THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF SOFTWARE ENGINEERING

ASSESSING THE VALIDITY OF USING AI TO ENCRYPT EVASIVE MALWARE

DAVID LENGEL
SPRING 2020

A thesis
submitted in partial fulfillment
of the requirements
for baccalaureate degree
in Software Engineering
with honors in Software Engineering

Reviewed and approved* by the following:

Zhifeng Xiao
Associate Professor of Computer Science and Software Engineering
Thesis Supervisor

Wen-Li Wang
Associate Professor of Computer Science and Software Engineering
Honors Adviser

*Signatures are on file in the Schreyer Honors College.

Abstract

The purpose of this research is to assess the validity of using artificial intelligence to encrypt and decrypt evasive malware capable of performing targeted attacks. This is done through the creation and analysis of a prototype system of machine learning models. It has been found that different machine learning models yield different results, but there are patterns and trends in the abilities of all models. In the end, the system is successful in being able to use artificial intelligence to encrypt and decrypt malware capable of accurately, but lacks in being a fully evasive malware because of its inability to effectively hide trigger conditions. Although it has a drawback, there are some considerations for future improvement to maintain the parts of the system that have been proven to work, while effectively improving its shortcomings.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I wish to express my sincere appreciation to my supervisor, Dr. Zhifeng Xiao. He is a subject matter expert in cyber security and applied cryptography at Penn State Behrend, and without his knowledge and support throughout this entire process, this research would not have been possible.

Chapter 1

Introduction

This section will cover some background information, context as well as some some key definitions, the purpose of this research, the significance and scope, and finally a general outline of the sections to come in the rest of the report.

1.1 Background

Ever since the invention of computers, people have sought to exploit their vast potential. Many of these feats have been beneficial to the populace, from Alan Turing's use of very early computers to crack the Enigma code in the 1940s to Andy Gavin's utilization of the PlayStation 2 to create a 3D platformer in the 1990s. One helped to win one of the largest wars in history while the other improved the success of a video game that is still being played decades later, but both Turing and Gavin used their ingenuity and knowledge of computers to break past boundaries and achieve something that had not been done before.

While there are many examples of people exploiting computers for the benefit of others, there are also those who wish to exploit computers for malicious purposes. From smaller viruses intended to steal information or extract a payment from individual computer users to large-scale viruses like Stuxnet which caused damage to nuclear reactors in Iran in the early 21st century, many people exploit vulnerabilites in computers with the intent to harm others.

As computers have begun to become more powerful over the decades, those people with malicious intent are now developing malware by not only just exploiting vulnerabilities of their target's computers, but also by exploiting the power of their own computers. With recent advancements in the availability and capability of artificial intelligence (AI), those who create malware may soon seek to take advantage of this opportunity and use AI models and techniques to make their malware more dangerous.

1.2 Context

This paper will focus on the use of AI in the *cryptography* of *malware*, specifically for creating *evasive malware* capable of performing *targeted attacks*. Some of these terms may be new to some readers so they'll be briefly covered below, starting with "malware".

Malware

Malware stands for *mal*icious soft*ware*. As can be inferred from its name, it is software that harmfully attacks other software. This means that it causes the actual behavior of the target software to behave differently from its intended behavior [1]. This can be done in a few different ways, the three most common of which are viruses, worms and Trojan horses [2].

A virus is a "self-replicating program" that looks for other programs to infect and destroys. A worm is program that travels on devices and networks which seeks to exploit one or more software vulnerabilites. Finally, a Trojan horse "masquerades as some other legitimate program that the device or user is tricked into executing". These are three traditional forms of malware. Today, malware is often a combination of of two or more of these types [2].

Now that malware has been defined for the purposes of this paper, it is important to know how that malware is hidden and activated.

Cryptography

Cryptography is the art of writing and solving codes, and is made up of two converse processes: *encryption* and *decryption*.

Encryption is the process of converting ordinary information into unintelligible information, while decryption is the process of converting that unintelligible information back into ordinary information. This has traditionally been done through the use of cipher, or an algorithm and key. The sender will use a key and an algroithm to encrypt a message and the receiver will use that same key and algorithm to decrypt the message. The idea is that both the receiver's algorithm and they key must match those of the sender in order for the data to be correctly deciphered [3].

The basics of malware and it's security have been covered, so now the details of the specific type of malware in this research can be covered.

Evasive Malware

Evasive malware is a subset of malware which is aware of its execution environment context and uses this to attempt to avoid detection from security solutions. The goal of this type of malware is to keep the presence and location of the malware hidden in order to avoid its discovery and dissection [4].

Next, the type of attack that that will be performed by the evasive malware for the purposes of this research will be covered.

Targeted Attacks

Targeted attacks are cyber-attacks which focus on a chosen individual or group. As opposed to viruses and worms which often attack broadly and indiscriminantly, targeted attacks involve careful information gathering and expensive resource spending to ensure that the desired individual or group is targeted [5].

Now that the basic terms for understanding this research have been defined, the first statement of this section can be reviewed in more detail.

To review, the context of this research can be summarized in the following phrase: "the use of AI in the *cryptography* of *malware*, specifically for creating *evasive malware* capable of performing *targeted attacks*."

Breaking the phrase down into its components based on the definitions provided above, the context of this research can be re-phrased as follows: "the use of AI in the *encryption and decryption* of *malicious software*, specifically for creating *such malicious software that attempts to avoid detection* capable of performing *cyber-attacks which focus on a chosen individual or group*.

Although much more of a mouthful, this broken-down phrase gives a better idea behind the context of this research for those not familiar with all of the concepts.

1.3 Purpose

The purpose of this research is to assess the validity of using AI to encrypt and decrypt evasive malware capable of performing targeted attacks through the creation and analysis of a prototype system of machine learning (ML) models.

1.4 Significance and Scope

On the surface, this research deals with assessing a potentially up-and-coming threat in cybersecurity. However, it also has a deeper meaning in the uniting of the two historically separate fields of *cryptography* and *machine learning*.

Cryptography typically deals with deterministic algorithms, where the output of a model is fully determined by the parameters and conditions that are provided as input. On the other hand, machine learning is typically stochastic, where the output of models is determined by some probability and is subject to some randomness. By creating a prototype that applies machine learning to the field of cryptography, this research seeks to bridge the gap between these two varying subjects.

1.5 Thesis Outline

The next section will be a Literature Review. This will look into research done by other professionals in certain fields of study that align with those used for this research.

Following that will be the Methodology section, where details about the procedures and techniques related to the research will be discussed.

Next will be the System Design, where technical details about the process and design of the prototype will be given.

After that will be the Evaluation, where various results of the prototype will be provided and assessed.

Finally, the Conclusions section will wrap up the report.

As mentioned above, the next section is the Literature Review, where some areas of related work will be covered in more detail.

Chapter 2

Literature Review

This section will provide an overview of DeepLocker, which was the inspiration for this research.

2.1 DeepLocker

As described by Marc Ph. Stoecklin, DeepLocker is a ”new breed of highly targeted and evasive attack tools powered by AI”[6]. Stoecklin is a member of the IBM Reserach team, the creators of DeepLocker. Stoecklin and his team, including Jiyong Jang and Dhilung Kirat, have been looking into the ways that AI can be used by cybercriminals, and have been developing DeepLocker to ”better understand how several existing AI models an be combined with current malware techniques to create a particularly challenging new breed of malware”[6] which is evasive and able to perform powerful targeted attacks.

DeepLocker works by concealing the malware payload in ”benign carrier applications”. Once in the target environment, the AI makes use of special ”trigger conditions” in order to unlock and activate the malicious attack. The idea is that payload will only be unlocked when the trigger conditions are met by intended target. This is done using a deep neural network(DNN) to produce the correct key to unlock the payload only when the conditions are met; otherwise, it behaves in a benign manner. DeepLocker takes advantage of the ”black box” nature of DNNs and their ability to accept a very large number of inputs to prevent the trigger conditions and payload from being reverse-engineered. In order to achieve a large number of inputs, DeepLocker uses visual, audio, geolocation and system features as inputs to the DNN AI model [6].

