An incremental malware detection model for meta-feature API and system call sequence

Pushkar Kishore
Dept. of C.S.E.
NIT Rourkela
Odisha, India
518CS1002@nitrkl.ac.in

Swadhin Kumar Barisal
Dept. of C.S.E.
NIT Rourkela
Odisha, India
swadhinbarisal@gmail.com

Durga Prasad Mohapatra
Dept. of C.S.E.
NIT Rourkela
Odisha, India
durga@nitrkl.ac.in

***Abstract*—In this technical world, the detection of malware variants is getting cumbersome day by day. Newer variants of malware make it even tougher to detect them. The enormous amount of diversified malware enforced us to stumble on new techniques like machine learning. In this work, we propose an incremental malware detection model for meta-feature API and system call sequence. We represent the host behaviour using a sequence of API calls and system calls. For the creation of sequential system calls, we use NITRSCT (NITR System call Tracer) and for sequential API calls, we generate a list of anomaly scores for each API call sequence using Numenta Hierarchical Temporal Memory (N-HTM). We have converted the API call sequence into six meta-features that narrates its influence. We do the feature selection using a correlation matrix with a heatmap to select the best meta-features. An incremental malware detection model is proposed to decide the label of the binary executable under study. We classify malware samples into their respective types and demonstrated via a case study that, our proposed model can reduce the effort required in STS-Tool(Socio-Technical Security Tool) approach and Abuse case.**

Theoretical analysis and real-life experiments show that our model is efficient and achieves 95.2% accuracy. The detection speed of our proposed model is 0.03s. We resolve the issue of limited precision and recall while detecting malware. User's requirement is also met by fixing the trade-off between accuracy and speed.

***Index Terms*—meta-feature, API call, system call, incremental malware detection, Abuse Case, STS-Tool**

I. Introduction

TODAY, we are facing one of the toughest security threats, malware. Whenever an unknown application is installed by a user on their systems, the malware detector uploads the application's executable on the cloud to verify whether an application is malicious or benign. After the executable is received, the detection system unpacks it using tools like PEiD[1], PolyUnpack [1], etc. Then, the detection system disassembles the binary to extract API or system calls and trains a machine-learning based model for classification.

Sequential series is a critical class of data, which can be applied in anomaly detection [2], trend analysis [3], periodic pattern detection [4], short-term prediction [5], etc. API call profile has API call sequence, e.g. <WriteFile; VirtualQueryEx; UnmapViewOfFile; Sleep; ...>. Anomaly score describes the sophisticated aggregation of the anomaly records. Numenta Hierarchical Temporal Memory (N-HTM) [6], an anomaly score generator, can be used to generate anomaly score for each API call in an API call sequence. We will treat the set of anomaly score of every instance in an API call sequence as a newer *API anomaly score sequence*, e.g. *API relative frequency call sequence* can be: <1, 1, 1, 1, 2, 2, ...>. For the case of the system call sequence, we use the dataset generated by NITRSCT [7]. Embedding various features in a single malware detector becomes non-functional when adversarial attack occurs. So, we design an incremental malware detector which accomplishes the task of malware detection if one of its layers fails. The accurate classification of malware families is still a tough problem and is also significant in malware analysis. Whenever software is used, security needs to be assured thoroughly among the users and software. During the software development life cycle (SDLC), the Abuse case and the STS-Tool approach can produce secured software. To specify security requirements for the software, Abuse case is used. Abuse case [8] is a model for specifying security requirements. The term Abuse case is an alteration of the use case. STS (Socio-Technical Security) [9] models security requirements considering actors as various stakeholders and their goals as main objectives. It tackles security-related issues during the early phase of the socio-technical system design.

Despite having modern malware detection systems, researchers are still facing many challenges. First, a single-layer malware detection system is prone to adversarial or evasion attacks and the detector will fail. Besides, accuracy is acutely limited during run-time [10]. Secondly, user's expectation is not met while fixing trade-off between accuracy and speed. Thirdly, a lot of effort is wasted in the wrong direction in STS-Tool approach and Abuse case. Lastly, it is very tough to provide labels to malicious samples according to their class. To address the above challenges, we propose a novel incremental malware detection model for meta-feature API and system call sequence. API calls can be extracted using tools like IDA[2], W32dasm[3], etc. This will help in quick preparation

[1]https://www.softpedia.com/get/Programming/Packers-Crypters-Protectors/PEiD-updated.shtml

[2]https://www.hex-rays.com/products/ida/

[3]https://www.softpedia.com/get/Programming/Debuggers-Decompilers-Dissasemblers/WDASM.shtml

of the collection of API call sequences. We use NITRSCT [7] generated datasets for the system calls. We use N-HTM for generating an anomaly score for each API call in an API call sequence. For an API call sequence, <WriteFile; VirtualQueryEx; UnmapViewOfFile; Sleep; ...>, <0.2, 0.3, 0.7, 0.1, ...> may be the anomaly score sequence. We predict the final label of the executable using an incremental model. At first, we apply malware detection on the system call dataset using one-class SVM. Then, we send only benign samples for testing to malware detector based on meta-feature API calls dataset using one-class SVM. The reason behind sending only benign samples is to ensure that none of the malicious executables gets executed due to wrong labelling by the first detector. So, we cross-check it with the second detector.
We select six meta-features, which represent the characteristics of the API anomaly score sequence. We assume that an API anomaly score sequence **X=** $\{x_1, x_2, x_3, ..., x_Z\}$, where Z is the length of the sequence. **X** is divided into m sub-sequences. For each sub-sequence, we calculate the values of meta-features: Kurtosis, Coefficient of Variation, Oscillation, Regularity, Square wave and Variation of trend. By combining various sub-sequences, we get the final dataset having the six meta-features. A correlation matrix with a heatmap is used to select the best meta-features. The incremental model helps detect malware whenever the dataset is ready. We classify the malicious binary executable into their respective classes. A case study is included to demonstrate that effort required in Abuse case and STS-Tool can be reduced by our proposed model.

The main objectives of this paper are:

1) To represent API call sequence, we propose to use N-HTM. In order to improve the convergence speed of the malware detector, we use meta-features derived from the API anomaly score sequence. We use the incremental model trained using one-class SVM to detect the malware.
2) To reduce the number of meta-features using correlation matrix displayed in heatmap. This reduction accelerates the convergence speed of our model.
3) To implement the incremental model and assess it using an extensive scale of real-world data set. We use anomaly score to assign malicious binary executable its proper class.
4) To demonstrate that effort required in Abuse case and the STS-Tool approach can be reduced using our proposed model.

