National College of Ireland

Adversarial Resilience in Malware Detection: A Two-Stage Structural Analysis Approach for Robust Cybersecurity

MSc Research Project
Msc Artificial Intelligence

Pavithrasri Udayakumar
Student ID: X22182730

School of Computing
National College of Ireland

Supervisor: Anh Duong Trinh

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Pavithrasri Udayakumar
Student ID:	X22182730
Programme:	Msc Artificial Intelligence
Year:	2024
Module:	MSc Research Project
Supervisor:	Anh Duong Trinh
Submission Due Date:	04/01/2024
Project Title:	Adversarial Resilience in Malware Detection: A Two-Stage Structural Analysis Approach for Robust Cybersecurity
Word Count:	6030
Page Count:	19

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	5th January 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Adversarial Resilience in Malware Detection: A Two-Stage Structural Analysis Approach for Robust Cybersecurity

Pavithrasri Udayakumar
X22182730

Abstract

In the fast-changing world of cybersecurity, more and better computer virus attacks are becoming a big issue for keeping digital security safe. This paper suggests a strong two-step system for detecting Malware. It combines simple malware detection and advanced harmful software detection to better identify and fight all types of unwanted programs, even when they use clever hidden methods. Using big computer learning techniques like Logistic Regression, Random Forest and Decision Tree plus the deep thinking skills of Long Short-Term Memory (LSTM), this system shows high results in correctly putting harmful software or normal files. The test, done on a big and complete BODMAS data set, shows the system's strength against harmful attacks from malware creator. It proves it can change quickly to deal with fast-moving malware or bugs in computer systems. The LSTM model stands out, getting it right 82% of the time. This shows that it is very good at spotting hard to find patterns linked with bad behavior. Even though it's been successful, the system knows that we need a bigger and more varied set of information to make itself even better. The study helps make computer security stronger by suggesting a new and changeable way to spot malware. It shows the need for constant improvement and growth in order to properly fight off changing threats over time.

1 Introduction

With the internet, we are now more dependent on technology than ever before. With the popularity of personal computers, smartphones and other so-called intelligent devices in today's world. we have a lot of sensitive information that criminals crave. Malware attacks have become an ever-changing and keenly fought battleground in the war between cybersecurity defenders, who are forever on their guard against malicious actors. Malware is one of the biggest obstacles to our digital security, because its threat landscape changes constantly.

Some types of computer attacks, shown in 1, are a big danger when it comes to fighting wars on the internet. The current malware world makes things hard and big. It's a very complex thing, so it shows that online platforms have huge safety problems according to Ferrag et al. (2020). Bad computer programs known as malware are made to hurt computers or networks. They often use money-grabbing ways for their goal. Importantly, attacks from harmful computer programs are now more focused on the Internet

of Things (IoT) devices, medical tools and systems that control infrastructure in both natural places like rivers or mountains but also man-made buildings. These attacks come with growing numbers as reported Wickramasinghe et al. (2018). The complexity of today's spyware makes it tough to spot. This is because its code and behavior are always changing, so hard for us to catch in action. Because of the spread of Malware, we now need to get much stronger protectionXu et al. (2019). Although antivirus soft-

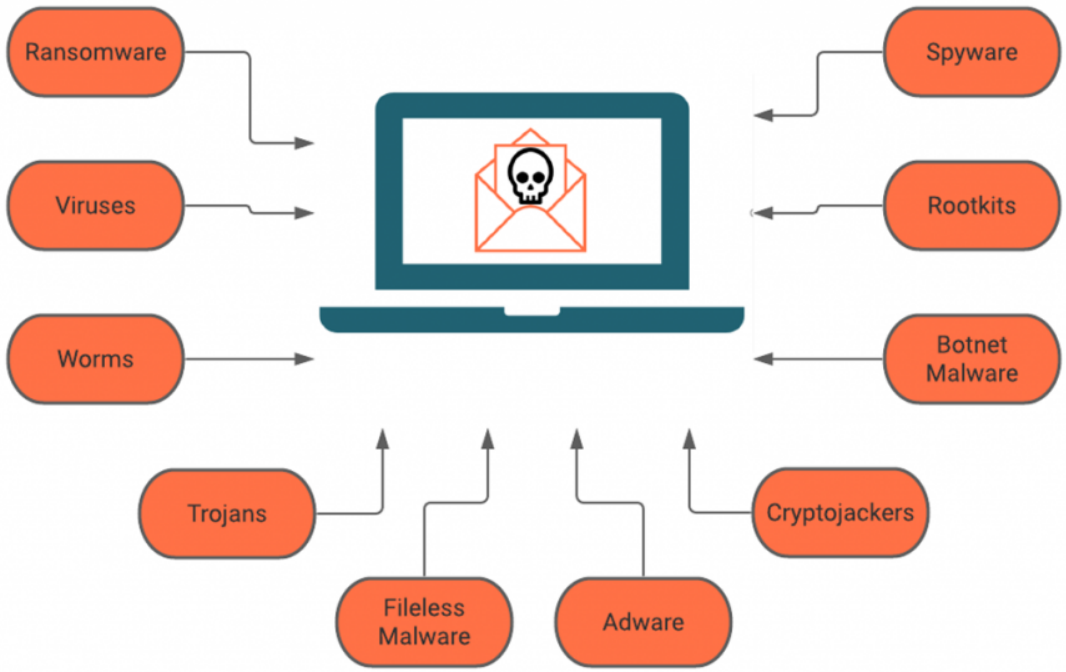

Figure 1: Different type of Malware Attacks

ware guards our computers, it doesn't always recognize malicious attacks. Now-legacy antivirus Signature-based checks, once the pillars of cybersecurity, are no longer effective against many types of malware. Although it uses a database of known signatures, this makes the software susceptible to zero-day attacks and has no ability to detect obfuscated packed or polymorphic malware. To effectively combat today's dynamic malware landscape, a multilayered security approach that combines advanced detection techniques, user education and proactive threat mitigation is needed.

