{2P_i}{P_i + Q_i}$ $d_{Kdiv} = \sum_{i=1}^{d} P_i \ln \frac{2P_i}{P_i + Q_i}$
14	Motyka: $d_M = \frac{\sum_{i=1}^{d} \max\left(P_i, Q_i\right)}{\sum_{i=1}^{d} \left(P_i + Q_i\right)}$	33	Tops $\varnothing e$: $d_T = \sum_{i=1}^{d} \left(P_i ln\left(\frac{2P_i}{P_i + Q_i}\right) + Q_i ln\left(\frac{2Q_i}{P_i + Q_i}\right)\right)$.
15	Tanimoto: $d_T = \frac{\left(max(P_i, Q_i) - min\left(P_i, Q_i\right)\right)}{\sum_{i=1}^{d} max\left(P_i, Q_i\right)}$	34	Jensen –Shannon: $d_{JS} = \frac{1}{2}\left[\sum_{i=1}^{d} P_i ln\left(\frac{2P_i}{P_i + Q_i}\right) + \sum_{i=1}^{d} Q_i ln\left(\frac{2Q_i}{P_i + Q_i}\right)\right]$
16	Jaccard: $d_J = \frac{\sum_{i=1}^{d} \left(P_i - Q_i\right)^2}{\sum_{i=1}^{d} P_i^2 + \sum_{i=1}^{d} Q_i^2 + \sum_{i=1}^{d} P_i Q_i}$	35	Jensen difference: $d_{JD} = \sum_{i=1}^{d} \left[\frac{P_i ln P_i + Q_i ln Q_i}{2} - \left[\frac{P_i + Q_i}{2}\right] ln\left[\frac{P_i + Q_i}{2}\right]\right]$
17	Dice: $d_D = \frac{\sum_{i=1}^{d} \left(P_i - Q_i\right)^2}{\sum_{i=1}^{d} P_i^2 + \sum_{i=1}^{d} Q_i^2}$	36	Taneja: $d_{Tj} = \sum_{i=1}^{d} \left(\frac{P_i + Q_i}{2}\right) ln\left(\frac{P_i + Q_i}{2\sqrt{P_i Q_i}}\right)$ $d_{Tj} = \sum_{i=1}^{d} \left(\frac{P_i + Q_i}{2}\right) ln\left(\frac{P_i + Q_i}{2\sqrt{P_i Q_i}}\right)$

continued on the following page

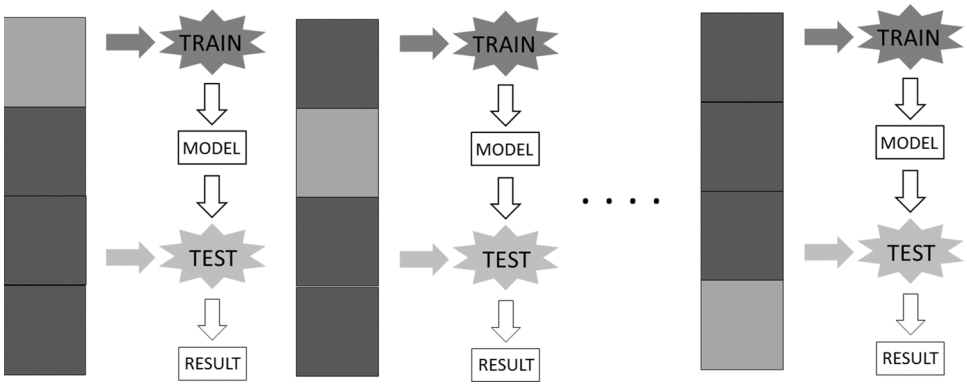
used to train and whereas other part is used during the testing or validation phase. The train and test is carried out through successive rounds of computations. It focuses mainly on the idea that testing is used at least once. The available data population is divided into *k*-folds of which one part is used for testing and rest of the parts is used for training (refer Figure 12). The process is repeated *k*-times.

One fold and two fold cross validation are the two cross validation techniques mainly used. Two fold divides the data set into two equal parts and one fold builds a decision tree. Later the roles are swapped. The final accuracy is calculated as the average of the two model accuracies.