Paper Organizations The remaining part of this paper is organized as follows: Section II briefly describes the related work. Section III introduces the methodology of our proposed model. Section IV presents the experimental results. Section V discusses the comparison with related work. Section VI shows the threats to the validity and Section VII presents the conclusions and future work.

II. Related Work

A. API Call Based Method

Many researchers used API calls to represent binary executable. Patnaik, Barbhuiya and Nandi [11] checked the target process's API call similarity with the API call signature of the malware. Huang, Zhang and Tan [12] detected stealthy behaviour by analyzing the user interface components of top-level function. Fan et al. [13] proposed constructing sub-graphs of API calls to represent the similar behaviour of malware of the same family.

B. System Call Based Method and Malware Detection Model

Canzanese, Mancoridis and Kam [14] used system call n-gram method for representing binary executable and support vector machine (SVM) for malware detection. The performance is quite good as system calls precisely represent the binary executable's behaviour. This method fails if any malware hides in a computer and conceals its malicious behaviour. Zhang, Qin, Zhang, Yin and Zou [15] proposed a lightweight framework for malware detection based on the graph and information theory. But, whenever a malware attack occurs, its detection will be complex as there will be numerous interactions between files, processes, etc, leading to the nexus of graph. Raff, Barker, Sylvester, Brandon, Catanzaro and Nicholas [16] used convolutional neural networks (CNN) and bytecode n-grams for malware detection. Bytecodes are noisy compared to opcodes, thus the accuracy is limited. Kang, Yerima, McLaughlin and Sezer [17] used the Naive Bayes (NB) method for detecting 2-opcode vectors represented malware. This method's accuracy is very small as NB assumes that the features are independent. Puerta, Sanz, Santos and Bringas [18] used opcode frequencies to represent binary executable and detected malware using SVM. Lack of simplicity of features jeopardizes the accuracy.

C. Sequential Series

Numerous approaches are available to find anomalies in univariate/multivariate sequences. We group these methods into four categories: (1) Statistics-based methods [19] (2) Intelligent- computing methods [20] (3) Bayesian networks [21] and (4) Model-based approaches [22]. Statistical based methods come from techniques that detect abnormal changes. A variety of intelligent computing methods are available for detecting anomalies, such as deep learning [23], SVM [5], fuzzy theory and rough sets theory.

D. Anomaly score generation

For generating anomaly scores of a sequential data, Ahmad, Lavin, Purdy and Agha [6] proposed a technique named N-HTM. It is suitable for real-time applications and robustly detects anomalies for any data stream. They have also shown that their system is efficient, produces accurate results even in the presence of noisy data, adaptable to statistical change in the data, detects subtle temporal anomalies and minimizes false positives.

E. Security Approaches

For the STS (Socio-Technical Security) approach, STS-ml [9] is used which includes actors and is a goal-oriented modelling language. This approach relates security requirements to interaction. Paja, Dalpiaz and Giogini [9] proposed a technique to handle security requirement conflicts in socio-technical systems. The STS modelling language allows stakeholders to impose security concerns over the interactions. For example, if buyers send their personal data to a seller, the seller must not disclose the data to third parties, only the buyer should be able to access them. There is a commitment between the actors which ensures that they will consider security requirements while delivering service. One example of the security requirement is that the seller commits that they will not reveal buyer's personal data to anyone.

III. Proposed Methodology

In this section, we propose an incremental malware detection model for meta-feature API and system call sequence. We present the architecture of our malware detection model in Figure 1. It comprises of seven steps: Creating system call dataset, classification using one-class SVM, unpacking and disassembly process, generating anomaly score sequence from API call sequence using Numenta Hierarchical Temporal Memory, defining meta-features for the anomaly score sequence, creating meta-feature API call dataset and classification using one-class SVM for benign binary executables. By performing the above seven steps, we can detect malware. We present the algorithm in Algorithm 1. The time complexity is $O(n^2)$ and space complexity is O(mn), where m is the number of features and n is the number of instances. The description of the above seven steps are discussed below.

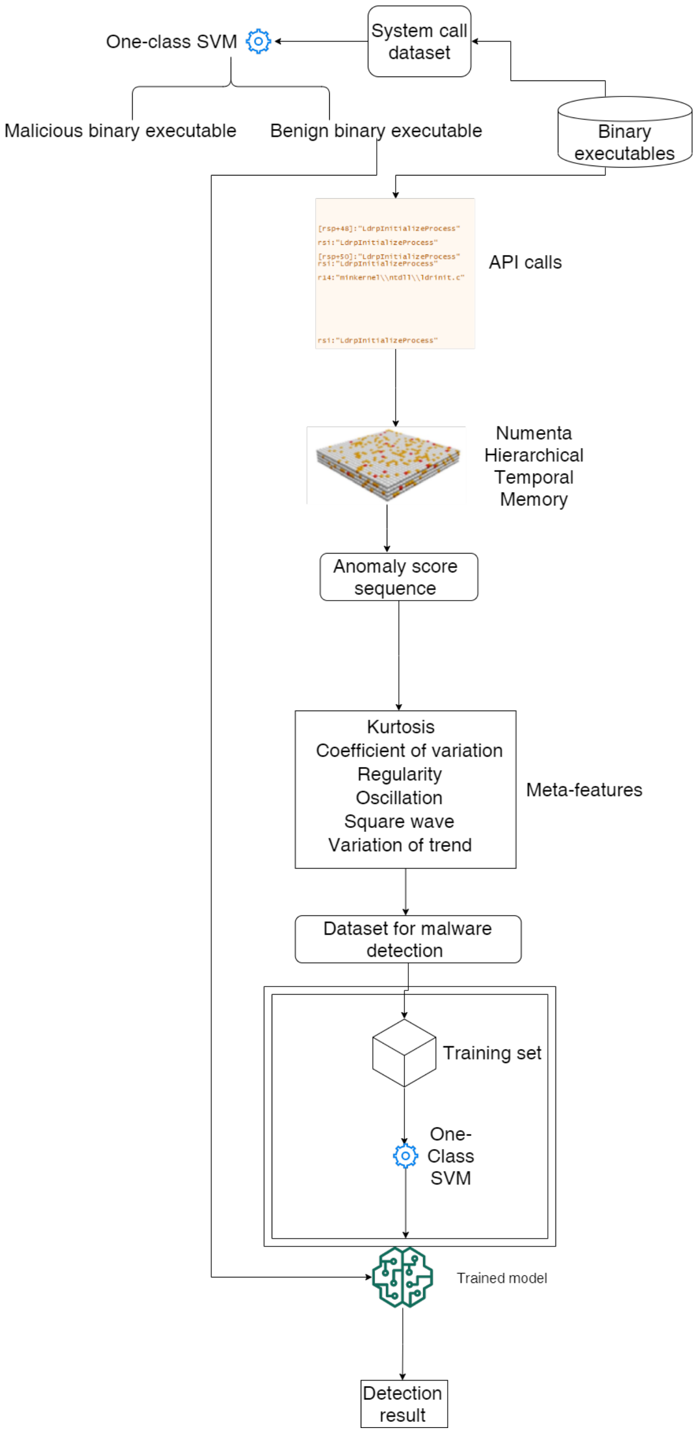
A. Creating system call dataset

We use NITRSCT [7] generated system call dataset[4]. This dataset contains system calls gathered from Windows OS based benign and malicious binary executables. The features are represented in the form of a vector having three consecutive ordered system calls.