In fact, a simple case of this Wong and Stamp (2006) is the paper that provided an extensive analysis of metamorphic malware. From their point of view, what they found was evidence to demonstrate that direction signatures had reached its limit; it couldn't properly respond to such dynamic code structures.

In addition, the extremely hostile nature of the cybersecurity environment means that malware may be designed specifically to avoid detection by ML. The idea of adversarial machine learning, as discussed by Biggio et al. (2013) highlights the fragility of ML models to manipulation or attack. If the system relies only on input features for spotting bad users, attackers can add small changes that are hard to see and unimportant in normal human sight. These changes, even though they don't affect how well we recognize things, allow for false looks that seem real.The system would thus lose all vision of their bad intent.

Adversarial content refers to the intentional and deliberate malware components designed

precisely to make security measures inoperative. Identifying, understanding and then expunging these threatening components are important steps in strengthening our digital defenses against continually changing cyber threats.
Such methods include the use of sophisticated machine learning algorithms, behavioral analysis and anomaly detection techniques. The aim of unrolling the tangled skein that is adversarial content is to provide cybersecurity experts with knowledge and weapons for conducting preventative warfare against digital ecosystems suffering from ever more polished malware insights.
The research suggests a two-stage detection process. In the first stage, system spots files containing unusually high proportion of beneficial segments. In the second stage, a more detailed analysis is conducted to determine whether malicious intent exists based on chunk structure and distribution. By making use of these differences, this research endeavors toward building an accurate detector that can divide benign files from adversarial examples of malware. The proposed method involves observing how localized the injected content is within executables. It's based on a given that adversarial examples share similar structural patterns. This research also helps to further develop malware detection methodologies, pointing the way toward filling one of the most important gaps in static deep learning-based detectors and bringing increased robustness against increasingly complex evasion techniques as cybersecurity continues its rapid evolution.

2 Related Work

2.1 Datasets:

The BODMAS dataset is one of only a few large-scale efforts aimed at training machine learning systems to perform malware analysis. Yang et al. (2023) work includes the corpus in its entirety, making it an important early contribution to this field. In cybersecurity, temporal analysis faces limitations due to sparse datasets, hindering comprehensive studies on the evolving nature of malware across different time periods with detailed family information. To compensate for these shortcomings, BODMAS contains 57,293 malware and 77 report covers information about only selected malware families. We also seek to use this dataset to assist efforts at research, especially in the areas of exploring concept drift and changes over time amongst malware families.

Many malware analysis workflows now rely on machine learning models, and the BODMAS data set has become a vital resource in efforts to improve knowledge about how these programs behave over time. The experiment highlights the difficulties of an " open-world " scenario and concept drift, with a preliminary analysis showing how malware classifiers ' decision boundaries are constantly undergoing change due to new families appearing along with mutations in existing ones. The release of BODMAS gives the research community a powerful new tool for exploring movements in malware, and it provides an important contribution to temporal analysis applied to PE malware.

BODMAS dataset, which focuses on the requirement for fresh and carefully handled PE malware infection examples. Ma et al. (2021) does a very big study on how programs that are like malware against privacy data sort themselves into groups using ways they learn. The outcomes show all methods get slower when new ideas come about changing the way things work. Mat Kiah et al. (2010) talks about ways to find things wrong with computer privacy. He suggests a model that's like human body defense, and works

through special touch jolts from their system to stay healthy. Azeez et al. (2021) found a group way for recognizing Windows PE bad bugs with the best results when using brain-thinking systems and another classifying method called ExtraTrees.

2.2 Neural Network for Malware Detection:

In this study Raff et al. (2017), the author talk about using a special kind of algorithm called MalConv to find problems with computer programs - known as malware. They use machine learning and show that it could be important for further studies in this field. In contrast to regular ways of fighting viruses that look for specific signatures, MalConv uses deep analysis on two kinds of files. It focuses more on raw bits and bytes in these binary files, working hard towards building a strong and growing solution against possible attacks by harmful computer codes or worms (malware). The model helps with problems that only come up in finding malware software. These include issues related to more than two million steps of time and the way they get slowed down by things called batch normalization. The design, changed because it had to deal with changes in data position for tasks that can be run. Used layers and slot-finding methods so positions don't get seen as important differences. The model had trouble with batch normalization but was quick to tell accurate information and AUC. It shows it can work well on variable sets of data. The research shows how easy it is to understand the model using few active class maps and points out that binary executables have some problems because they use batch normalization in a certain situation. It's suggested to find new ways for buildings, fix big memory problems and use useful things other than just catching virus.

This study Catak et al. (2020) looks at how malware is changing, mainly focusing on metamorphic types. These are the most advanced forms of their kind. Old ways of fighting viruses using signatures can't find shape-changing malware software because it has a lot different design. The study introduces a new way to classify malware software based on how they behave. It uses a brand-new set of data about Windows actions made by different kinds of harmful programs. Using the Long Short-Term Memory (LSTM) method, a popular way to classify and study data in sequence. The research does very well with an accuracy level of up to 95% and has a good F1 score around 0.83. Importantly, the report gives not only a strong classification model but also one-of-a kind data for Windows operating systems. A big help is creating a special set of data, not used before. This makes it easier for more studies on how to find bad software or malware. Limitations are that they depend on behavioral signs and possible changes in how antivirus works, showing the continuous difficulty of dealing with complex malware dangers.

2.3 Machine Learning for Malware Detection:

The study Gibert et al. (2023) is about machine learning are full of ways to make malware finders better and stronger against anyone trying to trick them. They explained a lot about many methods that can help classifiers work harder in fighting malware things like viruses or hacks. Previous studies have mainly dealt with pictures and writing, but they haven't paid much attention to computer files that tell the machine what actions to take. In this new way, (de)randomized smoothing is used for malware detection. It's a well-known defense against patch attacks in image classifiers that has been changed here to suit the needs of fighting viruses. They came up with an innovative plan where

pieces could be spread out without having them meet each other while also covering one another properly by mixing and separating random. But the current studies have problems dealing with big programs and might not completely consider all kinds of ways to attack in battle. The suggested piece-based protection is okay, but it brings a swap between exact monitoring and tiny details. We need more study to make things right. It's for looking at harder designs and finding a better way of finding all kinds of bad computer programs, not just in simple ways like the ones we have now. These old models take up lots of memory on GPUs which is a problem that needs solving too with new ideas.