Stoecklin, Jiyong and Kirat demonstrated a proof of concept for DeepLocker at the Black Hat USA 2018 conference which involved hiding ransomware in a video conferencing application. They trained the AI model to unlock and execute the ransomware when the face of a specific person was recognized. They showed that when other faces were detected on computer’s camera, the video conferencing application worked as one would expect a video conferencing application to work and the ransomware acted benignly. However, when the person whose face the AI model was trained to recognize stood in front of the camera, the ransomware unlocked and began to execute [7].

Although IBM has yet to release more specific details of how DeepLocker works under the surface, the research and the proof of concept show that using AI to hide and activate powerful malware is possible. Furthermore, it may be possible to apply this technique of creating evasive malware to other areas of cyber security.

2.2 Summary

The main topic discussed in this section was DeepLocker and how it was the inspiration for this research. In the next chapter, the methodology will be discussed relating to how the ideas presented by DeepLocker can be utilized in this research.

Chapter 3

Methodology

This section will cover information on the general methodology behind the research. This includes the approach to creating and evaluating the machine learning models, the methods of data collection, the analysis of computed data and an overview of the tools used.

3.1 Approach

Recall that in the introduction section, the purpose of this research is ”to assess the validity of using AI to encrypt and decrypt evasive malware capable of performing targeted attacks through the creation and analysis of a prototype system of machine learning models.” In order to meet this purpose, the methodological approach to this research will have two stages: (1) the creation of the ML models for encrypting and decrypting malware and (2) the analysis of those model.

Creation of the Machine Learning Models

The first stage of the approach is the creation of the ML model itself. The purpose of the prototype ML models is to accept as inputs information about the computer environment from the computer that it is currently on, and output a single key. The goal is that the system will output the same key with which the malware was encrypted when on a desired target computer. Otherwise, it will generate a non-matching key. When a matching key is generated, the malware may then be decrypted and activated on the target computer.

The ML models make the malware both evasive and capable of performing targeted attacks. It makes the malware evasive by both ensuring that the malware remains dormant while not active and making it difficult to reverse-engineer. Similarly, it makes the malware capable of performing targeted attacks by outputting the desired decryption key only when in a target environment.

This will be accomplished by making one large system composed of several smaller ML models. Each smaller model will accept all environmental properties as inputs and output a single binary bit. The binary bits output by all of the smaller models will be combined into a key, which will be the output of the larger model. By having all ML models accept the same inputs, yet having the outputs only be affected by certain varying environmental properties, the properties that are influencing the outputs will be hidden. The details of such a design will be discussed more in Chapter 4: System Design, and a graphic of this design can be seen in Figure 4.2.

The particular types of ML models that will be used in this research are Decision Trees and Neural Networks because they are both able to perform supervised learning and have widely different structures. This will allow for the assessment of the effects of different types of models on the creation of the key.

Different models will then be trained to output certain binary bits when conditions are met based on the enviornmental inputs. For example, a model can be trained to output a value of ’1’ when the environment has less than 2 antivirus products installed and not a virtual machine and it is running on Windows 7 OS, otherwise it will output a value of ’0’. As an added layer of security, a pre-defined series of 0s and 1s will be created that will define which models will evaluate to ’1’ when conditions are met and which models will evaluate to ’0’ when conditions are met. This will lead to a final key with individual binary bit values from which significance will be harder interpret.

After the creation of the ML model, the results will then have to be evaluated.

Evaluation of the Machine Learning Models

The second of the two stages is the evaluation of the ML models. There are two main points to be considered in assessing the validity of such a system.

The first main point is in the system's ability to generate the correct key for decryption of the malware. To be considered a valid approach, there must be a pattern in each of the machine learning models' abilities to generate the desired bits of the key. If there is no such trend, the malware may not be activated reliably.

The second main point is in the difficulty of the ML models to be reverse-engineered. Machine learning models vary in their ability to be reverse-engineered, and there are several features that are specific to this research that must be considered in order to asses its ability to be reverse-engineered. Both of the factors must be taken into consideration, and more details about the specific considerations are provided in the Evaluation chapter.

Please note that in this section was the general methodological approach to the evaluation of the results. For more detailed information about analysis, refer to the Analysis section later in this chapter.

3.2 Data Collection

In order to properly train the ML models, varying enironmental data from a large number of computers was needed. Instead of manually collecting this data, a pre-existing online dataset was used.

The "Microsoft Malware Prediction" dataset was created by Microsoft and is hosted on Kaggle (https://www.kaggle.com/c/microsoft-malware-prediction/data). The dataset was originally created for use in a comptition to "predict a Windows machine's probability of getting infected by various families of malware, based on different properties of that machine"[8]. To compliment their competition, they created a dataset for teams to use to train and test their submissions which contains 8,921,483 unique training samples and 7,853,253 unique test samples of Windows machines, with each sample containing data on 83 environement properties. While there were 83 environement property fields provided, many of them were not desired to be used and were removed for this research.

There are a few reasons why fields were removed. First, some fields contained unique data for each sample. This information would not be useful for training a ML model because the model would not be able to establish a pattern based on this data. Second, some fields contained data that, in order to meet business constraints, Microsoft left the meaning of the data used undefined. This information was not used because it was not clear if the representation of the data by Microsoft reflected the true nature of the data. Third, fields were removed that contained data which was no longer maintained or updated. Finally, fields were removed if they did not contain data derived directly from the environment. For example, the field Census_DeviceFamily from the original dataset indicated "the type of device that an edition of the OS is intended for" [8], which is ultimately derived from the OS edition, another field in the dataset.

After pre-processing was complete, these were the remaining environmental properties used for training and testing the model with definitions taken from Kaggle:

1. ProductName - Defender state information

2. EngineVersion - Defender state information
3. AppVersion - Defender state information
4. AvSigVersion - Defender state information
5. IsBeta - Defender state information
6. AVProductsInstalled - Number of antivirus products installed
7. AVProductsEnabled - Number of antivirus products enabled
8. HasTpm - Trusted Platform Module state information
9. CountryIdentifier - ID for the country the machine is located in
10. CityIdentifier - ID for the city the machine is located in
11. OrganizationIdentifier - ID for the organization the machine belongs in, organization ID is mapped to both specific companies and broad industries
12. GeoNameIdentifier - ID for the geographic region a machine is located in
13. LocalEnglishNameIdentifier - English name of Locale ID of the current user
14. Platform - Calculates platform name (of OS related properties and processor property)
15. Processor - This is the process architecture of the installed operating system
16. OsVer - Version of the current operating system
17. OsBuild - Build of the current operating system
18. OsSuite - Product suite mask for the current operating system
19. OsPlatformSubRelease - Operating system platform sub-release
20. SkuEdition - The Product Type defined in the MSDN
21. IsProtected - True if there is at least one active and up-to-date antivirus product running on this machine
22. AutoSampleOptIn - This is the SubmitSamplesConsent value passed in from the service, available on CAMP 9+
23. PuaMode - Pua Enabled mode from the service
24. SMode - True when the device is known to be in 'S Mode', as in, Windows 10 S mode, where only Microsoft Store apps can be installed
25. Firewall - True if Windows firewall is enabled