B. Classification using one-class SVM

We train the system call dataset using the one-class SVM model. Upon testing, we send the benign results to the next malware detector, i.e, meta-feature API detector. Benign binary executable is sent for re-verification since we do not want any malicious binary executable to damage the host. If any evasion or adversarial attack occurs, attackers make sure that the label of the real malware is misrepresented as benign. In this case, our incremental model comes to rescue, we will recheck that false labelled malware using another detector which blocks execution of the malicious executables. Meta-feature API detector is based on N-HTM technique and is least prone to attacks compared to deep learning techniques. After analysing the robustness of the models, we decide to keep machine-learning based detector in front and N-HTM at last so that our incremental model will still work if an attack occurs on the machine-learning based detector.

[4]https://github.com/pushkarkishore/NITRSCT



```
[rsp+48]:"LdrpInitializeProcess"
rsi:"LdrpInitializeProcess"
[rsp+50]:"LdrpInitializeProcess"
rsi:"LdrpInitializeProcess"
r14:"minkernel\\ntdll\\ldrinit.c"
rsi:"LdrpInitializeProcess"
```

Fig. 1. Proposed architecture of our approach

C. Unpacking and disassembly process

Unpacking and disassembly processes are used to unpack and disassemble executables for getting their API calls. Attackers may have packed some binary executables using some packing tools which are harder to disassemble. We unpack them first, if they are packed with ASPack[5], UPX[6], etc. Then, we disassemble the unpacked executable to get the API calls using Ollydbg[7]. We use limited disassembly tools to avoid distortion of the results. After completion of disassembling, we build an API call sequence having a list of API calls.

Algorithm 1: Meta-feature API and system call based malware detection

```
Input: A set of API calls, APIs = {API_1, API_2, . . ., API_j}, where API_j represents the jth call;
       System call dataset.
Output: A final label informing whether the binary executable is malicious or benign.
1  Function MalwareDetector(APIs):
2      for training instances in the system call dataset do
3          Apply one-class SVM technique to train the malware detection system;
4      for each test instance in the system call dataset do
5          Apply one-class SVM classifier to predict the final label, i.e. anomalous or benign;
6          Send the benign samples for testing to meta-feature API detector;
7      for API_j in APIs do
8          Generate a vector of API relative frequencies V(api_i) according to a set of API calls;
9          Apply N-HTM model to create API anomaly score sequence of each API call sequence;
10         Generate dataset having best meta-features;
11     for training instances in the meta-feature API call dataset do
12         Apply one-class SVM technique to train the malware detection system;
13     for each test instance obtained from system call detector do
14         Apply one-class SVM classifier to predict the final label, i.e. malicious or benign;
15     return;
16 end
```

D. Generating API anomaly score sequence using Numenta Hierarchical Temporal Memory (N-HTM)

The *API anomaly score sequence* is generated from the API call sequence using relative frequency, e.g. for an API call sequence, <WriteFile; VirtualQueryEx; UnmapViewOfFile; Sleep; WriteFile; Sleep ...>, the *API relative frequency call sequence* will be: <1, 1, 1, 1, 2, 2, ...>. N-HTM [6] calculates an anomaly score for an API call upon receiving new patterns from the API call sequence. If the received pattern is predicted, then anomaly score is zero, while for the completely non-predictable pattern, it is one. Partial prediction of pattern has an anomaly score between zero and one. The similarity between actual and received patterns is calculated using sparse distributed representation. The anomaly score is dependent on the difference of overlap between actual and predicted bits.

[5]http://www.aspack.com
[6]https://upx.github.io.
[7]http://www.ollydbg.de/

TABLE I
ANOMALY SCORES OF SAMPLE API CALL SEQUENCE

Timestamp	Value	Anomaly Score
1-3-20 0:00	1	0.03010299967
1-3-20 0:00	1	0.03010299967
1-3-20 0:01	2	0.03010299967
1-3-20 0:01	2	0.03010299967
1-3-20 0:02	3	1
1-3-20 0:02	4	1
1-3-20 0:03	5	1

The anomaly score of a sample API call sequence is depicted in Table I. In Table I, 'Value' represents the API call sequence, where 1, 2, 3, ... represents the relative frequency of the API call. Anomaly score is associated with all the entries of the API call sequence.

E. Defining meta-features for the anomaly score sequence

We define six meta-features which is statistical representation of the API anomaly score sequence. The approach used in the different meta-features is discussed below:

1) **Kurtosis**: It measures whether a sequence is heavily tailed or lightly tailed related to the normal distribution [24]. For ECG data [25], kurtosis is effective in detecting the abrupt peaks from a sequence. It reflects variability of a sequence. It actually measures the number of outliers present in the distribution. Sequence with high kurtosis has generally heavy tails; but, low kurtosis shows light tails. We use Equation 1 for the subsequence created from API anomaly score sequence for calculating kurtosis.

$$K_z = \frac{1}{n}\sum_{i=1}^{n} D_i^4 - 3 \tag{1}$$

where, n is the length of the z-th sub-sequence derived from an API anomaly sequence; D_i values are the standardized data values defined using standard deviation with n as the denominator.

2) **Coefficient of variation**: It calculates the local variability relative to the complete sequence [26]. Local variability of a sub-sequence significantly rises if the abrupt peak occurs within the sub-sequence's interval. This meta-feature indicates the sub-sequence's sharp curve changes. It is used mainly for checking the consistency of sequence. We use Equation 2 for the subsequence created from the API anomaly score sequence for calculating coefficient of variation.

$$C_z = \frac{\sigma_z}{\mu} \tag{2}$$

where, σ_z denotes the standard deviation of the z-th sub-sequence; and μ is the mean value of all sub-sequences of API call sequence.