In the study Rahul et al. (2020), it is found that methods to find malware software have changed. Nowadays machine learning models are being used instead of old ways like looking for specific codes in bad programs. This has been happening over the last few years. The study categorizes malware detection techniques into three sections based on feature analysis: Static, Dynamic, and Hybrid. For example, special classifiers like K-Nearest Neighbor or Decision Trees help find harmful things faster than old sign method or hit and try methods. These include stuff from Support Vector Machine to Naïve Bayes Rules circles of trust in math world - machines all seem busy helping this happen better since it works well for them now! The survey knows the hardships that come with a static analysis when it must deal with ways of hiding things, and how much power is needed to work out dynamic analyses. Methods like PCA, Variance Threshold and Tree/Forest based selection help to pick out the important features. The study says we might look at doing learning with half-yes and no activity. This means using a mix of changes that stay the same over time, big data methods to handle lots of information all together, then building ways quicker real time about sorting stuff out fast as new goals for future work too! - Using static data (data that remains consistent) or dynamic features such as But, the drawbacks are like needing better real-time grouping. Also there's a need to balance between staying still and changing around. And that problem with especially tricky malware which hide their true self is tough too.

The article Mohammed et al. (2020) shows a new way to find malware programs using computer learning methods, like decision tree and random forest, inside an online app builder. The study recognizes that malware is getting worse and stresses how important it is to protect your private information. The suggested system, when working with a list of real and malware software program files gets very high accuracy rates. It did 98.9% for decision trees and 99.4% for random collections tied together using the process mentioned above: "making trees." The paper is missing a complete review of existing works. It doesn't properly explain the different ways used to detect malware software and how they compare with each other in fighting this issue. Also, the things that make this system not perfect are not talked about enough. These include its reliance on if training data is good and represents all kinds of stuff well, getting wrong results like finding something where there isn't one or missing what is actually important to find; additionally these systems always need regular updates because malware computer programs change over time. A more complete study and a detailed talk about flaws could make the suggested method stronger and better to use.

2.4 Malware Detection

In 2018, Suciu et al. (2019) and Kreuk et al. (2018) both highlighted that malware detection models can be easily tricked by using bad examples. Suciu et al. (2019) looks closely at the weaknesses of current ways to avoid attacks. They also ask if new attack methods can work better or not. However, Kreuk et al. (2018) suggests a new loss function in Adversarial Examples. This lets tiny addition sequences be put into any two-way files as extra content. Wang et al. (2021) went further and used ways to make code confusing. This was done with bad software detectors as the main target, making it hard for us to use features comparing or guess methods. These studies together show how important it is to have strong systems that can quickly find and stop bad code.

The study Yuste et al. (2022) on fake examples (AEs) for computer virus detectors using machine learning is getting more attention. People want to fix these weak points, especially the ones that use big math structures like MalConv. Many ways have been thought up to make AEs, such as hiding code methods and optimization tricks like Genetic Algorithms. Though these methods have shown success in avoiding malware detectors, there are significant limits. Current research shows problems like models for making AE can't be explained easily, biases in data sets used to test it out and the reliance on certain ways of finding malware software such as MalConv. People know that AE techniques can be used with commercial antivirus programs, but how effective they are might differ because of watching problems and setting issues on systems like VirusTotal. Moreover, the effect of training with bad examples on how strong machine learning models are is still being studied. There's also more work to be done in adapting AE techniques for use with other file formats or detection setups as future projects. In short, the current state of technology shows that we need good and easy-to-understand solutions to make machine learning malware detection systems stronger.

3 Methodology

3.1 Dataset

The BODMAS dataset is full of Portable Executable (PE) malware software examples. It's like a goldmine for researchers to use when trying out new ways to spot and understand these kinds of attacks. The dataset has a big collection of 57,293 harmful samples and 77,142 helpful ones. It shows how malware threats are always changing.

Table 1: Key Features of the BODMAS Dataset

Feature	**Detail**
Number of Samples	57,293 (malicious) + 77,142 (benign)
Temporal Coverage	August 2019 - September 2020
Malware Families	581
Feature Vectors	2,381 features using LIEF library

Researchers can follow how malware software changes over time using marked samples. Meanwhile, they use numbers to represent sample features that are standardized with the LIEF library for PE file study. Marking malware groups gives more use to the data set by

providing information about how common and what kind of various virus families they belongs too.

The BODMAS group helps in making and checking malware software detection ways. It also studies changes over time on these methods, how they grow with different families of malware species and understanding what makes them work this way. Its big size, different types and well-organized options make it a must use tool for researchers studying PE malware. As malware software keeps changing, the BODMAS set will always be an important part of study work to fight these growing dangers.

3.2 Data Preparation and Preprocessing

The good use of malware detective system starts with the quality of its teaching data. So, the task starts by collecting a big set of data with good and bad computer program files. The information is picked out very carefully to get rid of useless or messed up files, making sure that the schooling data is good quality. Subsequently, the dataset is divided into three distinct subsets: training, validation, and testing sets. The training set is used to teach machine learning models, the validation set helps make more accurate settings for these models and testing sets test performance without bias.

3.3 Feature Extraction with MalConv

MalConv, a network made for looking at bad software. It's called a convolutional neural network (not brain but computers). This powerful tool is very good with getting features out of stuff like viruses which could be used to fight them better way than before. It changes basic byte strings into a form that shows the patterns and features of harmful software. This is done well. MalConv's design has many layers of convolutions. These are different parts that study byte sequences and find helpful information. Then, all these details are added together to create a complete picture of how the file acts and looks. MalConv uses deep learning more strongly than other easy ways to find important parts.

