**EXISTING SYSTEM:**

Malware detection methods can be static or dynamic. In dynamic malware detection approaches, the program is executed in a controlled environment (e.g., a virtual machine or a sandbox) to collect its behavioral attributes such as required resources, execution path, and requested privilege, in order to classify a program as malware or benign. Static approaches (e.g., signature-based detection, byte-sequence n-gram analysis, opcode sequence identification and control flow graph traversal) statically inspect a program code to detect suspicious applications. David et al proposed Deepsign to automatically detect malware using a signature generation method. The latter creates a dataset based on behaviour logs of API calls, registry entries, web searches, port accesses, etc, in a sandbox and then converts logs to a binary vector. They used deep belief network for classification and reportedly achieved 98.6% accuracy. In another study, Pascanu et al. Proposed a method to model malware execution using natural language modeling. They extracted relevant features using recurrent neural network to predict the next API calls. Then, both logistic regression and multi-layer perceptrons were applied as the classification module on next API call prediction and using history of past events as features. It was reported that 98.3% true positive rate and 0.1% false positive rate were achieved. Demme et al. examined the feasibility of building a malware detector in IoT nodes’ hardware using performance counters as a learning feature and K-Nearest Neighbor, Decision Tree and Random Forest as classifiers. The reported accuracy rate for different malware family ranges from 25% to 100%. Alam et al. applied Random Forest on a dataset of Internet-connected smartphone devices to recognize malicious codes. They executed APKs in an Android emulator and recorded different features such as memory information, permission and network for classification, and evaluated their approach using different tree sizes. Their findings showed that the optimal classifier contains 40 trees, and 0.0171 of mean square root was achieved.

**PROPOSED SYSTEM:**

Junk code injection attack is a malware anti-forensic technique against OpCode inspection. As the name suggests, junk code insertion may include addition of benign OpCode sequences, which do not run in a malware or inclusion of instructions (e.g. NOP) that do not actually make any difference in malware activities. Junk code insertion technique is generally designed to obfuscate malicious OpCode sequences and reduce the ‘proportion’ of malicious OpCodes in a malware In our proposed approach, we use an affinity based criteria to mitigate junk OpCode injection anti-forensics technique. Specifically, our feature selection method eliminates less instructive OpCodes to mitigate the effects of injecting junk OpCodes. To demonstrate the effectiveness of our proposed approach against code insertion attack, in an iterative manner, a specified proportion (f5%, 10%, 15%, 20%, 25%, 30%g) of all elements in each sample’s generated graph were selected randomly and their value incremented by one. For example, in the 4th iteration of the evaluations, 20% of the indices in each sample’s graph were chosen to increment their value by one. In addition, in our evaluations the possibility of a repetitive element selection was included to simulate injecting an OpCode more than once. Incrementing Ei;j in the sample’s generated graph is equivalent to injecting OpCodej next to the OpCodei in a sample’s instruction sequence to mislead the detection algorithm. Algorithm 2 describes an iteration of junk code insertion during experiments, and this procedure should repeat for each iteration of k-fold validation. To show the robustness of our proposed approach and benchmark it against existing proposals, two congruent algorithms described in Section 1 are applied on our generated dataset using Adaboost as the classification algorithm. All evaluations were conducted using MATLAB R2015a running on a Microsoft Windows 10 Pro personal computer powered by Intel Core i7 2.67GHz and 8GB RAM. A 10-fold cross validation was used in the validating, and the comparative summary is presented in Table 1. It is clear that our proposed approach outperforms the proposals of Hashemi et al.and Santos et al. . The approach of Santos et al. is a basic and commonly-known OpCode based malware detection algorithm and the approach of Hashemi et al. is the most similar in terms of using eigenspace as the basis. Accuracy is a general criteria for evaluating performance of an algorithm for both malware and benign class identification. The proposed approach achieves a high accuracy of 99.68%, while the approaches of Hashemi et al. and Santos et al.respectively achieve 98.59% and 95.91% accuracy. Recall or detection rate is an important criteria and the proposed approach achieves 98.37%, in comparison to 81.55% and 77.70% for the other two approaches. Our proposed approach also outperforms the approaches of Hashemi et al. and Santos et al., in terms of precision rate and F-Measure. Utilizing class-wise feature selection appears to result in beneficial features of minor class to be more effective during classification phase. Also, using Formulation to calculate OpCode’s distance leads to the ability to represent more OpCode sequence patterns in the sample’s graph. It also appears that employing deep neural networks for classification leads to a better classifier.