3) **Oscillation**: It is a periodic fluctuation between two consecutive anomaly scores in a sequence. In this work, we calculated the oscillation of a sub-sequence.
4) **Regularity**: Sample entropy [27] is used to calculate the regularity of series. It is also widely used for diagnosing the presence or absence of a disease [28]. Regularity will be higher, if there are less number of abrupt peaks in the sequence.
5) **Square wave**: These waves are generated by binary logic devices and encountered in digital switching circuits. A sequence can start and maintain the signal with high values in the first half, and sharply reduces for the second half. We have assumed that the curve of a variable is consistent if the square wave is represented and consistent with expectation. In the case of API anomaly score sequence, z-th sub-sequence is $X_z = \{ x_{z,1}, ..., x_{z,i}, x_{z,i+1}, ..., x_{z,N} \}$ and $i = \lfloor 0.5N \rfloor$. The binarized sub-sequence of X_z, represented as TX_z is calculated using Equation 3.

$$TX_z = X_z > 0.5 * max(X_z) \tag{3}$$

Confirmation of the z-th sub-sequence being square wave is done using Equation 4 .

$$S_z = 0.5 - rs * TX_z/LEN(TX_z) \tag{4}$$

where, LEN denotes the length of TX_z, vector rs = $\{1, 1, ..., 1, 0, 0, ..., 0\}$ filters the signal with high values in TX_z. In vector rs, the value of “1” is i. According to Equation 4, the sub-sequence corresponds to a lower value, if it represents a square wave.
6) **Variation of trend**: Trend analysis provides a way to differentiate between two series. We smoothen the original API anomaly score sequence and calculate the variation on it. The variation of trend of z-th sub-sequence is defined using Equation 5 .

$$T_z = std(smooth(X_z)) \tag{5}$$

where, smooth is used for smoothing the original API sequence and std is used for finding its standard deviation. For a sequence having random trend, T_z will be small, if abrupt peaks are absent.

We have used the correlation matrix with heatmap to select the best meta-features. A heatmap is a graphical representation of data where values are represented as colours. So, viewer can refer to the colour for getting the value of data.

F. Creating meta-feature API call dataset

Two datasets are considered in performance analysis of our malware detection model: a malware dataset and a benign dataset. We have collected benign binary executables from 10 hosts in offices, computer laboratories, and isolated testbed to test within real-life environment. The malware which we use in our experiments are collected from VirusTotal[8]. In order to make sure that, the instances are unpacked, we detect the packers. Using the unpacking tools, we have unpacked the executables. Our final dataset contains 5114 malware binary executables and 4800 benign binary executables as shown in Table II. We split the malware and benign dataset into training dataset and a test dataset. The volume of the malware training dataset and benign dataset is fixed to avoid training biases. We randomly choose 2000 malware and benign samples for training purpose, while remaining samples are used for testing purpose. The final label is predicted using incremental model.

TABLE II
MALWARE DATASET

Sl. No.	**Malware family**	**Number of samples**
1	Backdoor	1352
2	Worm	559
3	Trojan	2394
4	Virus	809
Total number of samples		5114

G. Classification using one-class SVM for benign binary executables

We use a one-class support vector machine, which classifies by separating hyperplane. Linear SVM [29] uses a hyperplane $w^T\mathbf{x}$, which separates data points belonging to two classes, optimally. Here, w defines the hyperplane that learns from the training data points using stochastic gradient descent (SGD) method. We express the objective function used in SGD as a feature vector $\mathbf{x}_i$ belonging to X and their respective labels y_i having a value between 0 to 1. A regularisation constant α penalises the model having a higher complexity and the loss function L determines the objectives of SGD. We have used the soft-margin SVMs, where L denotes the hinge loss, which is represented by Equation 6.

$$L(t, y) = max(0, 1 - t_y) \tag{6}$$

The objective function used is as follows:

$$E(\mathbf{w}) = \frac{1}{p}\sum_{i=1}^{p} L(y_i, \mathbf{w}^T\mathbf{x}_i) + a\,\|\mathbf{w}\|_2 \tag{7}$$

where, p is the number of training points and $\mathbf{w}$ represents the weight. During detection, an instance is labelled 'malicious' if

$$\mathbf{w}^T\mathbf{x} > \Lambda \tag{8}$$

Testing instances consist of those binary executables which are marked benign by the system call detector.

IV. EXPERIMENTAL RESULTS

In this section, we present the details of our experiment to show the performance of the proposed approach. First, we present the experimental setup, and then we discuss the performance of our model. We show that our approach can

[8]https://www.virustotal.com/

perform better by comparing with the state-of-art methods. We also assign proper class to the malicious samples after finding that it is malicious. Lastly, we demonstrate via a case study that effort required in Abuse case and STS-Tool approach can be reduced using our proposed model.

A. Setup

We implement all the experiments on one computer. The version of the CPU is Intel i5-3470 @ 3.20 GHz, the RAM is 16.0 GB and the operating system is Windows 10. We implement our approach using Python programming language in which the matrix computations are dependent on numpy.

B. API calls

To extract API calls, we use Ollydbg to disassemble binary executables and then obtain API calls. We collect API calls from 9914 binary executables having 100 to 10,000 number of API calls in each binary executable. In a few binary executables, we are not able to extract API calls; thus, we use a vector of all zero values to represent them.

C. Selection of Meta-features

To accelerate the convergence speed of our proposed model, we use a correlation matrix with heatmap to select the best meta-features. The heatmaps obtained from both malicious and benign binaries available in the dataset having six meta-features are shown in Figure 2 and 3 respectively. In the case of heatmap of malware binaries, coefficient of variation is highly correlated with variation of trend, oscillation is highly correlated with square wave. We have removed the regularity feature, since it is neutrally correlated with other features. And, kurtosis is not also considered since it's not strongly correlated with others as (coefficient of variation, variation of trend) and (oscillation, square wave) does. So, we select 4 features from malware binaries namely coefficient of variation, variation of trend, oscillation and square wave. In the case of heatmap of benign binaries, coefficient of variation is highly correlated with variation of trend and oscillation is highly correlated with square wave. So, we select the same 4 features available in malware binary executable. So, the final meta-features used for the final dataset creation are coefficient of variation, variation of trend, oscillation and square wave.