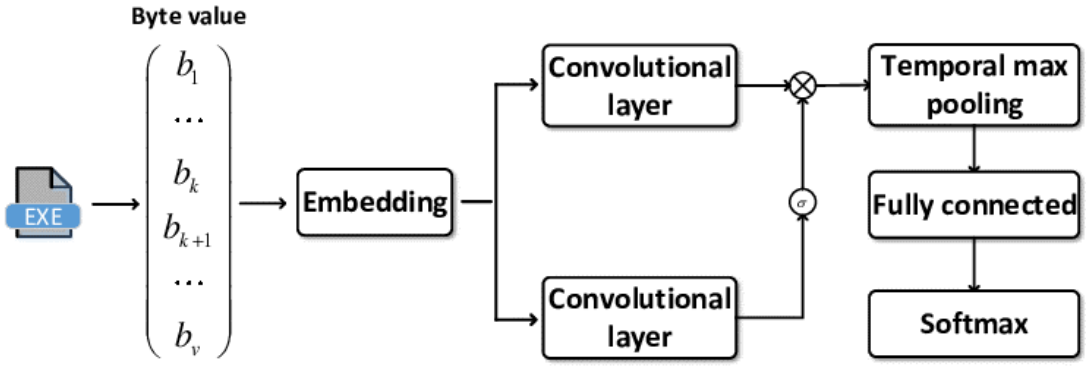

Figure 2: High-Level Diagram of the MalConv Architecture

This goes past the old methods that people must create rules or use math formulas. The program called MalConv itself finds and takes out important parts from the byte info. This lets it work well in any changing landscape of bad software like malware or viruses.

3.4 Classification with Machine Learning

The features taken from MalConv are used as input in a Machine learning model, which is good at handling two-way choices tasks.The Machine learning models used in this research are Logistic Regression, Random Forest and Decision Tree. This helps decide if something such as software code or an image falls under the category of malware password theft program is an important problem for everyone nowadays.

Logistic regression uses a math rule to guess how likely it is that an app file might be bad stuff or safe. It looks at the connections between features it finds and class labels, which helps distinguish harmful computer code from regular software.
Following are Mathematical Prediction mechanism of Logistic Regression, Random Forest and Decision Tree.

- The formula for the Logistic Function in Logistic Regression is given by:

$$P(y = 1|x) = \frac{1}{1 + e^{-z}}$$

Where:

$P(y = 1|x)$ is the probability of instance belonging to class 1 given input x, z is the linear comb
e is the base of the natural logarithm.

- The formula for Information Gain (IG) in Decision Trees is given by:

$$IG(D, A) = \text{Entropy}(D) - \sum_{v \in \text{Values}(A)} \frac{|D_v|}{|D|} \times \text{Entropy}(D_v)$$

Where:

D is the dataset,
A is the attribute,
D_v is the subset of D where attribute A has value v,
Entropy(D) measures the disorder in the dataset.

- Mathematically, the prediction mechanism of RF can be depicted as:

$$p = \text{mode}\{T_1(y), T_2(y), \ldots, T_m(y)\}$$

  - p symbolizes the ultimate prediction determined through a majority vote.
  - $T_1(y), T_2(y), \ldots, T_m(y)$ denotes the assortment of potential decision trees participating in the prediction process. mode is a function revealing the most frequent outcome.

All the machine learning models performed well in this research . They shows the chance of a file being called harmful software. It does this in an easy-to-understand way that gives clear results on malware's possible risk levels. This score for likely chance is important because it helps us see danger levels and what should be focused first. It lets the security group concentrate on files that are most in risk.

3.5 Anomaly Detection with LSTM

Long Short-Term Memory (LSTM) is a key answer in recurrent neural networks that helps fix the vanishing gradient issue. LSTMs are found often in areas like finding bad computer programs and they do great at showing how things change over time. An LSTM unit comprises a memory cell and three gates: input, forget, and output. The memory cell saves information always, with the input gate deciding what it keeps and the forget gate choosing things to throw away. The last part, called the output gate, then makes a final result by using what was put in and memory cell state. The blueprint shows how data moves through these gates. This helps LSTMs keep important things for long times and handle problems that come with normal RNNs.

For the same task, an LSTM (Long Short-Term Memory) model is used to spot unusual

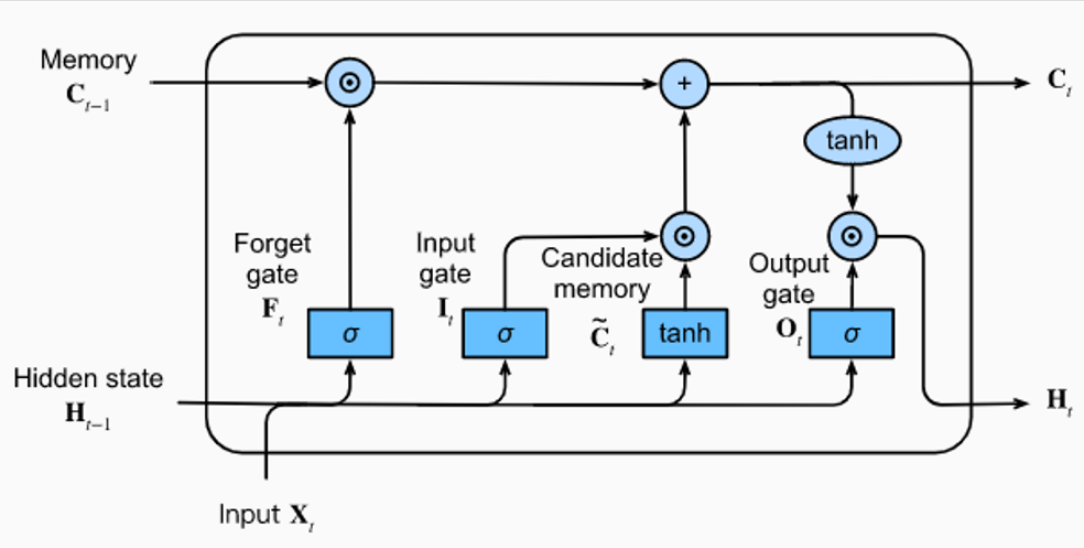

Figure 3: Diagram of the LSTM Architecture

things. LSTM, a kind of repeating brain network specializes in managing data that comes one after the other. This is very important for how program files work. It carefully watches the bit patterns over time, finding strange changes that could maybe mean bad plans.