26. UacLuaenable - Whether or not the ”administrator in Admin Approval Mode” user type is disabled or enabled in UAC
27. Census_ProcessorCoreCount - Number of logical cores in the processor
28. Census_PrimaryDiskTotalCapacity - Amount of disk space on primary disk of the machine in MB
29. Census_PrimaryDiskTypeName - Friendly name of Primary Disk Type - HDD or SSD
30. Census_SystemVolumeTotalCapacity - The size of the partition that the System volume is installed on in MB
31. Census_HasOpticalDiskDrive - True indicates that the machine has an optical disk drive (CD/DVD)
32. Census_TotalPhysicalRAM - Retrieves the physical RAM in MB
33. Census_OSInstallTypeName - Friendly description of what install was used on the machine
34. Census_OSWUAutoUpdateOptionsName - Friendly name of the WindowsUpdate auto-update settings on the machine
35. Census_IsPortableOperatingSystem - Indicates whether OS is booted up and running via Windows-To-Go on a USB stick
36. Census_GenuineStateName - Friendly name of OSGenuineStateID
37. Census_IsFlightsDisabled - Indicates if the machine is participating in flighting
38. Census_FlightRing - The ring that the device user would like to receive flights for
39. Census_IsSecureBootEnabled - Indicates if Secure Boot mode is enabled.
40. Census_IsVirtualDevice - Identifies a Virtual Machine
41. Census_IsTouchEnabled - Is this a touch device ?
42. Census_IsPenCapable - Is the device capable of pen input ?
43. Census_IsAlwaysOnAlwaysConnectedCapable - Whether the battery enables the device to be AlwaysOnAlwaysConnected
44. Wdft_IsGamer - Indicates whether the device is a gamer device or not

The final step in the pre-processing of data was the *one-hot encoding* of the categorical data. Categorical data is data that contains labels. Many machine learning models are unable to use these labels, and instead require numeric values, which the process of one-hot encoding resolves. One-hot encoding transforms labels to numeric values by replacing a single field of labels with a number of new numeric binary fields, one for each unique label. For each sample, the new binary field corresponding with the label of that sample is given a value of 1, while the other binary fields

are given a 0. This encoding technique actually expands the number of environmental proprties from 44 to 91. The effects of this encoding technique on the system will be discussed in further detail in the next chapter.

3.3 Analysis

The data collected from each ML model was analyzed to extract certain performance measures, including a confusion matrix, accuracy and f1-score.

The confusion matrix was used to determine the number of true positives, false postives, true negatives and false negatives among the cases in order to derive the rest of the measures. The accuracy was used simply to measure the correctly identified keys. In order to contrast the accuracy, the f1-score was derived from the precision and recall to get a measure of the incorrectly classified cases. Refer to Chapter 5 for details of the actual data gathered.

The performance measures were gathered from multiple tests on different models and used to compare their effectiveness, as well as identify any trends in the capabilities (or lack thereof) of the models. In particular, the models were analyzed by comparing the number of inputs passed to the model, the number of outputs from the model, and the overall type of the model. The details of these specifications will be covered in System Design chapter, and the evaluation of these results will be covered in the Evaluation chapter.

3.4 Tools

The Python programming language was used for this research, and certain Python libraries were useful for the necessary data processing, development and analysis involved.

For the data processing, the Python library Pandas was primarily used. Pandas is an ”open source data analysis and manipulation tool” build on top of Python [9]. It was used specifically for reading and writing to and from comma-separated values (CSV) files and manipulating dataframes. The CSV files were used to store the initial dataset, the refined dataset and the labels to be used by the models. Dataframes were used to efficiently store and manipulate large amounts of data.

A few different Python libraries were used for development of the ML models, particularly skearn.neural_network for creating neural networks and sklearn_tree for creating decision trees. Both of these libraries belong to ”scikit-learn”. Scikit-learn is a set of tools for predictive data analysis in machine learning for Python [10], and was used for both the creation and prediction of the models used in this research.

The Python library used for analysis of the results was sklearn.metrics. This is the same ”scikit-learn” product that was used for the development of the ML models, but the metrics library is used specifically for analyzing the results of the models. In this case, it was used for constructing the confusion matrices of the models, which were then used to obtain other performance measures.

Chapter 4

System Design

This chapter will cover the system design of this research, the main parts of which are the process and design of the overall system. Recall that the overall system is responsible for outputting a key by accepting environmental data as input. It is also important to note that the overall system is composed of a number of ML models that are responsible for outputting a single bit of the key for each model. Both the overall system design and the different ML model designs will be discussed in this section, but they are only a part in the overall system process which will be discussed first in this chapter.

4.1 System Process

While the training and testing of different models in the system design are very important in this research, they are not possible without the steps leading up to and following them.

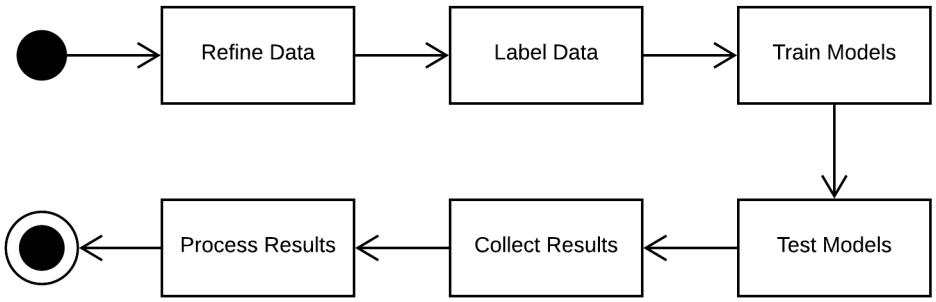

Figure 4.1: System process

As can be seen in figure 4.1, there are two major steps leading up to training the models: refining the data and labeling the data. These steps are part of the data preprocessing that was discussed in the Data Collection section of the previous chapter, and are integral to the functioning of the system. Refining involved removing the unnecessary and unwanted data from the larger dataset, as well as limiting it to only the number of samples that would be used to train and test the models. Labeling the data was a necessary step in order to perform the supervised learning in this study. The label given to each sample was the key that the models would be trained with and attempt to predict.

The next steps were the training and testing of the models. The models were trained using the training data and labels that were prepared in the previous steps. Once this was complete, the models were tested using the test data that was prepared with the training data.

Finally, the results from testing the models were gathered and processed. By using the tools discussed in the Tools section of the previous chapter, metrics were able to be gathered from the results. These results were then processed to yield actual, readable data based on the specifics of system design. The results will be discussed further in the next chapter, but the rest of this chapter will focus on the System Design used by the training and testing steps.

4.2 Overall System Design

As mentioned in the introduction to this chapter, the overall system is responsible accepting enironemntal data as inputs and outputting a key. These two layers of the overall design can be seen in Figure 4.2 with all environmental variables as input nodes on the left and all of the bits of the final key as output nodes on the right. In order to transform from the inputs to the outputs in this system, ML models are used, which are represented in a middle layer between the two outer layers.