Table 3. Continued

S.No	Distance/Formula	S.No	Distance/Formula
18	Bhattacharyya: $d_{b}=-\ln\sum_{i=1}^{d}\sqrt{P_i-Q_i}$	37	Kumar Johnson: $d_{KJ}=\sum_{i=1}^{d}\left(\frac{\left(\left(P_i^2-Q_i^2\right)^2\right)}{2\left(P_iQ_i\right)^{\frac{3}{2}}}\right)$
19	Hellinger: $d_{h}=\sqrt{2\sum_{i=1}^{d}\left(\sqrt{P_i-Q_i}\right)^2}$		$d_{KJ}=\sum_{i=1}^{d}\left(\frac{\left(\left(P_i^2-Q_i^2\right)^2\right)}{2\left(P_iQ_i\right)^{\frac{3}{2}}}\right)$
			$d_{KJ}=\sum_{i=1}^{d}\left(\frac{\left(\left(P_i^2-Q_i^2\right)^2\right)}{2\left(P_iQ_i\right)^{\frac{3}{2}}}\right)$
			$d_{KJ}=\sum_{i=1}^{d}\left(\frac{\left(\left(P_i^2-Q_i^2\right)^2\right)}{2\left(P_iQ_i\right)^{\frac{3}{2}}}\right)$

Figure 12. Procedure of four-fold cross-validation



Experiment with Cross Validation

The entire dataset is used for this experiment; the distance FVT is fed to WEKA. The experiment is carried with different percentage split (40, 60 and 70) with 10 folds. The possible outcomes of the experiment are *true positive (TP), true negative (TN), false positive (FP) and false negative (FP). True positive* denote the malware file is correctly classified as malware. *True negative* is benign file is correctly classified as benign.

The *false positive* indicate the total number of benign files classified as malware and *false negative* is the total number of malware files misclassified as benign (refer to Table 4). Perfect malware detector should have high *True Positive Rate (TPR)* with minimum *False Positive Rate (FPR).*

A Receive Operating Characteristic (ROC) curve is used to evaluate the results of the experiments. An ROC is a graphical plot which illustrates the performance of a binary classifier system A

Table 4. Confusion matrix

Input\Output	Malware	Benign
Malware	TP	FN
Benign	FP	TN

ROC space is defined by FPR and TPR plotted on *x* and *y* axes respectively. The best possible prediction method would yield a point, and this point is called a perfect classification. Experiments is performed with different classification algorithm such as Naïve Bayes, J-48, AdaBoost M1 (with J-48 as base classifier), and Random Forest all algorithms implemented in WEKA. In all cases classification accuracy obtained with Random forest was found to be better. Primary reason is that Random forest is an ensemble based classifier and final decision is estimated by collecting voted decision from multiple classifiers. Due to lack of space we report the classification accuracy obtained with Random Forest and AdaBoost classifiers. ROC curve obtained is shown in Figure 13 and Figure 14.

Inference

The Figure 13 and Figure 14 depict that 100% accuracy is obtained in the cross validation. Standardizing the result using cross validation proves that the model developed precisely identified malware from files from large collection of benign samples. Experiment also suggest that *bi-gram* features are effective in identifying malware and benign executables. The study also indicate that *non-signature* based methods provide a better detection rate where signature based detector fail to recognize new instances.

Train and Test

In this experiment, the testing and training set is randomly split using customized random file splitter module generated by us. The training set is identified, and supervised learning is performed where each instance is labelled in FVT. Feature vector table for test data is prepared where the class labels are not specified for any instances.

Initially, training model is developed using WEKA. Unseen samples are evaluated using performance metrics. We observe that an overall accuracy of 100% is obtained with classifier (refer Figure 15 and Figure 16) for different proportion of malware and benign programs.

Overall Inference

A Morphing or mutation engine is the heart of any metamorphic malware that incorporates structural change in new malware variant. The purpose of this metamorphic engine is to generate new patterns not found in signature repository to escape from the signature-based scanner. Although, the obfuscation techniques attempts to convert the binary sequence of the code. However, we feel that there is a limit to which an instruction could be modified, otherwise the maliciousness of the program will be lost, and sometime the resultant program would undesirable. The proposed system exhibited that the malware samples can be detected from an imbalanced data set consisting of large number of benign samples.