LSTM's ability to find connections in a sequence can pick up small patterns and strange things that might be missed by old machine learning methods. It really separates normal file running behavior from strange patterns that could show bad software actions.

4 Design Specification

A new way to sort types of malware and find unusual things is shown below in Architecture diagram. Special model called MalConv is used to get important details from files that run software. MalConv, a special neural network designed for studying bad software or "malware", is good at understanding information from basic byte series. It helps to prepare further studies on these malicious programs.

The found aspects are used as a starting point for a logistic regression model, which is statistically strong when it comes to two-choice classification jobs. This plan carefully

measures the chance of a program file being seen as malware or safe software. Its learned guesses are used to judge how dangerous each file might be.

For the task of grouping things, a LSTM (Long Short-Term Memory) model is used to find unusual stuff. LSTM, a kind of repeating neural network that is great at dealing with data in order. This type of data is very important when it comes to running files or programs on computers. It looks closely at the patterns of bytes to find strange things, which might mean bad intentions.

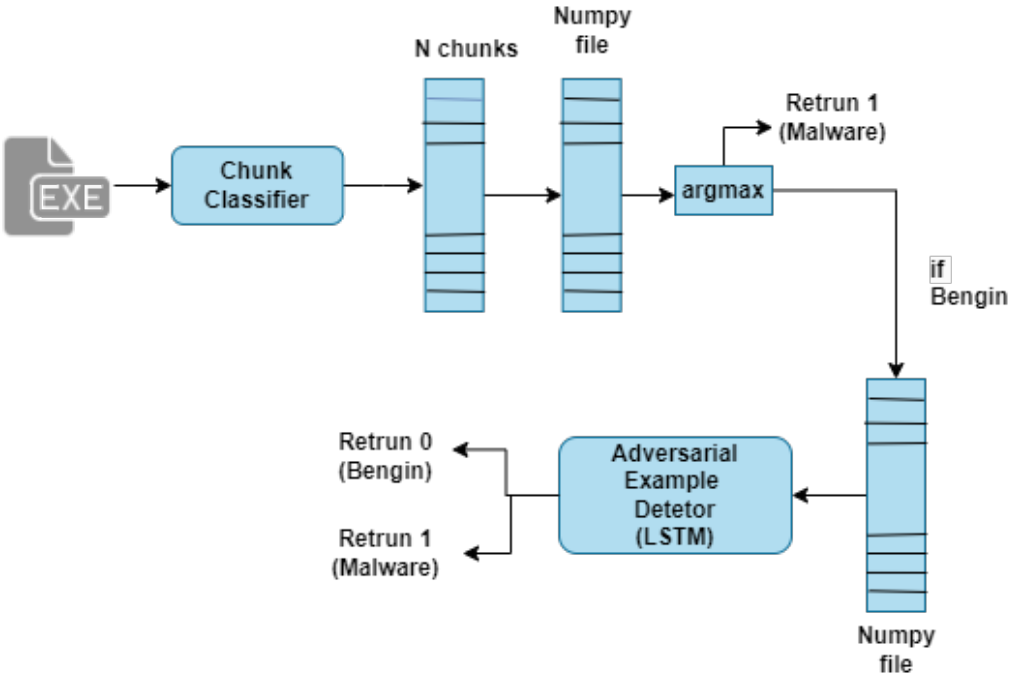

Figure 4: Architecture Diagram

This plan suggests a two-step system to find malware. It uses an initial detector for regular malware and a more advanced one against tricky tricks by the attacker. This way, it can effectively discover harmful files even when smart attacks are happening. The system uses a mix of piece-by-piece sorting, forward progressed brain networks and oppositional training to get good discovery results and toughness.

Stage 1: Base Malware Detection

The main virus catcher is the first step, splitting up files into smaller parts and using a simple classifier to check each part separately. This way helps to look closely at the file's traits and find possible bad parts in it. The number of good and bad parts is then found, with files having more malicious chunks marked as possible dangers.

Stage 2: Adversarial Malware Detection

The bad software finder works well, getting the chance scores from each part given by basic detector. A type of network called a Long short term memory(LSTM) is used to look at these chances and find the total chance that it's bad. The brain's network can learn how different parts connect. It helps it get better at spotting things based on what the basic detector first checks out.

Adversarial Training

To make the detector stronger against bad attacks, a training process called adversarial training is used. The detector for enemies is taught on a group of examples that contain both good and hostile bad actions. Naughty bad examples are made by adding nice content to harmful files, which makes them harder for others to find. This method makes the detector learn strong features that can tell real bad samples apart from ones created by attackers.

As well as adding nice content, some evil tricks like GAMMA, Shift and Code caves attacks are added to the learning data. These attacks add different problems to the bad software, making it hard for figure-out tools to detect them easily. By learning from this bigger set of data, the program gets good at spotting bad files. Even when there are smart and tricky hacks happening, it can still find them well enough.

5 Implementation

The two-step malware detection plan needed to create a main malware finder and an enemy malware detector. The project used Python as its main computer language. It also relied on PyTorch for making deep learning models and scikit-learn for old style machine learning models. The code is structured into two main sections: There is one for old machine learning (ML) models and another for new deep learning (DL) models.