D. Performance analysis of malware detection

The parameters and metrics which we use for performance analysis of our proposed model are classification accuracy, detection false positive rate, detection true negative rate, detection false negative rate, detection precision, detection recall, F1-score, training time cost and detection time cost. The classification accuracy is calculated using Equation 9. Recall of malware detection model is the true positive rate evaluated using Equation 10, where True Positive (TP) is the number of malware instances correctly classified and False Negative (FN) is the number of malware cases misclassified as benign one. TNR is the true negative rate, which is evaluated using Equation 11 , where False Positive (FP) is

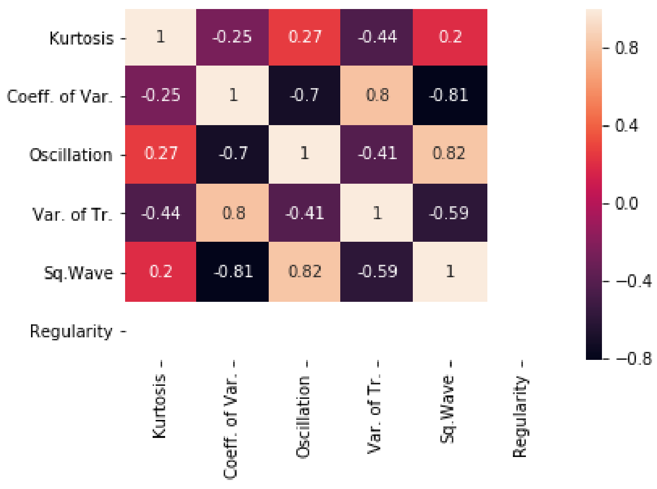


Fig. 2. Heatmap of malware binaries

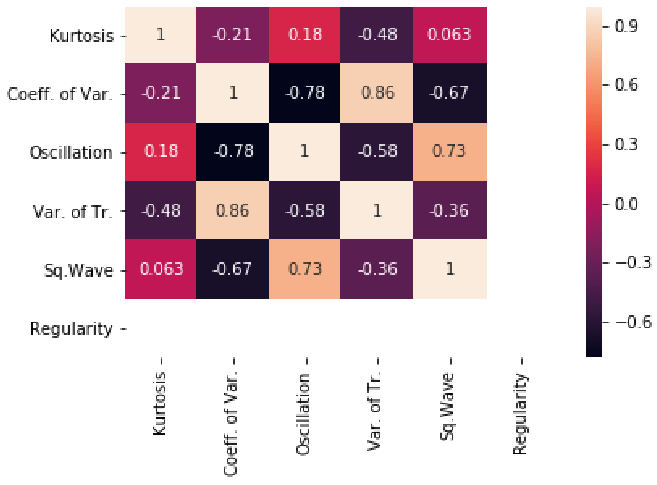


Fig. 3. Heatmap of benign binaries

the number of benign instances which are misclassified as malware binaries and True Negative (TN) is the number of benign instances which are correctly classified. FPR represents the false positive rate, FNR represents the false negative rate, Precision represents malware detection precision, and F1-score is computed using Precision and Recall, which are shown in Equations 12-15.

$$accuracy = \frac{TP+TN}{TP+FN+TN+FP} \tag{9}$$

$$TPR(Recall) = \frac{TP}{TP+FN} \tag{10}$$

$$TNR = \frac{TN}{FP+TN} \tag{11}$$

$$FPR = \frac{FP}{FP+TN} \tag{12}$$

TABLE III
PERFORMANCE EVALUATION OF OUR MODEL

Sl. No.	Performance Parameters	Value
1	Accuracy (%)	95.2
2	Recall (%)	93
3	TNR (%)	88
4	FPR (%)	12
5	FNR (%)	7
6	Precision (%)	95.4
7	F1-score (%)	94.1
8	Detection Time (s)	0.03
9	Training Time (s)	160

$$FNR = \frac{FN}{FN + TP} \tag{13}$$

$$Precision = \frac{TP}{FP + TP} \tag{14}$$

$$F1 - score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{15}$$

The performance evaluation of our model is shown in Table III.

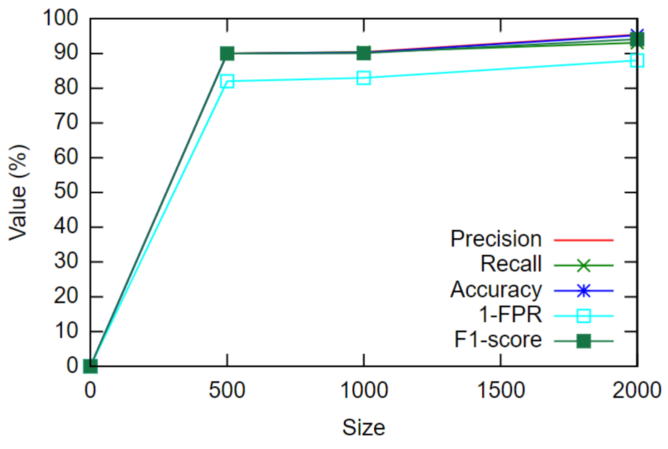


Fig. 4. Stability evaluation of accuracy

E. Stability evaluation of malware detection

Since malware variants are swiftly growing in numbers, we always face issue that training samples are always smaller than the volume of the test dataset. When the detection set contains numerous binary executables and the training set is smaller, then the ratio of training/(training+detection) is small and will lead to limited accuracy. So, here we evaluate the stability of our proposed malware detection model by testing with different volumes of training sets. The various sizes of our training sets is 500, 1000, and 2000. Figure 4 represents the precision, recall, 1-FPR, and the F1-score of our proposed detection model for different sizes. From Figure 4, we can infer that our model is stable when the ratio of training/(training+detection) is less than 0.2.

Figure 5 shows the training time of our approach for different sizes of data set. We observe that the training time

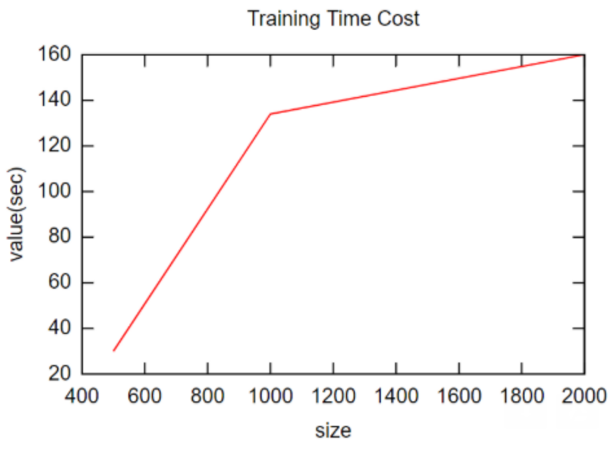


Fig. 5. Training time of our approach for different volume of data sets

is moving in a steep way upto size 1000, and then smoothens after size, 1200.

F. Classification accuracy of malware families

We evaluate the classification accuracy of malware families evaluated using one-versus-all strategy SVM[9]. To make the evaluation quick, we use the average anomaly score of each sample only, reducing it to single feature. Results are represented in Table VIII. We observe that detecting virus is not feasible, but have better accuracy for backdoor and Trojan. Worms can be easily labelled by our model.