The NGVCK tool claims to produce highly obfuscated codes. Also, the metamorphic worm generated in earlier studies depict that obfuscation was in the form of dead code. These observations prove that the optimal feature vector can accurately identify the files. Therefore, the statistical method based on computed similarity/distance metrics is found to be effective in identifying obfuscated files.

CONCLUSION

A model was developed to identify the malware files from benign files. The opcodes from the train/test files were extracted and the *bi-gram* opcode were generated. The robust features are developed

Figure 13. Cross validation using random forest

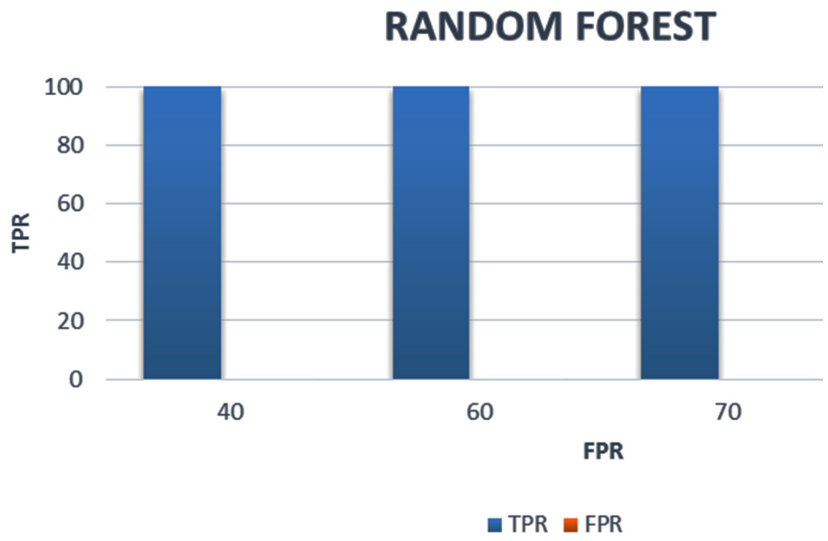
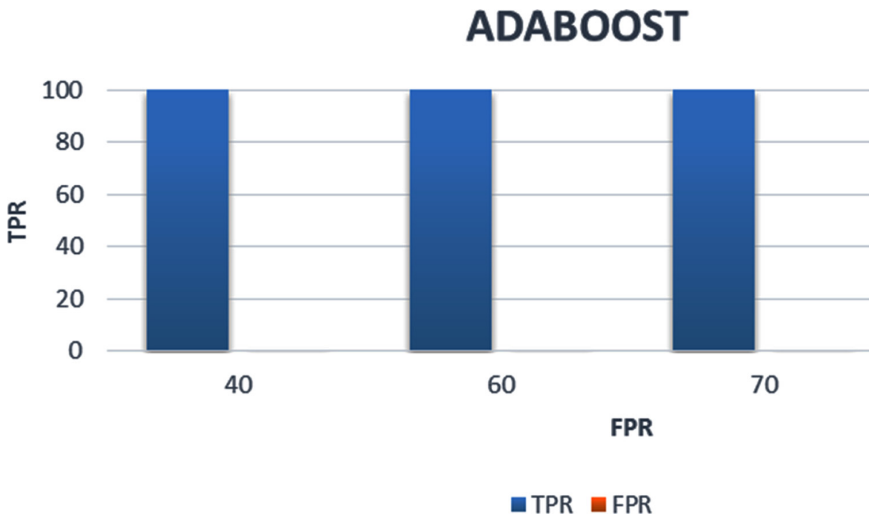


Figure 14. Cross validation using AdaBoost



from the *bi-gram* using the similarity indices. It was observed that the proposed method could classify benign samples with better accuracy. The observations of the experiments carried out led to the following conclusions:

1. The results shows 100% accuracy in both experiments. This shows that the malware used in the study contained only minimal obfuscation.
2. The metamorphic engine develops the variants which concentrate on changing the opcodes with equivalent ones, whereas the prominent features remain more or less the same.
3. The changes brought about in the program may simply be produced by inserting junk benign codes which doesn't affect the analysis.