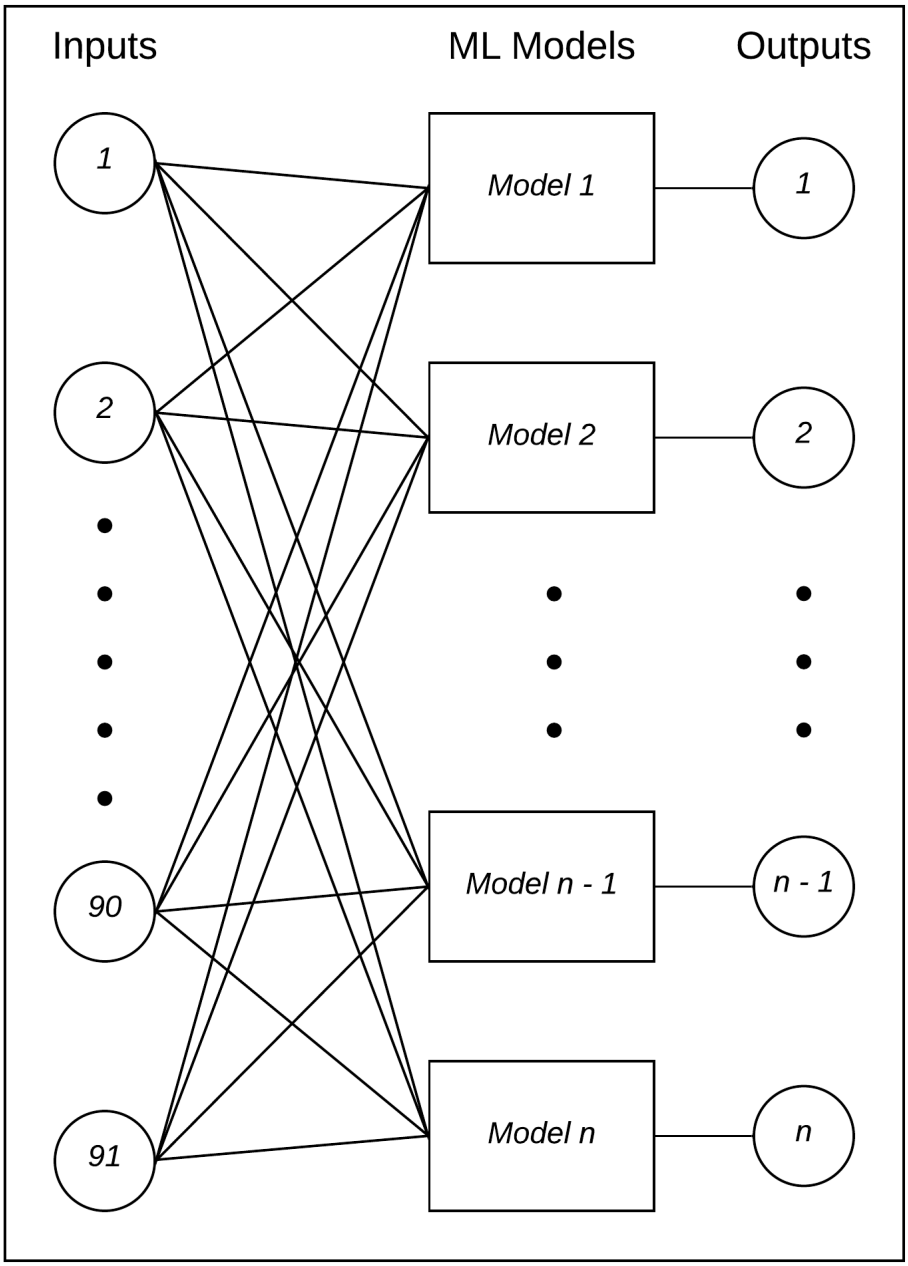

Figure 4.2: Overall system design

The leftmost layer is the input layer. As previously stated, the nodes in this layer represent the the environmental variables that are the inputs to this system. Note here that there are 91 input nodes used in this system while there are only 44 environmental properties considered after data collection and refinement. This is result of the one-hot encoding process discussed in Section 3.2: Data Collection which has expanded the number of environmental variables to consider by splitting the categorical data into multiple fields.

The middle layer is the ML models layer. As can be seen in Figure 4.2, this layer is composed of n ML models, where n is the length of the final key. Each of these models will accept all 91 inputs and output a binary bit to one of the output nodes.

The rightmost layer is the output layer, whose nodes are used to create the key. This is done by using each output node in order as a binary bit in the final key, allowing for the creation of a key of length n as represented by Figure 4.2. A system capable of generating a key of variable length is necessary for this research as the effects of different length keys will be assessed.

4.3 Machine Learning Model Designs

This section will focus on the designs and uses of the ML models from Figure 4.2. As discussed in the methodology chapter, the ML models that will be used in this research are Decision Tree and Neural Network. All models will receive the same inputs and will output a single binary bit and only one single type of model will be used at a time. In other words, all n models will either be Decision Trees or Neural Networks with no exceptions.

The Decision Tree model used for this research is constructed from the Scikit Learn Decision Tree Classifier and uses the classifier's default parameters [11].

The Neural Network model used is constructed from the Scikit Learn Neural Network Multi-layer Perceptron Classifier and uses all of its default parameters except for the shape of the hidden layers [12].

As noted in the methodology chapter, these models will be trained by placing significance on different environmental inputs. In other words, one model may be trained to output a certain value when the environment has 0 antivirus products enabled, but this value may have no impact on the output of another model.

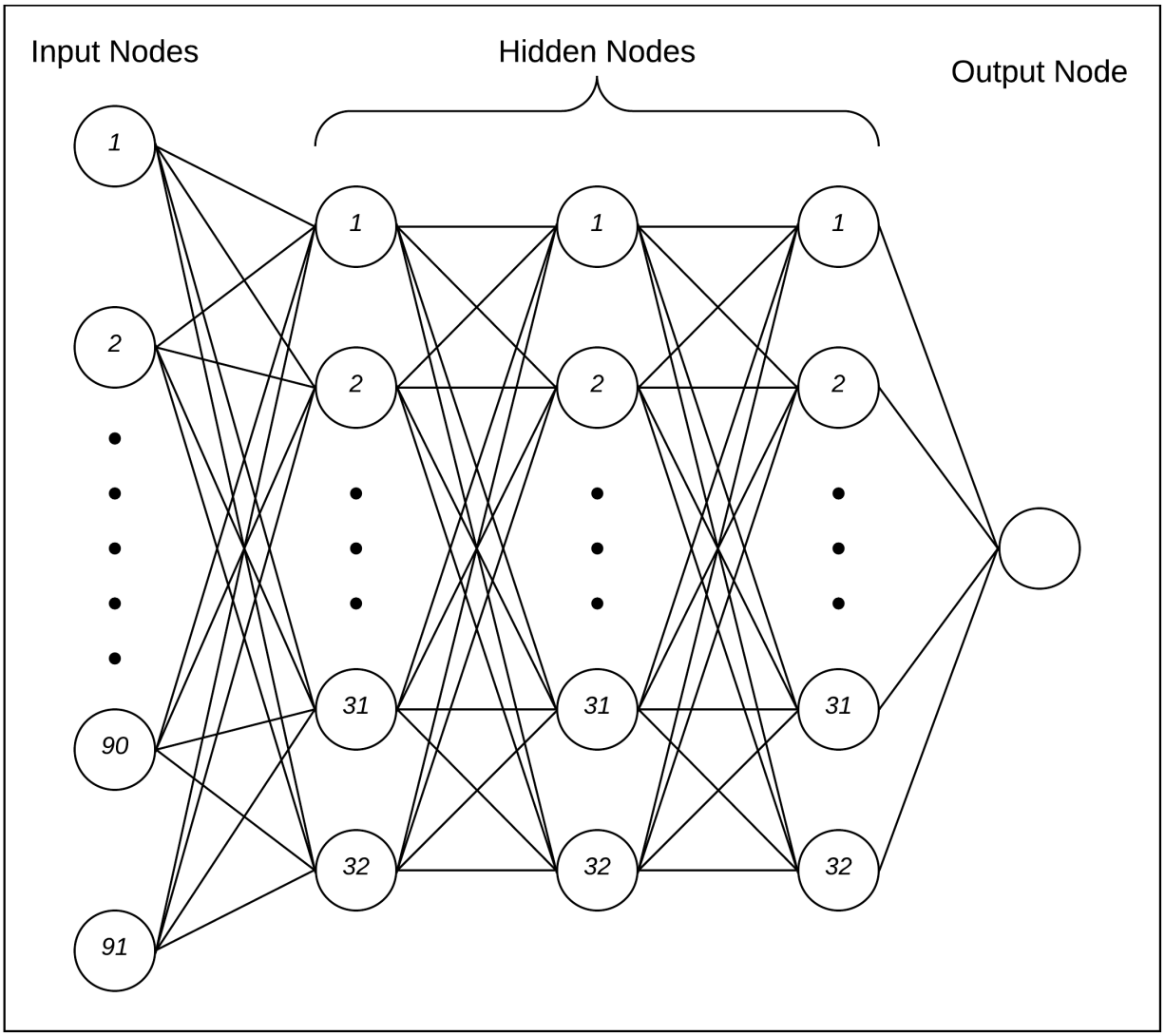

Figure 4.3: Single neural network design

As shown in Figure 4.3, this is a deep neural network with three hidden layers that are each composed of 32 hidden nodes. Furthermore, there is a connection between all nodes of consecutive layers.

This system design was used to test several different key-generation methods. Various results from those tests were gathered and will be discussed in the next chapter: Results.

Chapter 5

Evaluation

5.1 Test Results

This section will cover the results gathered from implementing the system design of the previous chapter and running the tests and data through it that were described in the Methodology chapter.