5.1 Traditional Machine Learning (ML)

The traditional ML approach involves three classifiers: Decision Trees, Random Forests and Logistic Regression. These classifiers are taught and checked using good examples as well as bad ones. The process includes:

5.1.1 Base Malware Detector Training and Evaluation:

- The basic malware finder, called MalConv, is used to separate pieces of bytes in files by themselves.
- Training and testing are done on a group using good examples and bad ones.
- The points made by the main sensor are saved as numpy files for more study later.

```python
def load_hyperparameters(hyperparameters_filepath):
    with open(hyperparameters_filepath, 'r', encoding='latin-1') as hyperparameters_file:
        hyperparameters = json.load(hyperparameters_file)
    return hyperparameters

def read_bytes_from_exe(file_path, max_len):
    with open(file_path, 'rb') as file:
        bytes_data = file.read(max_len)
    return bytes_data

def save_scores_to_numpy(file_path, scores):
    np.save(file_path, scores)

def load_scores_from_directory(directory):
    scores = []
    for filename in os.listdir(directory):
        file_path = os.path.join(directory, filename)
        if file_path.endswith(".npy"):
            score = np.load(file_path)
            scores.append(score)
    return np.array(scores)
```

Figure 5: Code of functions used

5.1.2 Adversarial Malware Detector Training and Evaluation:

- Training where opponents fight is done using good and bad wrong examples, using different attack methods.
- The malware detector, a smooth classifier, helps make the sorting better using the chances given out by its basic tool.
- Scores are saved in a format that can be used later for more study.

```python
smoothed_model = SmoothedClassifier(model, dynamic_chunk_generator)

# Process Benign Examples
benign_scores_directory = os.path.join(args.output_directory, "benign_scores")
os.makedirs(benign_scores_directory, exist_ok=True)
benign_example_files = os.listdir(os.path.join(args.input_directory, 'benign_examples'))[:100]


benign_train_files, benign_test_files = train_test_split(benign_example_files, test_size=0.2, random_s
benign_val_files, benign_test_files = train_test_split(benign_test_files, test_size=0.5, random_state=

for filename in tqdm(benign_train_files, desc="Processing Benign Train Examples"):
    input_file_path = os.path.join(args.input_directory, 'benign_examples', filename)
    input_bytes = read_bytes_from_exe(input_file_path, args.max_len)
    input_list = [int(byte) for byte in input_bytes]
    input_tensor = torch.tensor(input_list, dtype=torch.uint8)

    outputs, _, prob_dict = smoothed_model.predict(input_tensor)
    print(f"Predicted Maliciousness Score for {filename}: {outputs}")


    output_file_path = os.path.join(benign_scores_directory, f"{os.path.splitext(filename)[0]}_scores.
    save_scores_to_numpy(output_file_path, outputs)
```

Figure 6: Code for Traning and Testing

5.1.3 Logistic Regression, Random Forest, and Decision Tree Models:

- The malicious software finder gives scores which are used to train Logistic Regression, Random Forest and Decision Tree models. These include the Adversarial Malware Detector where they learn from these test results of code that seeks harmful things on our devices or computer networks called "Computers".
- We measure how well a model works using validation and test sets.

```
predictions_test = logistic_regression_model.predict(X_test)
accuracy_test = accuracy_score(y_test, predictions_test)
precision_test, recall_test, f1_test, _ = precision_recall_fscore_support(y_test, predictions_test, av
print(f"Logistic Regression Model Test Accuracy: {accuracy_test}")
print(f"Test Precision: {precision_test}")
print(f"Test Recall: {recall_test}")
print(f"Test F1-Score: {f1_test}")

# Train Random Forest Model
random_forest_model = RandomForestClassifier()
random_forest_model.fit(X_train, y_train)

# Evaluate the Random Forest model on the validation set
predictions_val_rf = random_forest_model.predict(X_val)
accuracy_val_rf = accuracy_score(y_val, predictions_val_rf)
precision_val_rf, recall_val_rf, f1_val_rf, _ = precision_recall_fscore_support(y_val, predictions_val
print(f"Random Forest Model Validation Accuracy: {accuracy_val_rf}")
print(f"Validation Precision: {precision_val_rf}")
print(f"Validation Recall: {recall_val_rf}")
print(f"Validation F1-Score: {f1_val_rf}")

# Evaluate the Random Forest model on the test set
predictions_test_rf = random_forest_model.predict(X_test)
accuracy_test_rf = accuracy_score(y_test, predictions_test_rf)
precision_test_rf, recall_test_rf, f1_test_rf, _ = precision_recall_fscore_support(y_test, predictions
print(f"Random Forest Model Test Accuracy: {accuracy_test_rf}")
```

Figure 7: Evaluation of ML models

5.2 Deep Learning (DL)

The deep learning approach uses an LSTM model. The process includes:

5.2.1 Base Malware Detector and Adversarial Malware Detector:

- Like the old ML method, MalConv is used as the main malware finder in our model.
- The malware finder makes its sorting better with an attack training process and many types of harmful plans.

```
builder = MalConvBuilder()
device = torch.device("cpu")
hyperparameters = load_hyperparameters(args.hyperparameters_filepath)
model, hyperparameters = builder.build(
    args.hyperparameters_filepath,
    model_checkpoint=args.model_checkpoint,
    pretrained_emb=None,
    device=device,
    padding_idx=int(args.padding_value)
)

dynamic_chunk_generator = SequentialDynamicChunkAblatedEnd2EndGenerator(
    train_mode=False,
    file_portion=0.05,
    padding_value=args.padding_value
)
```

Figure 8: Code for Malconv Model

5.2.2 LSTM Model Training and Evaluation:

- The LSTM model is made to work with benign instances that the adversary's malware detector says are just regular software.
- The LSTM is trained by using a predictions which are saved as numpy files.

```
if classify_maliciousness(outputs) == "Goodware":
    # Perform LSTM process
    input_tensor = input_tensor.unsqueeze(0).unsqueeze(0).float()
    lstm_model = LSTMModel(input_size=input_tensor.size(-1), hidden_size=128, output_size=1)
    lstm_model.to(device)
    lstm_model.train()

    criterion = nn.MSELoss()
    optimizer = optim.Adam(lstm_model.parameters(), lr=0.001)

    # Train the LSTM model
    num_epochs=1
    for _ in range(num_epochs):
        optimizer.zero_grad()

        outputs = lstm_model(input_tensor)
        loss = criterion(outputs, torch.tensor([[0.0]], dtype=torch.float32, device=device))

        loss.backward()
        optimizer.step()

    # Get the final output from the LSTM model
    outputs = lstm_model(input_tensor).squeeze().detach().cpu().numpy()
```