Figure 15. Test and train data using random forest

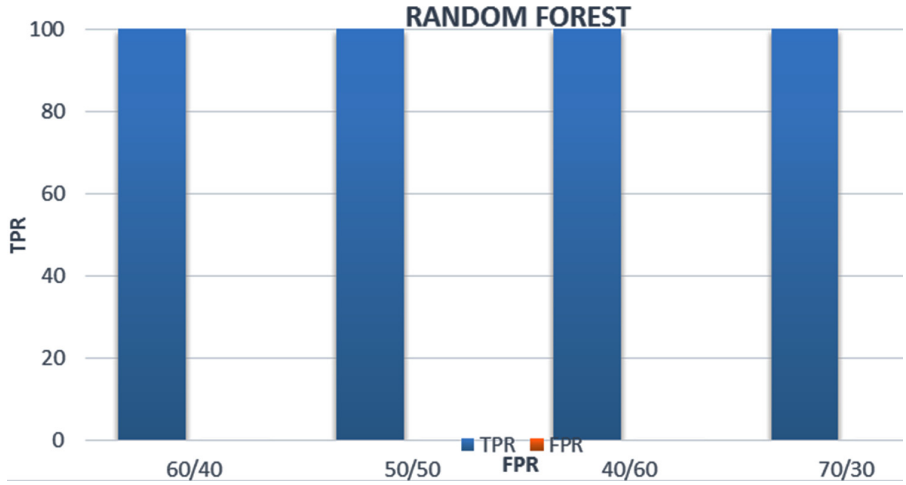
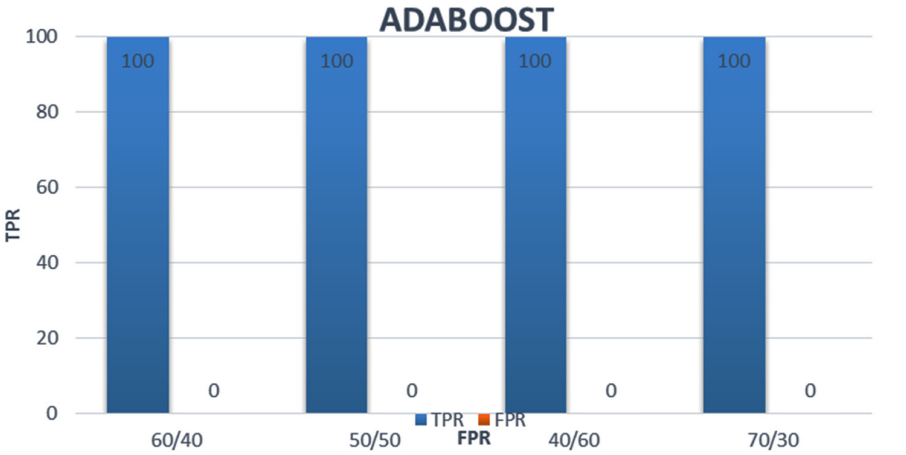


Figure 16. Test and train data using Adaboost



4. The data set used by the earlier researchers to claim that the metamorphic engines produce high degree of obfuscation, where as our proposed method justifies that the obfuscation is weak. Also, dataset in prior study use large number of malware files in comparison to benign specimens. This setup of the dataset might bias identifying malware samples with high accuracy. Moreover, database used in our experiments is closer to real world since the number of malware is lesser than that of benign.
5. The experiments show that the feature set was reduced drastically from 10925 to 325 in benign class for 80/20. Whereas, in malware files the feature length was reduced from 807 to101. The reduced feature length delivered 100% accuracy, which shows that the proposed system is very efficient with respect to execution time during the detection phase.

FUTURE SCOPE

The proposed system has accurately identified and raised appropriate alarms. The efficiency when used in real malware samples needs to be tested.

The proposed system though have produced accurate results needs further advancement in the certain areas.