There were 14 tests in total that were run as part of this research: 7 neural network tets and 7 decision tree tests. For each type of ML model, the tests were organized by (1) the number of relevant conditions that should affect each bit of the key and (2) the length of the final key. In order to establish patterns in these two factors, the tests were organized such that one factor would remain constant while the other was permitted to change. As a result, the 7 tests for each type of model were:

1. Number of Relevant Conditions: 2; Key Length: 16 bits
2. Number of Relevant Conditions: 3; Key Length: 16 bits
3. Number of Relevant Conditions: 4; Key Length: 16 bits
4. Number of Relevant Conditions: 5; Key Length: 16 bits
5. Number of Relevant Conditions: 2; Key Length: 32 bits
6. Number of Relevant Conditions: 2; Key Length: 64 bits
7. Number of Relevant Conditions: 2; Key Length: 128 bits

Furthermore, it is important to note that each of these tests was run 10 times, with all values in this section being the averages of the values gathered from the 10 iterations. For each iteration, the model was trained with 15,000 samples and tested with 5,000 samples consistent across all models. This has been done to account for the variety in the models predictions even when trained and tested with the same data, particularly with neural networks.

The confusion matrices for each of the 14 tests are provided below. The values from these confusion matrices will be used throughout the section to represent the results.

%		Actual	
		Positive	Negative
Predicted	Positive	6.20	5.70
	Negative	5.98	82.12

Table 5.1: Confusion Matrix: Neural Network; Relevant Conditions: 2; Key Length: 16 bits

%		Actual	
		Positive	Negative
Predicted	Positive	17.60	9.25
	Negative	3.70	69.45

Table 5.2: Confusion Matrix: Neural Network; Relevant Conditions: 3; Key Length: 16 bits

%		Actual	
		Positive	Negative
Predicted	Positive	7.20	14.41
	Negative	3.32	75.07

Table 5.3: Confusion Matrix: Neural Network; Relevant Conditions: 4; Key Length: 16 bits

%		Actual	
		Positive	Negative
Predicted	Positive	1.45	9.80
	Negative	10.28	78.42

Table 5.4: Confusion Matrix: Neural Network; Relevant Conditions: 5; Key Length: 16 bits

These first four confusion matrices (Tables 5.1 through 5.4) show the results from the neural network tests in which the key length remained the same and the number of relevant conditions increased. As can be seen, the system is able to correctly predict many keys, but still makes a significantly high percentage of false predicitions.

%		Actual	
		Positive	Negative
Predicted	Positive	3.64	11.64
	Negative	6.28	78.44

Table 5.5: Confusion Matrix: Neural Network; Relevant Conditions: 2; Key Length: 32 bits

%		Actual	
		Positive	Negative
Predicted	Positive	1.96	4.40
	Negative	7.20	86.44

Table 5.6: Confusion Matrix: Neural Network; Relevant Conditions: 2; Key Length: 64 bits

%		Actual	
		Positive	Negative
Predicted	Positive	1.26	4.32
	Negative	8.44	85.98

Table 5.7: Confusion Matrix: Neural Network; Relevant Conditions: 2; Key Length: 128 bits

These three confusion matrices (Tables 5.5 through 5.7) show the results from the neural network tests in which the number of relevant conditions remained the same while the key length increased. Similar to the other neural network tests, it can be seen that the system is able to predict many keys correctly, but still makes a large amount of false predictions.

%		Actual	
		Positive	Negative
Predicted	Positive	12.18	0.00
	Negative	0.00	87.82

Table 5.8: Confusion Matrix: Decision Tree; Relevant Conditions: 2; Key Length: 16 bits

%		Actual	
		Positive	Negative
Predicted	Positive	21.30	0.00
	Negative	0.00	78.70

Table 5.9: Confusion Matrix: Decision Tree; Relevant Conditions: 3; Key Length: 16 bits

%		Actual	
		Positive	Negative
Predicted	Positive	10.48	0.00
	Negative	0.04	89.48

Table 5.10: Confusion Matrix: Decision Tree; Relevant Conditions: 4; Key Length: 16 bits

%		Actual	
		Positive	Negative
Predicted	Positive	11.78	0.04
	Negative	0.00	88.18

Table 5.11: Confusion Matrix: Decision Tree; Relevant Conditions: 5; Key Length: 16 bits

Similar to the first four confusion matrices of the section, Tables 5.8 through 5.11 show tests for increasing the number of relevant conditions while keeping the key length the same, but using decision trees instead of neural networks. It is clear from these figures that decision tree is significantly better than neural network at predicting the correct keys as the length increases. In fact, Tables 5.8 and 5.12 have 0% false predictions and Tables 5.13 and 5.14 have only 0.04%.

%		Actual	
		Positive	Negative
Predicted	Positive	9.92	0.00
	Negative	0.00	90.08

Table 5.12: Confusion Matrix: Decision Tree; Relevant Conditions: 2; Key Length: 32 bits

%		Actual	
		Positive	Negative
Predicted	Positive	9.16	0.00
	Negative	0.00	90.84

Table 5.13: Confusion Matrix: Decision Tree; Relevant Conditions: 2; Key Length: 64 bits

%		Actual	
		Positive	Negative
Predicted	Positive	9.70	0.00
	Negative	0.00	90.30

Table 5.14: Confusion Matrix: Decision Tree; Relevant Conditions: 2; Key Length: 128 bits

These final four confusion matrices (Tables 5.12 through 5.14) have very similar results to the first four decision tree tests. These last three tests had the same number of relevant conditions with an increase in the key length. All of these confusion matrices have 0% false predictions.

These results and the patterns and implications that can be derived from them will be discussed in the next two sections.

5.2 Evalutating Ability to Generate the Correct Key

In the Methodology section, it was noted that two main concepts would be analyzed in this research: (1) the ability of the system to generate the correct key and (2) the difficulty of the system to be reverse-engineered. This section and the next will explore these two areas of evaluation, beginning with the ability to generate the correct key.

This first section focuses on evaluating the results presented in the previous chapter, starting with the overall accuracy and f1-scores of the different types of models and then moving to more specific results.

Model	Average accuracy	Average f1-score
Neural Network	0.8503	0.3240
Decision Tree	0.9999	0.9995

Table 5.15: Cumulative Model Results

When analyzing accuracy and f1-score, both values are desired to be as close as possible to 1.0. Comparing the cumulative results of all tests for each model as shown in figure 5.15, it is clear that decision tree has both a better accuracy and f1-score than neural network. This means that it is better at both predicting the correct key and not incorrectly predicting the correct key. To determine why this is, patterns in the tests can be analyzed, starting with neural network.

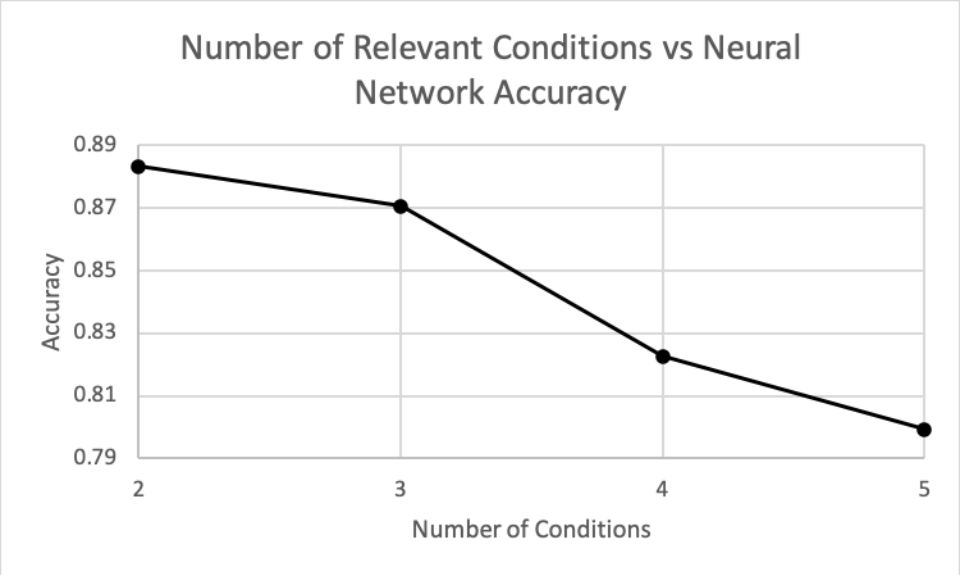

Figure 5.1: Neural Network: Number of Relevant Conditions vs Accuracy

Figure 5.1 shows the relationship between the number of relevant conditions per model and the accuracy in predicting the key using a neural network. This clearly shows that a system of neural network models is capable of being trained to output the correct key when provided with environment features. However, increasing the number of features that should affect the output of the models leads to a lower overall accuracy. This makes sense, as the models must be trained to identify and distinguish the effects of more features on the final outcome for each bit of the key.