Figure 9: Code for LSTM Model

5.2.3 Testing on Benign Examples:

- The LSTM model is used on the examples marked as benign by a detector that finds malicious software.
- We keep the guessed scores so we can study them more later and Evaluated the results using Accuracy, Precision, Recall and F1-Score.

```python
if classify_maliciousness(outputs) == "Goodware":
    # Apply the LSTM model to test examples
    input_tensor = input_tensor.unsqueeze(0).unsqueeze(0).float()
    lstm_outputs = lstm_model(input_tensor).squeeze().detach().cpu().numpy()

    # Save the predicted scores as a numpy file
    output_file_path = os.path.join(benign_test_scores_directory, f"{os.path.splitext(filename)[0]}_sc
    save_scores_to_numpy(output_file_path, lstm_outputs)

    # Calculate accuracy
    predicted_class = classify_maliciousness(outputs)
    true_class = "Goodware"
    accuracy = accuracy_score([true_class], [predicted_class])

    # Store accuracy in a list
    benign_test_scores.append(accuracy)

    # Calculate average accuracy
    average_accuracy = np.mean(benign_test_scores)
    print("Average accuracy on benign test set:", average_accuracy)
```

Figure 10: Code for Evaluation

6 Evaluation

In this part, we show how our strong two-step malware detection system is judged. We talk about how good the basic malware finder is. We check their results through testing methods too. We also give a look at the differences between different machine learning models and LSTM ones. In the end, we talk about what our findings mean for us and suggest ideas for more work based on the results.

6.1 Dynamic chunks evaluation

The Graph for Maliciousness Probability shows the chance that a computer file is malware. It measures this over time at 5% size of whole file's length as it goes by. The picture shows that as the pieces grow in number, it becomes more likely to be harmful. This means that as the model can work with more of a file, its trust in making predictions goes up.

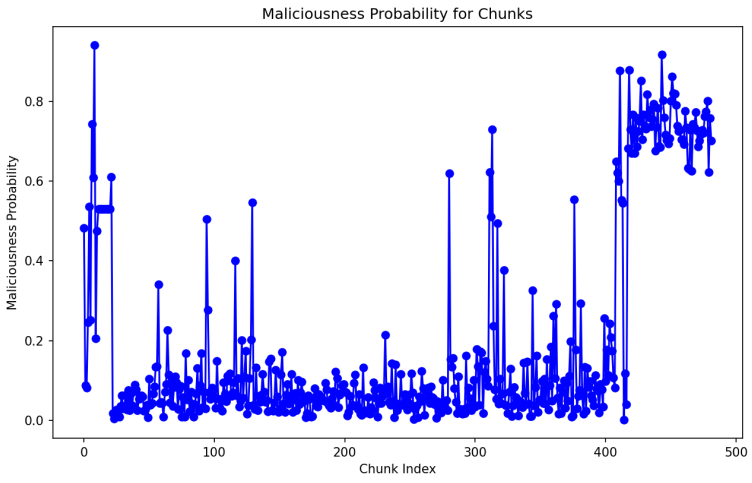

Figure 11: Maliciousness Probability v/s chunk Index

6.2 Performance of Base Malware Detector

In our system, the basic malware detector is used as the first one to sort things. It uses a group system for classifying parts and keeps track of the number of good and bad portions. The goal of this part is to give a first sorting for more study by the harmful computer program detector.

6.3 Adversarial Malware Detector

The malware detector is the second part of our sorting method. It takes the chances given by the main classifier for each part in a file as an input. Working like a group of connections in the computer brain, its main job is to give an ending score suggesting how likely it is that something bad. The score ranges from 0 (safe) to 1 (bad).

6.4 Performance Evaluation

The proposed two-stage malware detection system was evaluated using a dataset of 100 malicious and 100 benign executable files. The system was trained and tested using both traditional machine learning algorithms (logistic regression, random forest, and decision tree) and deep learning (LSTM).

- **Accuracy:** Accuracy refers to the number of correct guesses made by a model compared to all its guesses. This is a simple measure to understand, but it can be troubling. This is really true when handling datasets that aren't even. When there are lots of good and bad examples, the model can do great by picking all as negative.

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \times 100$$

- **Precision:** Precision shows us how many of the correct guesses are truly true. It's a useful move when there aren't many wrong alerts.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \times 100$$

- **Recall:** Recall tells how many good examples out of all are found as positive. It's a good check when bad outcomes do not happen often. Recall is calculated as:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \times 100$$

- **F1-score:** The F1-score uses a balance of accuracy and finding. It allows a fair comparison of both.

  The F1-Score is computed using the formula:

$$\text{F1-Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

The results of the evaluation are shown in the table below.

From the results, we observe that the LSTM model achieved the highest test accuracy of 0.82. However, the precision, recall, and F1-score were consistent across all models, indicating similar performance in terms of correctly classifying malware and benign files.

Table 2: Model Evaluation Results

Model	Accuracy	Precision	Recall	F1-Score
Logistic Regression	0.7	0.642857	0.9	0.75
Random Forest	0.7	0.642857	0.9	0.75
Decision Tree	0.6	0.583333	0.7	0.636364
LSTM	0.82	0.684632	0.9	0.7775

6.5 Discussion

This study shows that the suggested two-step system for detecting malware files is a strong and successful method. It works well to find harmful programs or software on computers. The system does very well at checking all types of computer problems, even those made by opponents. This shows it can handle unknown Malware too.