1. The efficiency of the system could be still evaluated by further reducing the feature length. Another advantage of the reduced feature length is that the overhead is decreased and the detection becomes faster.
2. In the proposed system, 37 similarity indices are used in order to find the similarity between two files. Investigation needs to done so as to further decrease the similarity measurement indices and still achieve higher accuracy.
3. The threshold can be determined (needs to be fixed) for identifying the malware and benign files. To find the threshold, a base file needs to be figured out that is highly similar to the rest of the samples in a family. Further distance of test file could be computed with the base file order to classify the files. Further experiments can be carried out in real and bigger malware dataset.

OPEN RESEARCH PROBLEMS

Like desktop malware detection, mobile malware especially Android malware research is attracting large number of researchers. The detection of mobile malware is in infancy stage. The domain is attracting huge number of researchers due to the increasing popularity and computational power of smart phones. Following are some research areas which can be useful to tackle the detection of injected malicious code.

1. Need to develop malicious program that are structurally similar to benign code but have malicious intents. This would help researchers understand the how the two classes of program differ at low level operations.
2. Identification and classification of prominent applications downloaded by users and inspect points of vulnerabilities.
3. Understand various sandboxes available for mobile/desktop malware detection and inspect different points in systems which are exposed to VM installation. This is mandatory as the future malware would exploit these vulnerable infection vectors to launch virtual machine attack.
4. Determine behavioral features for malware and benign applications. Study the default permissions and modifications during series of time interval. This could help us develop strong benign and malware models.
5. Investigate robust feature selection and reduction techniques to extract prominent/discriminant features for classification of samples (suspicious or benign).
6. Identification of malicious APIs and prepare white, grey and blacklist for the same.

ACKNOWLEDGMENT

We would like to especially thank Prof. Mark Stamp, Department of Computer Science, San Jose State University, USA, for sharing malware database for carrying out this study.

REFERENCES

Andrew, W., Michael, V., Matthew, H., Christopher, T., & Arun, L. (2007). *A: Exploiting Similarity between Variants to Defeat Malware: Vilo Method for Comparing and Searching Binary Programs.* Retrieved from https://blackhat.com/presentations/bh-dc-07/Walenstein/Paper/bh-dc-07-walenstein-W%P.pdf

Annie, H. T., & Mark, S. (2013). Chi-squared distance and metamorphic virus detection. Journal in Computer Virology, 9(1), 1-14

Asaf, S., Robert, M., Yuval, E., & Chanan, G. (2009). Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey. *Information Security Technical Report*, *14*(1), 16–29. doi:10.1016/j.istr.2009.03.003

Attaluri, S., McGhee, S., & Stamp, M. (2009). Profile hidden Markov models and metamorphic virus detection. *Journal in Computer Virology*, *5*, 151–169. doi:10.1007/s11416-008-0105-1

Aycock, J. (2006). Computer Viruses and Malware. Academic Press.

Baker, K. (2013). *Singular Value Decomposition Tutorial website*. Retrieved from http://www.ling.ohio-state.edu/~kbaker/pubs/Singular_Value_Decomposition_Tutorial.pdf

Baysa, D., Low, R. M., & Stamp, M. (2013). Structural entropy and metamorphic malware. Journal of Computer Virology and Hacking Techniques, 9(4), 1-14, doi: doi:10.1007/s11416-013-0185-4

Cha, S.-H. (2007). Comprehensive Survey on Distance/Similarity Measures between Probability Density Functions. *International Journal of Mathematical Models and Methods in Applied Sciences*, *1*, 300–307.

Chouchane, M. R., & Lakhotia, A. (2006). Using engine signature to detect metamorphic malware: In *Proceedings of the 4th ACM Workshop on Recurring malcode*, (pp. 73-78). ACM.

Dima, S., Robert, M., Zvi, B., Yuval, S., & Yuval, E. (2009). Using artificial neural networks to detect unknown computer worms. *Neural Computing & Applications*, *18*(7), 663–674. doi:10.1007/s00521-009-0238-2

Eitan, M., Asaf, S., Lior, R., & Yuval, E. (2009). Improving malware detection by applying multi-inducer ensemble. *Journal In Computational Statistics & Data Analysis*, *53*(4), 1483–1494. doi:10.1016/j.csda.2008.10.015

Henchiri, O., & Japkowicz, N. (2006). A Feature Selection and Evaluation Scheme for Computer Virus Detection. In Proceedings of ICDM (pp. 891-895). IEEE Computer Society.