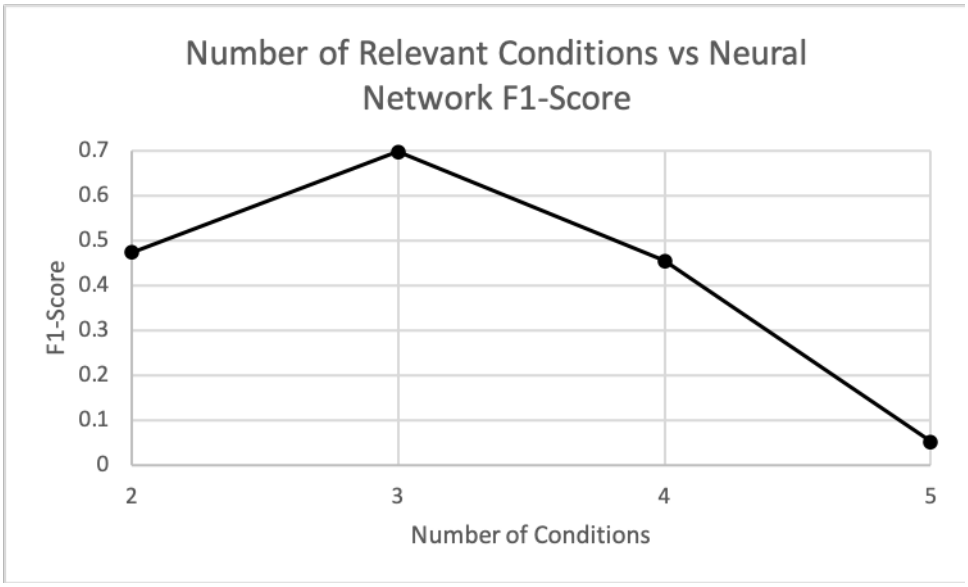

Figure 5.2: Neural Network: Number of Relevant Conditions vs F1-Score

As is shown figure 5.2, the f1-score also decreases as more features affect the output of the models. This correlates with the results from figure 5.1 and makes sense for the same reason as the decrease in accuracy, in that the models must be trained to identify and distinguish the effects of more features and are therefore less successful at their predictions.

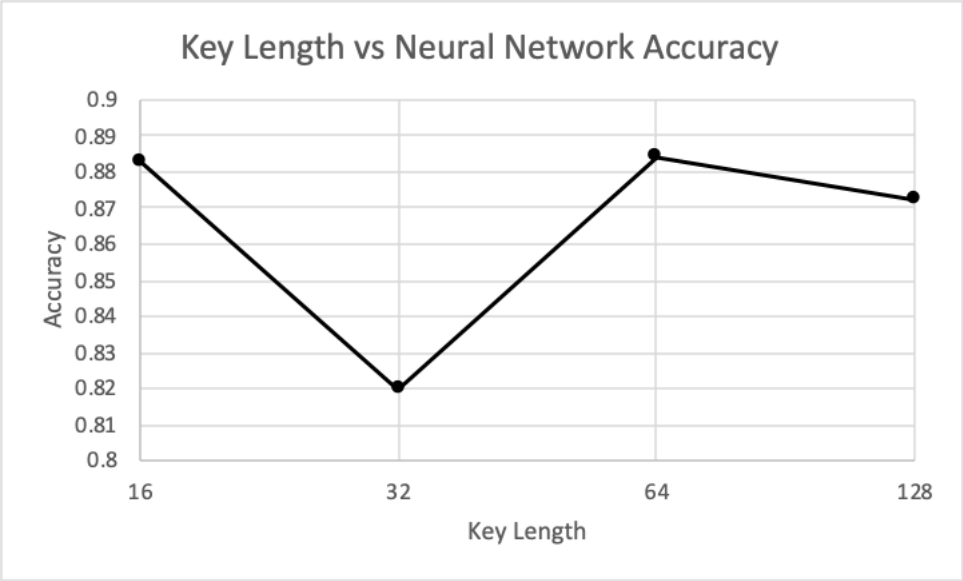

Figure 5.3: Neural Network: Key Length vs Accuracy

Figure 5.3 shows no clear pattern in the accuracy of the system as the key length increases. Therefore, the f1-score will be analyzed in order to infer a more clear interpretation of the system's abilities as result of an increase in key length.

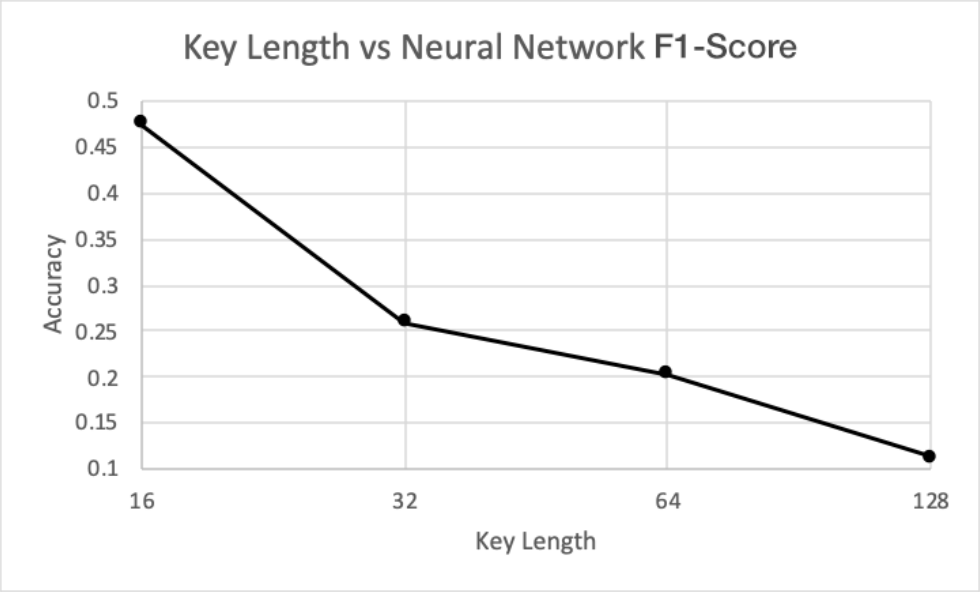

Figure 5.4: Neural Network: Key Length vs F1-Score

As opposed to the accuracy, figure 5.4 shows a clear decrease in f1-score as the key length increases. This means that, similar to the increase in relevant conditions, a system of neural networks becomes increasingly unable to generate the correct key as the key length increases. This makes sense because as the number of bits that make up a key increase, so do the number of possible keys that can be generated. For example, a key with a length of 16 bits yields 2^{16} possbile keys, while a key of length of 128 yields 2^{128}.