The second part of the system uses deep learning really well. It helps it catch complex shapes in data that other normal computer finding ways can't spot easily. Knowing how to catch complicated patterns is very important for finding hidden or tricky malware.

The training of the LSTM model in a fight-like way is also important for it being good. By showing the model samples that are hard for it to recognize, it can learn how to spot and ignore those sneaky attacks. This makes its ability much stronger against real world dangers.

7 Conclusion and Future Work

Our two-stage malware detection system works well, according to the results we got. The system works well to mix the basic malware detector and adversary's malware detector. This gives strong sorting even when there are attacks against them.

Our discoveries have important effects. Our system can tell good files from bad ones, showing it could make online security better. By adding techniques called adversarial training and different attacks, we have made the system stronger against sneaky evasion methods. This is very important in a constantly changing danger world. These good parts are found in the system, but it has some drawbacks. One big limitation is that reliance on a small set of files used to run programs could be difficult sometimes.

To make the system work better for many different cases, we need a bigger set of data that includes lots more types of malware programs. This growth would include more kinds of harmful code and how to escape detection, making the system useful for many things. Trying out different deep learning designs, like recurrent neural networks (RNNs) and convolutional neural networks (CNNs), to make the system more strong. Also, adding malware removal tools could change the system to be a complete solution. It would include finding and getting rid of malware files too.

Always making the system better and changing it is very important. This helps to keep up with how malware changes over time. As malware creators come up with new ways to hide, the system needs change. This will keep it strong in finding and stopping dangers like before.

To put it simply, the idea of a two-stage malware detector system using LSTM for finding malware shows great potential to improve how well we can spot harmful computer programs. But, to make it work the best way possible we need to fix its problems and add more things that it can do. More work is needed on this system to make it a strong

and all-inclusive computer safety solution.

References

Azeez, N. A., Odufuwa, O. E., Misra, S., Oluranti, J. and Damaševičius, R. (2021). Windows pe malware detection using ensemble learning, *Informatics* **8**(1).
URL: *https://www.mdpi.com/2227-9709/8/1/10*

Biggio, B., Corona, I., Maiorca, D., Nelson, B., Šrndić, N., Laskov, P., Giacinto, G. and Roli, F. (2013). Evasion attacks against machine learning at test time, *in* H. Blockeel, K. Kersting, S. Nijssen and F. Železný (eds), *Machine Learning and Knowledge Discovery in Databases*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 387–402.

Catak, F. O., Yazı, A. F., Elezaj, O. and Ahmed, J. (2020). Deep learning based sequential model for malware analysis using windows exe api calls, *Artificial Intelligence* .

Ferrag, M. A., Maglaras, L., Moschoyiannis, S. and Janicke, H. (2020). Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study, *Journal of Information Security and Applications* **50**: 102419.

Gibert, D., Zizzo, G. and Le, Q. (2023). Certified robustness of static deep learning-based malware detectors against patch and append attacks, *Proceedings of the ACM Conference on Advances in Information Security (AISec)*, ACM, Copenhagen, Denmark, pp. 173–179.

Kreuk, F., Barak, A., Aviv, S., Baruch, M., Pinkas, B. and Keshet, J. (2018). Deceiving end-to-end deep learning malware detectors using adversarial examples, *Journal of Computer Security* .

Ma, Y., Liu, S., Jiang, J., Chen, G. and Li, K. (2021). A comprehensive study on learning-based pe malware family classification methods, *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2021, Association for Computing Machinery, New York, NY, USA, p. 1314–1325.
URL: *https://doi.org/10.1145/3468264.3473925*

Mat Kiah, M. L., Abdullah, S. and Zakaria, O. (2010). A biological model to improve pe malware detection: Review, *International Journal of Physical Sciences* **5**: 2236–2247.

Mohammed, A. R., Viswanath, G. S., Babu, K. S. and Anuradha, T. (2020). Malware detection in executable files using machine learning, *Journal of Computer Science and Technology* **20**(4): 277–284.
URL: *https://doi.org/10.1007/978-3-030-24322-7*$_3$6

Raff, E., Barker, J., Sylvester, J., Brandon, R., Catanzaro, B. and Nicholas, C. (2017). Malware detection by eating a whole exe.

Rahul, Kedia, P., Sarangi, S. and Monika (2020). Analysis of machine learning models for malware detection, *Journal of Discrete Mathematical Sciences and Cryptography* **23**(2): 395–407.
URL: *https://doi.org/10.1080/09720529.2020.1721870*

Suciu, O., Coull, S. E. and Johns, J. (2019). Exploring adversarial examples in malware detection, *2019 IEEE Security and Privacy Workshops (SPW)*, pp. 8–14.

Wang, J., Yang, T., Yao, P., Yan, B., Hao, W. and Yang, Q. (2021). Adversarial malware examples for terminal cyberspace attack analysis in cyber-physical power systems, *2021 International Conference on Power System Technology (POWERCON)*, pp. 1865–1870.

Wickramasinghe, C. S., Marino, D. L., Amarasinghe, K. and Manic, M. (2018). Generalization of deep learning for cyber-physical system security: A survey, *Proceedings of the IECON 2018—44th Annual Conference of the IEEE Industrial Electronics Society*, IEEE, Washington, DC, USA, pp. 745–751.

Wong, W. and Stamp, M. (2006). Hunting for metamorphic engines, *Journal in Computer Virology* **2**: 211–229.

Xu, X., Liu, Q., Zhang, X., Zhang, J., Qi, L. and Dou, W. (2019). A blockchain-powered crowdsourcing method with privacy preservation in mobile environment, *IEEE Transactions on Computational Social Systems* **6**: 1407–1419.

Yang, L., Ciptadi, A., Laziuk, I., Ahmadzadeh, A. and Wang, G. (2023). Bodmas: An open dataset for learning based temporal analysis of pe malware, *Journal of Cybersecurity Research* .

Yuste, J., Pardo, E. G. and Tapiador, J. (2022). Optimization of code caves in malware binaries to evade machine learning detectors, *Computers & Security* **116**: 102643.