Virus Collection Website. (n.d.). Retrieved from http://vx.netlux.org/lib

Igor, S., Felix, B., Javier, N., Yoseba, P., Borja, S., Carlos, L., & Pablo, B. (2010). Idea: Opcode-sequence-based malware detection: In Proceedings of Engineering Secure Software and Systems (LNCS) (vol. 5965, pp. 35-43). Springer.

Jeremy, Z. K., & Maloof, M.A. (2004). Learning to Detect Malicious Executables in the Wild. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, (pp. 470-478). New York, NY: ACM

Karim, M. E., Walenstein, A., Lakhotia, A., & Parida, L. (2005). Malware phylogeny generation using permutations of code. *Journal in Computer Virology*, *1*, 13–23. doi:10.1007/s11416-005-0002-9

Kephart, J. O., & Arnold, B. (1994). A Feature Selection and Evaluation of Computer Virus Signatures. In *Proceedings of the 4th Virus Bulletin International Conference*, (pp. 178-184). Academic Press.

Kolter, J. Z., & Maloof, M. A. (2006). Learning to Detect and Classify Malicious Executables in the Wild. *Journal of Machine Learning Research*, *6*, 2721–2744.

Lin, D., & Stamp, M. (2011). Hunting for undetectable metamorphic viruses. *Journal in Computer Virology*, *7*, 201–214. doi:10.1007/s11416-010-0148-y

Menahem, E., Rokach, L., & Elovici, Y. (2009a). Troika - An improved stacking schema for classification tasks. *Inf. Sci.*, *179*, 4097–4122. doi:10.1016/j.ins.2009.08.025

Menahem, E., Shabtai, A., Rokach, L., & Elovici, Y. (2009b). Improving malware detection by applying multi-inducer ensemble. *Computational Statistics & Data Analysis*, *53*, 1483–1494. doi:10.1016/j.csda.2008.10.015

Merkel, R., Hoppe, T., Kraetzer, C., & Dittmann, J. (2010). Statistical Detection of Malicious PE-Executables for Fast Offline Analysis. In B. De Decker & I. Schaumller-Bichl (Eds.), Communications and Multimedia Security, (Vol. 6109, pp. 93-105). Springer.

Moskovitch, R., Nissim, N., & Elovici, Y. (2009). Acquisition of Malicious Code Using Active Learning. In Proceedings of Privacy, Security, and Trust in KDD. Springer-Verlag.

Moskovitch, R., Stopel, D., Feher, C., Nissim, N., Japkowicz, N., & Elovici, Y. (2009). Unknown malcode detection and the imbalance problem. *Journal in Computer Virology*, *5*, 295–308. doi:10.1007/s11416-009-0122-8

Nissim, N., Moskovitch, R., Rokach, L., & Elovici, Y. (2012). Detecting unknown computer worm activity via support vector machines and active learning. *Pattern Analysis & Applications*, *15*, 459–475. doi:10.1007/s10044-012-0296-4

Open Source Machine Learning Software Weka website. (n.d.). Retrieved from http://www.cs.waikato.ac.nz/ml/weka/

Runwal, N., Low, R. M., & Stamp, M. (2012). Opcode graph similarity and metamorphic detection. *Journal in Computer Virology*, *8*, 37–52. doi:10.1007/s11416-012-0160-5

Santos, I., Penya, Y. K., Devesa, J., & Bringas, P. G. (2009). N-grams-based File Signatures for Malware Detection. In J. Cordeiro & J. Filipe (Eds.), ICEIS (vol. 2, pp. 317-320). Academic Press.

Schultz, M. G., Eskin, E., Zadok, E., & Stolfo, S. J. (2001). Data Mining Methods for Detection of New Malicious Executables. In *Proceedings of IEEE Symposium on Security and Privacy* (pp. 38-49). IEEE Computer Society.