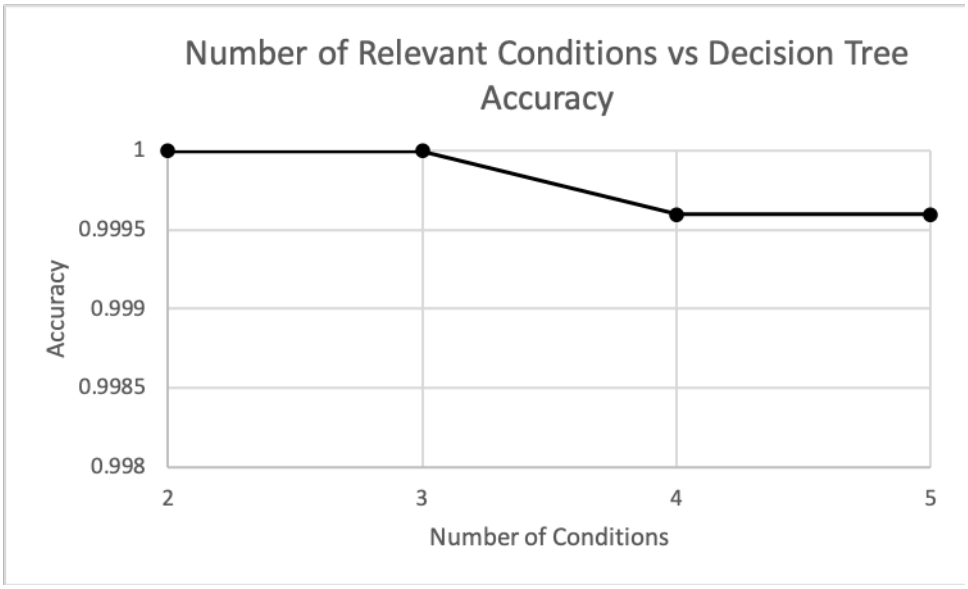

Figure 5.5: Decision Tree: Number of Relevant Conditions vs Accuracy

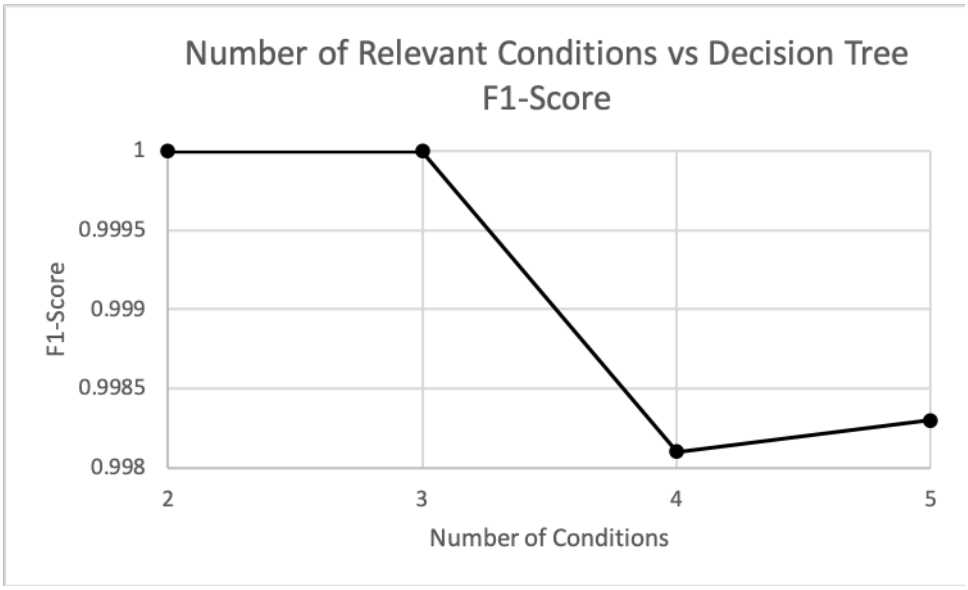

Figure 5.6: Decision Tree: Number of Relevant Conditions vs F1-Score

Figures 5.5 and 5.6 show the patterns in accuracy and f1-score in terms of the number of relevant conditons for a system of Decision Tree models. It is clear from these figures that a system consiting of Decision Trees is much better at generating the correct key as the number of relevant conditions increases than a system of Neural Networks. Although both accuracy and f1-score decrease as the number of relevant conditions increases, it is important to note that this decrease is very slight, and much less than the decrease found in Neural Networks when presented with the same changes.

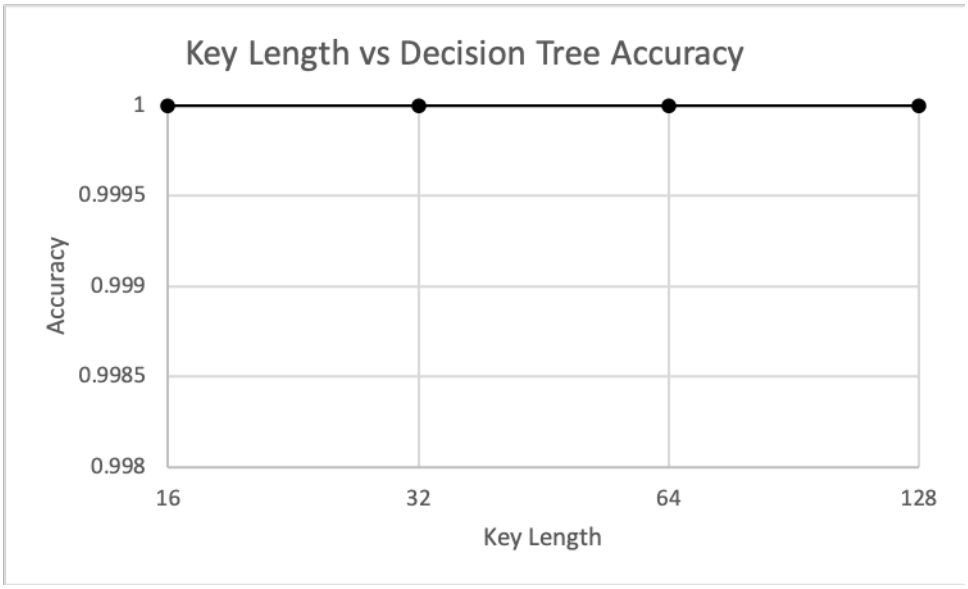

Figure 5.7: Decision Tree: Key Length vs Accuracy

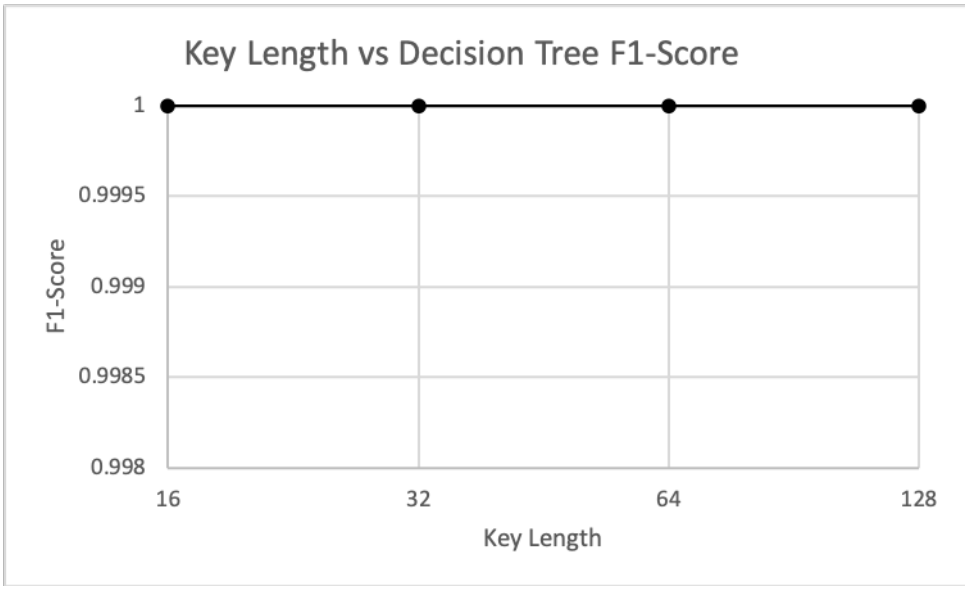

Figure 5.8: Decision Tree: Key Length vs F1-Score

Figures 5.7 and 5.8 show consistently perfect accuracies and f1-scores as key length increases when using a system of Decision Trees. This again shows that Decision Trees are much better at generating the correct key than Neural Networks. This is also very good for potentially expanding the length of keys in future systems to reflect longer encryption keys, as there is no downward trend from the tests that have been run as part of this research.