G. Case Study

We have considered a case of travel planning scenario to demonstrate how our proposed model optimizes the effort required in Abuse case and STS-Tool approaches. Abuse case is determined by those interactions between an actor and the system which can harm the resources associated with actors, stakeholders or systems. STS model comprises of three complementary views: social, information and authorisation. These three views together help plan a model for system-at-hand. Now, we discuss below the effort optimization process for the above case study.

We identify the agents and roles in the model, for example, Tourist, Travel Agency Service (TAS) and Hotel are roles, while Hotel Service, Bob, Payment service and Amadeus Service (AS) are agents. Then, we identify the goals of agents and roles, as described in Table V. We design a goal model of an actor that ties together the goals and documents, For example, an actor possesses documents; an actor needs documents to fulfil a goal; an actor produces documents during goal fulfilment; an actor modifies a document while fulfilling a goal. The goal model is described in Table VI. Using the technique of goal delegation, we can transfer the fulfilment responsibility of the goal from one actor to another. Also, the

[9]https://machinelearningmastery.com/one-vs-rest-and-one-vs-one-for-multi-class-classification/

TABLE IV
SECURITY REQUIREMENTS FOR TRAVEL PLANNING SCENARIO

Roles	Security Requirements	Requester	Why security requirement is handled by our model
TAS	non-repudiation-of-acceptance (Tourist, TAS, Tickets booked)	Tourist	Trojans and Worms are detectable
Tourist	non-repudiation-of-delegation (Tourist, TAS, Tickets booked)	TAS	Trojans and Worms are detectable
TAS	true-redundancy-multiple-actor (Tickets booked)	Tourist	Unhandled
Hotel	no-redelegation (hotel booked)	Tourist	Unhandled
AS	integrity-of-transmission (provided(TAS, AS, Itinerary details))	TAS	Unhandled
All agents	not-achieve-both (eticket generated, credit card verified)	Org	Unhandled
AS	availability (flight ticket booked, 85%)	TAS	Backdoor is detectable
Tourist	delegated To(trustworthy(Hotel))	Tourist	Backdoors and Trojans are detectable
TAS	need-to-know (Personal data and Itinerary, Tickets booked)	Tourist	Unhandled
TAS	non-modification (Personal data and Itinerary)	Tourist	Trojans and Viruses are detectable
TAS	non-production (Personal data and Itinerary)	Tourist	Trojans and Backdoors are detectable
TAS	non-disclosure (Personal data and Itinerary)	Tourist	Trojans and Backdoors are detectable

TABLE V
GOAL OF AGENTS

Name of Agent or Role	Goals
Amadeus Service	Eticket generated and credit card verified
TAS	Flight ticket booked and Train ticket booked
Payment Service	Prepayment made
Hotel Service	Room selected and Prepayment made
Tourist	Tickets booked and Hotel booked
Hotel	Hotel booked and Room selected

TABLE VI
GOAL MODEL

Agent or Role	Goal	Asset rules
AS	Eticket generated	Produce flight tickets
AS	Eticket generated	Need itinerary details
TAS	Flight ticket booked	Need itinerary details
TAS	Train ticket booked	Produce tickets
TAS	Tickets booked	Need travelling order
Tourist	Trip planned	Produce travelling order
Tourist	Trip planned	Need travelling order
Tourist	Trip planned	Modify travelling order
Tourist	Hotel booked	Need IdDoc copy
Hotel	Hotel booked	Need and modify IdDoc copy

information is exchanged between actors, named document provision. Goal delegations and document provisions for every roles and agent are shown in Table VII. We eliminate some security issues as they are handled by our proposed malware detection model, such as non-repudiation of delegation or

TABLE VII
GOAL DELEGATIONS AND DOCUMENT PROVISIONS

Agent or Role	Delegations or Provisions
TAS	Delegates flight ticket booked to AS
TAS	Provisions itinerary details to AS
Hotel Service	Delegates prepayment made to Payment Service
Hotel	Delegates hotel booked to Hotel Service
Tourist	Delegates hotel booked to Hotel
Tourist	Provisions IdDoc copy to Hotel
Tourist	Delegates tickets booked to TAS
Tourist	Provisions traveling order to Hotel

TABLE VIII
CLASSIFICATION ACCURACY OF MALWARE FAMILIES

Sl. No.	Actual class	Accuracy (%)
1	Backdoor	66
2	Worm	99
3	Trojan	70
4	Virus	1

acceptance, trustworthiness, and availability. Only we consider no-redelegation, integrity of transmission, confidentiality of transmission, separation of duties, combination of duties and redundancy concerns. Security requirements for this example is described in Table IV.

There may be malware which can cause TAS to non-repudiate acceptance from Tourist. This type of malware is easily discovered by our proposed model. Similarly, the malware causing non-repudiation of delegation by Tourist, when requested by TAS, is also detected by our model. TAS tries to book tickets using either railways or airways, so true-redundancy-multiple-actor security requirement is there, which is uncoverable by our model. Hotel cannot redelegate the request done by tourist, and it is undetected by our model. Amadeus service's integrity maintenance can be done by applying intrusion detection in network channels, which cannot be done on hosts. Agent's plan of action is previously defined and is undetectable by our model. Non-Availability of any service for specific duration is easily detected by our model. Trustworthiness is easily insured as our proposed model will detect the malware which results in suggestions without considering ratings of the desired results. Since the data is stored in database or files, the modification, production and disclosure are easily recognized by our model. However, the information which is pre-required is undetected by our model.

For the Abuse case, we see that its assets'safety condition is embedded within the STS-Tool approach. Hence, Abuse case approach is not needed if we follow the STS-Tool approach. After analysing the case study, we have seen that our proposed model can tackle 7 out of 12 security requirements as shown in Table IV. Henceforth, the effort required for designing security model is reduced to approx 50%.