Seon, Y., & Ulrich, U. N. (2006). Towards Establishing a Unknown Virus Detection Technique using SOM. Journal in Computer Virology, 2(3), 163-186.

Sridhara, S. M., & Stamp, M. (2013). Metamorphic worm that carries its own morphing engine. *Journal in Computer Virology*, *9*, 49–58.

Tabish, S. M., Shafiq, M. Z., & Farooq, M. (2009). Malware detection using statistical analysis of byte-level file content. In H. Chen, M. Dacier, M.-F. Moens, G. Paass & C. C. Yang (Eds.), *KDD Workshop on CyberSecurity and Intelligence Informatics* (pp. 23-31). ACM.

Vinod, P., Jain, Golecha, Gaur, & Laxmi. (2010a). MEDUSA: Metamorphic malware dynamic analysis using signature from API. In *Proceedings of the 3rd International Conference on Security of Information and Networks.* SIN.

Vinod, P., Laxmi, & Gaur. (2012a). REFORM: Relevant Features for Malware Analysis. In *Proceedings of 26th International Conference on Advanced Information Networking and Applications Workshops.* Academic Press.

Vinod, P., Laxmi, Gaur, Kumar, & Chundawat. (2009). Static CFG analyzer for metamorphic Malware code. In *Proceedings of the 2nd International Conference on Security of Information and Networks.* SIN.

Vinod, P., Laxmi, V., & Gaur, M. (2011a). Metamorphic Malware Analysis and Detection Methods. In Cyber Security, Cyber Crime and Cyber Forensics: Applications and Perspectives, (pp. 178-202). Hershey, PA: IGI Global. doi: doi:10.4018/978-1-60960-123-2.ch013

Vinod, P., Laxmi, V., & Gaur, M. S. (2011b). Scattered Feature Space for Malware Analysis. In Proceeding Advances in Computing and Communications. Springer.

Vinod, P., Laxmi, V., Gaur, M. S., & Chauhan, G. (2012b). MOMENTUM: MetamOrphic malware exploration techniques using MSA signatures. In Proceedings of International Conferenceon Innovations in Information Technology (IIT). IIT.

Vinod, P., Laxmi, V., Gaur, M. S., Naval, S., & Faruki, P. (2013). MCF: MultiComponent Features for Malware Analysis. In *Proceedings of the 27th International Conference*. doi: doi:10.1109/WAINA.2013.147

Wei, J. L., Wang, K., Stolfo, S.J., & Herzog, B. (2005). Fileprints: Identifying File types by n-gram analysis. In *Proceedings of the Sixth Annual IEEE SMC 4th Virus Bulletin Conference.* IEEE.

Wong, W., & Stamp, M. (2006). Hunting for metamorphic engines. *Journal in Computer Virology*, *2*, 211–229. doi:10.1007/s11416-006-0028-7

Yan, W., Zhang, Z., & Ansari, N. (2008). Revealing Packed Malware. *IEEE Security & Privacy*, *6*, 65–69. doi:10.1109/MSP.2008.126

KEY TERMS AND DEFINITIONS

Bi-Grams: Collection of two words or opcodes or term extracted in sliding window fashion.

Classifier: An algorithm that implements a classification task that assign a unique class to an instance.

Code Obfuscation: The process of code hiding important details of the program so as to conceal it form reverse-engineering.

Cross Validation: Model estimation technique based on rotation.

Evaluation Parameters: Various criteria for estimating the classification model.

Feature Reduction: The process which involves selection of prominent features (subset) from a large feature space.

Feature: Is representative of attribute.

Malware Constructors: Tool or kit that assist any user to develop synthetic malicious code without having the knowledge how the code is to be developed.

Malware: Is a generic term used to refer to all potential malicious software.

Metamorphic Engine: Is a heart of metamorphic malware which induces structural transformation in the code structure.

Metamorphic: The process by which a program undergoes transformation.

Opcodes: Is operation code, refer to a part of the complete instruction.

Similarity Measure: Is a function applied over any two objects in order to measure if the objects have high proximity.

Static Analysis: Structural analysis of malware samples without its execution.