5.3 Evaluating Difficulty to be Reverse-Engineered

In order for this system to difficult to reverse-engineer, it should be able to accept a large number of inputs and hide the trigger conditions necessary to generate the correct decryption key. This comes from an article on DeepLocker written by Marc Ph. Stoecklin (refer to the Literature Review for more information on DeepLocker). Steocklin states that by using a large enough number of inputs, it is ”virtually impossible to exhaustively enumerate all possible trigger conditions”[6]. This combined with the ”black box” nature of many AI models (namely Deep Neural Networks) helps to hide the trigger conditions. Therefore, in order to determine the difficulty of reverse-engineering this system, two points will be considered: (1) the ability of the system to accept a large number of inputs and (2) the ability of the system to hide the trigger conditions.

The way this system is set up, it is scalable for the number of inputs it can accept. As can be seen in Figure 4.2, the system is composed of a number of models which each accept all inputs to the system. Being that each of these models is separate from the rest of the models in the system and only used to generate a single bit in the final key, they are each able to accept any number of inputs. Each of these models also has the ability to be altered on a case-by-case basis without affecting the inputs to the system as a whole. Therefore, this system would only be limited by the time necessary for training each of these models with the desired number of inputs, particularly as the length of the key (and thus the number of models) increases.

While the sytem may be scalable for a large number of inputs, there is a potential drawback to

the design of this system when it comes to the ability to hide the trigger conditions. Because the system is set up as a number of models, each of which predict a bit in the decryption key, it would be possible for an analyst to systematically change one value at a time that is fed as an input to the system and look at which of the bits of the generated key have changed. In doing this, they would be able to determine which conditions affect which the bits output by the system. This would get an analyst one step closer to determining the trigger conditions, as they could then compare the outputs of certain models to determine when a trigger condition is met or not.

5.4 Summary of Evaluations

In terms of the ability to generate the correct key, this chapter has demonstrated that both a model of neural networks and a model of decision trees are able to accuractely predict the correct key; however, decision trees are much better than neural networks. A system of decision trees does not significantly decrease in accuracy or f1-score as the number of relevant conditions and key length increase, while both accuracy and f1-score decrease significantly with a sytem of neural networks.

In terms of the ability of the system to be reverse-engineered, this chapter has shown that the system is scalable because it uses a large number of smaller models. However, for the same reason that it is scalable, it lacks in its ability to effectively hide the trigger conditions.

The next chapter will take the results of these evaluations into account with the rest of the research and seek to provide a response to the initial purpose of this reserach: to assess the validity of using AI to encrypt and decrypt evasive malware capable of performing targeted attacks through the creation and analysis of a prototype system of machine learning models.

Chapter 6

Conclusions

This conclusion seeks to make a response to the objective posed in the purpose of this research, but first, the results gathered from the entire research process will be reviewed.

6.1 System Review

Pros

This research has shown that it is possible to collect data from a large number of computer environments and refine it into relevant features and valid samples in order to be used as input to machine learning models.

It is possible to then choose a desired key length, assign relevant trigger conditions to each of the bits in that key, label each sample in the data based on those trigger conditions, and then split the data into a training set and test set.

A system of machine learning models can then be trained and tested with that labeled data, and data which represents the results of the predictions on those trained models can be gathered in order to analyze the performance of the system as a whole.

It was determined using different types of models that, while some models are much better than others, it is possible to use machine learning models to accurately predict the correct key.

Finally, this system is scalable in terms of the number of inputs, allowing for a very large number of inputs to be fed into it.

Cons

One downside to this design is that it is possible to reverse-engineer the trigger conditions, as a result of having one model per bit of the resulting key.

6.2 System Validity Assessment

Recall that in the purpose of this research as defined in the Introduction was ”to assess the validity of using AI to encrypt and decrypt evasive malware capable of performing targeted attacks through the creation and analysis of a prototype system of marchine learning models.”

While there more pros than cons to using this sytem, it is still important to note that the single con listed in the previous section is a rather large problem. The ability to hide the trigger conditions is part of what makes the malware evasive. If the system is able to be easily reverse-engineered and unlocked, it does not succeed in being evasive malware. As a result, the system that was created as part of this research does not entirely meet the requirements presented in the purpose, and therefore cannot be considered an entirely valid system. However, that does not mean this research was for naught.

While the system does not necessarily meet the requirement of being effectively evasive, it is still capable of using AI to encrypt and decrypt malware capable of accuratlely performing targeted attacks. This system is still successful in a number of the requirements set at the beginning of the research, and can be improved upon in order to eliminate its current shortcomings.

6.3 Considerations for Future Improvement

In order to improve this system, it could be changed to be composed of one large machine learning model. Such a model would accept the inputs in the same way that the smaller machine learning models in this system do, but output the entire decryption key instead of just one of the bits. The techniques and tools used for data collection, data refinement, data labeling, results collection and results analysis could be adopted from the current system. Ideally, the system would be able to train and test such a model with results that are as good as or better than the results of the current system with the added bonus of not having individual trigger conditions affect the output of just a single bit of the output key. This new system would maintain parts of this research that are proven to work, while effectively improving its shortcomings as an evasive system.

Bibliography

[1] Simon Kramer and Julian C. Bradfield. A general definition of malware. *Journal in Computer Virology*, 6(2):105–114, 2010.

[2] Roger A. Grimes. *Hacking the Hacker: Learn from the Experts Who Take down Hackers*. John Wiley & Sons, Incorporated, 2017.

[3] Bhushan Kapoor, Pramod Pandya, and Joseph S. Sherif. Cryptography. *Kybernetes*, 40(9):1422–1439, 2011. Copyright - Copyright Emerald Group Publishing Limited 2011; Last updated - 2019-09-06; CODEN - KBNTA3.

[4] A. Jadhav, D. Vidyarthi, and M. Hemavathy. Evolution of evasive malwares: A survey. *2016 International Conference on Computational Techniques in Information and Communication Technologies (ICCTICT)*, 2016.

[5] A. K. Sood and R. J. Enbody. Targeted cyberattacks: A superset of advanced persistent threats. *IEEE Security & Privacy*, 11(1):54–61, 2013.

[6] Marc Ph. Stoecklin. Deeplocker: How ai can power a stealthy new breed of malware. https://securityintelligence.com/deeplocker-how-ai-can-power-a-stealthy-new-breed-of-malware/.

[7] Dhilung Kirat, Jang Jiyong, and Marc Ph. Stoecklin. *DeepLocker - Concealing Targeted Attacks with AI Locksmithing*. Black Hat USA 2018, Aug 2018.

[8] Kaggle. Microsoft malware prediction. https://www.kaggle.com/c/microsoft-malware-prediction/data.

[9] pandas. pandas. https://pandas.pydata.org/.

[10] scikit learn. scikit-learn machine learning in python. https://scikit-learn.org/stable/.

[11] scikit learn. sklearn.tree.decisiontreeclassifier. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html.

[12] scikit learn. sklearn.neural_network.mlpclassifier. https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html.

ACADEMIC VITA

David Lengel

Education:

Bachelor of Science Degree in Software Engineering
with Honors in Software Engineering
The Pennsylvania State University Erie, The Behrend College
Erie, PA
Spring 2020
Thesis Title: Assessing the Validity of Using AI to Encrypt Evasive Malware
Thesis Supervisor: Zhifeng Xiao

Related Experience:

Software Engineering/Development Intern
University of Pittsburgh Medical Center (UPMC) Health Plan
Pittsburgh, PA
Summer 2018, Summer 2019

IT Helpdesk Technician
The Pennsylvania State University Greater Allegheny
McKeesport, PA
August 2016 – May 2018

Awards:

Dean's List
Behrend Excellence Award
GTE Foundation Scholarship
Young Award in Engineering
Penn State Greater Allegheny 2018 Scholarship Recipient of the Year
Blue & White Scholarship
Greater Allegheny General Scholarship