TABLE IX
COMPARISON OF PERFORMANCE OF OUR APPROACH WITH EXISTING STATE-OF-ART APPROACHES

Method	Dataset	Accuracy(%)	Precision(%)	Recall(%)	1-FPR(%)	F1-score(%)	Detection time(s)	Training time(s)
SVM [18]	Android Genome	83.5	86.5	80.6	87.4	83.4	0.001	31
NB [17]	Android Genome	79.7	78.3	82.2	77.2	80.2	0.005	134
CNN [16]	VirusShare	83.8	82.5	85.8	81.8	84.1	0.053	213467
SVM [14]	VirusTotal	86.6	92.4	79.8	93.4	85.6	0.094	179
Graph theory [15]	MobileSandbox	93.6	92.3	94	92.1	91.1	0.001	NA
Our approach	VirusTotal	95.2	95.4	93	88	94.1	0.03	160

V. COMPARISON WITH RELATED WORK

We have compared the performance of our model with that of several state-of-art methods and shown in Table IX. By comparing with the other state-of-art methods, we observe that our approach significantly improves the classification accuracy, the detection precision and F1-score while retaining the detection speed. Accuracy is better than other models, thus it can be used for industrial malware detection. Precision is also higher, which means that 95.4% of the results are relevant results. Recall is 93%, which is lower than the recall evaluated by using Graph theory method. It means that 93% of total relevant results are correctly classified by our model. In most problems, we either give maximum priority to higher precision or recall, which depends upon the problem under consideration. In general, we use a simple metric which will use precision and recall to maximize the number to improve the model. That metric is known as F1-score, which is the harmonic mean of precision and recall. Our model is imperative in terms of F1-score, which is 94.1%. Specificity is equivalent to "1-FPR", which means that instances which are benign and being labelled as benign is 88%. It is a subsidiary parameter, as it's lower value can only block the benign process. However, our main objective of not executing malware executable on the host will not be affected by lower specificity. In terms of detection time, our model lies behind models using SVM [18], NB (Naive Bayes) [17] and Graph theory [15] based approach. But, those models, i.e. SVM model [18], NB model [17] and Graph theory model [15] have lower detection accuracy, which implies that there is poor trade-off between detection time and accuracy. Only the models using SVM [18], NB [17] and Graph theory [15] have lower training time than our proposed model. But, models using SVM [18], NB [17] and Graph theory [15] have lower detection accuracy than our proposed model. So, we draw inference that two-phase approach of detection, representing API calls in the form of meta-features and using average anomaly score of instances for classification are effective in designing an industrial-applicable malware detector.

VI. THREATS TO VALIDITY

In this section, we identify some possible threats to the validity of our approach. Malware is generally packed using packers, but sometimes they are not detected by malware detectors. Majority of the packers can be unpacked using numerous techniques and tools such as PolyUnpack [1], which recover the original source file. For API calls based detection technique to work, unpacking techniques should always provide the original code. Obfuscation in the software will make it tough to de-obfuscate it. So, our discussion in this section is concerned about the limitation caused by obfuscation.

Obfuscation is a semantic preserving transformation which results in obfuscated programs. When we collect the obfuscated programs from the same source, then it is similar. Our model can detect obfuscation to some degree. Obfuscation can be of several types such as identifier renaming, junk code injection, control flow based obfuscation, etc. Identifier renaming obfuscation renames variables but it cannot impact the representations of binary executable. The junk code injection can change the distributions but can be easily detected and denoised. Control flow based obfuscation changes the control flow graph leading to error in control flow based detection. Some noisy instructions are added, but still, it is similar to the original instructions. In general, our approach can resist limited number of mistakes caused by obfuscation. Adversarial attack, which is a limitation of machine -learning based application is partially handled using our incremental model. But, we need to focus on many issues to obtain an appropriate technique for keeping our model safe from adversarial attacks.

We consider the malware targetting windows OS and NITRSCT is also limited to Windows OS. We assess the performance of our model on windows based dataset only. Android OS and Linux targetting malware may remain undetected.

VII. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed an incremental malware detection model for meta-feature API and system call sequence, which effectively identified malware. We use the system call dataset generated using NITRSCT. Then, the API calls are collected by disassembling malicious executables and legitimate ones. Then anomaly score sequence of API calls is generated using N-HTM. A dataset is created using anomaly score of API calls, which contains six meta-features. Number of meta-features are reduced using correlation matrix with heatmap. After the final meta-features are selected, then the proposed model is used to detect malware. We also provided the labels to malware according to its class which can analyse the feature selection process of the malicious samples. Through a case study, we have demonstrated that our proposed model eliminates the need for having Abuse case approach and reduces the effort required, to around 50% for STS-Tool. Our model smoothly used meta-feature API calls (high-level features) and system calls to cover characteristics of malware and improved detection precision and F1-score by more than 3%. Real-life experimental results have shown that our approach achieved 95.2% accuracy and have detection speed lower than 0.1s. In addition, training time is also lower which doesn't increase the time complexity of the model.

As future work, we will ensemble static analysis based features with dynamic analysis to reduce dependency on unpacking tools. We will try to apply this model to detect the malware present in Android OS, anti-fraud systems, other domains, etc. We will try to improve the classification accuracy of each malware family. Exploration of a few more meta-features will be done by us to find a minimal number of features which will provide higher accuracy.

References

[1] P. Royal, M. Halpin, D. Dagon, R. Edmonds, and W. Lee, "Polyunpack: Automating the hidden-code extraction of unpack-executing malware," in *2006 22nd Annual Computer Security Applications Conference (ACSAC'06)*. IEEE, 2006, pp. 289–300. [Online]. Available: https://doi.org/10.1109/acsac.2006.38

[2] K. Yan, Z. Ji, and W. Shen, "Online fault detection methods for chillers combining extended kalman filter and recursive one-class svm," *Neurocomputing*, vol. 228, pp. 205–212, 2017. [Online]. Available: https://doi.org/10.1016/j.neucom.2016.09.076

[3] C. S. Sharma, S. N. Panda, R. P. Pradhan, A. Singh, and A. Kawamura, "Precipitation and temperature changes in eastern india by multiple trend detection methods," *Atmospheric research*, vol. 180, pp. 211–225, 2016. [Online]. Available: https://doi.org/10.1016/j.atmosres.2016.04.019

[4] A. K. Chanda, C. F. Ahmed, M. Samiullah, and C. K. Leung, "A new framework for mining weighted periodic patterns in time series databases," *Expert Systems with Applications*, vol. 79, pp. 207–224, 2017. [Online]. Available: https://doi.org/10.1016/j.eswa.2017.02.028

[5] Z. Ji, B. Wang, S. Deng, and Z. You, "Predicting dynamic deformation of retaining structure by lssvr-based time series method," *Neurocomputing*, vol. 137, pp. 165–172, 2014. [Online]. Available: https://doi.org/10.1016/j.neucom.2013.03.073

[6] S. Ahmad, A. Lavin, S. Purdy, and Z. Agha, "Unsupervised real-time anomaly detection for streaming data," *Neurocomputing*, vol. 262, pp. 134–147, 2017. [Online]. Available: https://doi.org/10.1016/j.neucom.2017.04.070

[7] P. Kishore, S. K. Barisal, and S. Vaish, "Nitrsct: A software security tool for collection and analysis of kernel calls," in *TENCON 2019-2019 IEEE Region 10 Conference (TENCON)*. IEEE, 2019, pp. 510–515. [Online]. Available: https://doi.org/10.1109/tencon.2019.8929513

[8] G. McGraw, "Software security," *IEEE Security & Privacy*, vol. 2, no. 2, pp. 80–83, 2004. [Online]. Available: https://doi.org/10.1109/msecp.2004.1281254

[9] E. Paja, F. Dalpiaz, M. Poggianella, P. Roberti, and P. Giorgini, "Sts-tool: socio-technical security requirements through social commitments," in *2012 20th IEEE International Requirements Engineering Conference (RE)*. IEEE, 2012, pp. 331–332. [Online]. Available: https://doi.org/10.1109/re.2012.6345830

[10] W. Wang, Z. Gao, M. Zhao, Y. Li, J. Liu, and X. Zhang, "Droidensemble: Detecting android malicious applications with ensemble of string and structural static features," *IEEE Access*, vol. 6, pp. 31 798–31 807, 2018. [Online]. Available: https://doi.org/10.1109/access.2018.2835654

[11] C. K. Patanaik, F. A. Barbhuiya, and S. Nandi, "Obfuscated malware detection using api call dependency," in *Proceedings of the First International Conference on Security of Internet of Things*. ACM, 2012, pp. 185–193. [Online]. Available: https://doi.org/10.1145/2490428.2490454

[12] J. Huang, X. Zhang, L. Tan, P. Wang, and B. Liang, "Asdroid: Detecting stealthy behaviors in android applications by user interface and program behavior contradiction," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 1036–1046. [Online]. Available: https://doi.org/10.1145/2568225.2568301

[13] M. Fan, J. Liu, X. Luo, K. Chen, Z. Tian, Q. Zheng, and T. Liu, "Android malware familial classification and representative sample selection via frequent subgraph analysis," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 8, pp. 1890–1905, 2018. [Online]. Available: https://doi.org/10.1109/tifs.2018.2806891

[14] R. Canzanese, S. Mancoridis, and M. Kam, "System call-based detection of malicious processes," in *2015 IEEE International Conference on Software Quality, Reliability and Security*. IEEE, 2015, pp. 119–124. [Online]. Available: https://doi.org/10.1109/qrs.2015.26

[15] J. Zhang, Z. Qin, K. Zhang, H. Yin, and J. Zou, "Dalvik opcode graph based android malware variants detection using global topology features," *IEEE Access*, vol. 6, pp. 51 964–51 974, 2018. [Online]. Available: https://doi.org/10.1109/access.2018.2870534

[16] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. K. Nicholas, "Malware detection by eating a whole exe," in *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[17] B. Kang, S. Y. Yerima, K. McLaughlin, and S. Sezer, "N-opcode analysis for android malware classification and categorization," in *2016 International Conference On Cyber Security And Protection Of Digital Services (Cyber Security)*. IEEE, 2016, pp. 1–7. [Online]. Available: https://doi.org/10.1109/cybersecpods.2016.7502343

[18] J. G. de la Puerta, B. Sanz, I. Santos, and P. G. Bringas, "Using dalvik opcodes for malware detection on android," in *International Conference on Hybrid Artificial Intelligence Systems*. Springer, 2015, pp. 416–426. [Online]. Available: https://doi.org/10.1093/jigpal/jzx031

[19] E. Garoudja, F. Harrou, Y. Sun, K. Kara, A. Chouder, and S. Silvestre, "Statistical fault detection in photovoltaic systems," *Solar Energy*, vol. 150, pp. 485–499, 2017. [Online]. Available: https://doi.org/10.1016/j.solener.2017.04.043

[20] L. Dong, L. Shulin, and H. Zhang, "A method of anomaly detection and fault diagnosis with online adaptive learning under small training samples," *Pattern Recognition*, vol. 64, pp. 374–385, 2017. [Online]. Available: https://doi.org/10.1016/j.patcog.2016.11.026

[21] J. C. M. Oliveira, K. V. Pontes, I. Sartori, and M. Embiruçu, "Fault detection and diagnosis in dynamic systems using weightless neural networks," *Expert Systems with Applications*, vol. 84, pp. 200–219, 2017. [Online]. Available: https://doi.org/10.1016/j.eswa.2017.05.020

[22] M. Gan, C. P. Chen, H.-X. Li, and L. Chen, "Gradient radial basis function based varying-coefficient autoregressive model for nonlinear and nonstationary time series," *IEEE Signal Processing Letters*, vol. 22, no. 7, pp. 809–812, 2014. [Online]. Available: https://doi.org/10.1109/lsp.2014.2369415

[23] S. Kanarachos, S.-R. G. Christopoulos, A. Chroneos, and M. E. Fitzpatrick, "Detecting anomalies in time series data via a deep learning algorithm combining wavelets, neural networks and hilbert transform," *Expert Systems with Applications*, vol. 85, pp. 292–304, 2017. [Online]. Available: https://doi.org/10.1016/j.eswa.2017.04.028

[24] M. K. Cain, Z. Zhang, and K.-H. Yuan, "Univariate and multivariate skewness and kurtosis for measuring nonnormality: Prevalence, influence and estimation," *Behavior Research Methods*, vol. 49, no. 5, pp. 1716–1735, 2017. [Online]. Available: https://doi.org/10.3758/s13428-016-0814-1

[25] G. R. Iannotti, F. Pittau, C. M. Michel, S. Vulliemoz, and F. Grouiller, "Pulse artifact detection in simultaneous eeg–fmri recording based on eeg map topography," *Brain topography*, vol. 28, no. 1, pp. 21–32, 2015. [Online]. Available: https://doi.org/10.1007/s10548-014-0409-z

[26] K. N. Rajesh and R. Dhuli, "Classification of ecg heartbeats using nonlinear decomposition methods and support vector machine," *Computers in biology and medicine*, vol. 87, pp. 271–284, 2017. [Online]. Available: https://doi.org/10.1016/j.compbiomed.2017.06.006

[27] R. K. Tripathy, S. Deb, and S. Dandapat, "Analysis of physiological signals using state space correlation entropy," *Healthcare technology letters*, vol. 4, no. 1, pp. 30–33, 2017. [Online]. Available: https://doi.org/10.1049/htl.2016.0065

[28] P. Marwaha and R. K. Sunkaria, "Complexity quantification of cardiac variability time series using improved sample entropy (i-sampen)," *Australasian physical & engineering sciences in medicine*, vol. 39, no. 3, pp. 755–763, 2016. [Online]. Available: https://doi.org/10.1007/s13246-016-0457-7

[29] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and computing*, vol. 14, no. 3, pp. 199–222, 2004. [Online]. Available: https://doi.org/10.1023/b:stco.0000035301.49549.88