SAT Based Attacks on Machine Learning Models

A B. Tech Project Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of

Bachelor of Technology

by

Saaketh Gunti
(190101080)

under the guidance of

Dr. Chandan Karfa

to the

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI
GUWAHATI - 781039, ASSAM

CERTIFICATE

*This is to certify that the work contained in this thesis entitled "**SAT Based Attacks on Machine Learning Models**" is a bonafide work of **Saaketh Gunti** (**Roll No. 190101080**), carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati under my supervision and that it has not been submitted elsewhere for a degree.*

Supervisor: **Dr. Chandan Karfa**

Assistant/Associate

Professor,

May, 2023

Guwahati.

Department of Computer Science & Engineering,

Indian Institute of Technology Guwahati, Assam.

Acknowledgements

I would like to thank Dr. Chandan Karfa sir for his guidance throughout the year for this project.

Contents

Abstract

In this era where Machine Learning models are used in most of the applications that we use in day-to-day life, Model-Extraction problem is getting paramount attention. In Model extraction we try to find the network weights, provided we have the structure of the neural network and a black-box access to the network (from which we get the predictions of the model for the given set of inputs without knowing the inner details of the model). We try to find a SAT based attack for the Model-Extraction problem, given the oracle. In this report we try to present the results of the experiments we did on neural networks with SAT based attack (SMT Attack).

Chapter 1

Introduction

1.1 Background

The enormous research and development that has gone into this sector in recent years has allowed for the resolution of many issues that were not within the purview of neural networks. Since the last ten years, neural networks have become more effective and widespread, and they are now employed in a wide range of everyday applications. Some of these applications, including Alexa and Siri, employ neural networks for speech recognition. These have been refined to the point where a Microsoft neural network can discern images more effectively than the human eye.

Because of their critical importance, these neural networks must be protected with the utmost care. If the neural network has all the necessary parameters and a properly constructed network. According to [CLE+19], it is feasible to obtain crucial and sensitive information from neural network parameters. It is also possible to damage the neural network by feeding it examples that could make it worse. Therefore, rather than revealing all the parameter values to the user, a black-box access or query access is typically granted to the user in order to safeguard the confidentiality and integrity of the neural network and its data. Notice that it would be challenging for users to obtain the parameters through training if the network's structure was made public. This is because training requires expensive resources, such as

a large training data set and powerful computing power, neither of which are commonly available.

For the above reasons, people try to make a replica of the parameters of the network by using the oracle and the structure of the Neural Network. In this report, we try to explore this possibility by using a SAT-based attack for finding the parameters.

Here, we are attaching a model of a neural network to establish the basic idea required to understand the work in this report.

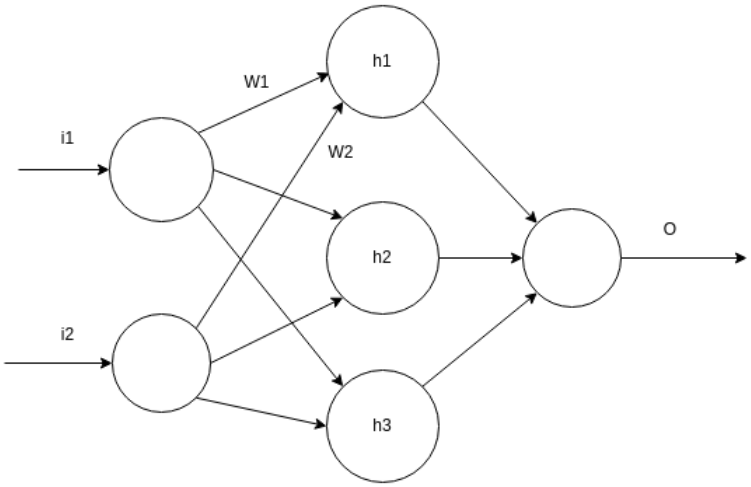


Fig. 1.1: 2*3*1 NN with no biases

The above network is a three-layer network. They are an input layer, a hidden layer, and an output layer with 2, 3, and 1 neurons respectively. i1 and i2 are the inputs to the input layer. h1, h2 and h3 are the values at the hidden layer and o is the output of the neural network. The value at a layer in a neural network is computed as the weighted sum of inputs from the previous layer i.e $h1 = i1 * W1 + i2 * W2$. If there is non-linearity (ReLU) at the hidden layer then the values of h1 are the maximum of zero and $i1 * W1 + i2 * W2$.

1.2 Related Work and Motivation

The research work in the domain of Model Extraction problems is categorized into two types fidelity and task-accuracy extraction [JCB+20]. Fidelity extraction is trying to reproduce

the network that is perfectly consistent with the oracle for all inputs whereas task-accuracy extraction concentrates on getting similar models that solve the underlying ground task. Some of the works in task-accuracy extraction of networks are Stealing machine learning models via prediction [TZJ+16]. Some works in fidelity extraction are Evaluating and testing unintended memorization in neural networks [JCB+19]. We focus on the fidelity extraction domain in our report.

Logic encryption is done so that the structure of the hardware is not known to everyone. Key inputs which are kept private and have fixed values are added to the hardware to keep it secure. SAT-based attacks are used to find the value of the keys with the help of the oracle and the structure of the Integrated Circuit (IC). There are a variety of attacks based on SAT such as the Incremental SAT Attack proposed by [YZL+17] and the Cyclic SAT attack proposed by [ZJK17].

The problems of Model Extraction in Neural Networks and the breaking of Logic Encrypted Integrated Circuits are similar. In both cases, we have the structure and the oracle with query-based access and we want to find out the values of parameters and key inputs respectively. Our motivation in this work is to explore the problem of Model Extraction with SAT-based attacks on which there is no significant research work. In this work, we try to check and measure the limitations and advantages we can get by using the SAT attack on the Model Extraction problem.

In phase 1 of the project, we had the type of oracle where we get the output of the last layer of the neural network. In the paper Stealing Neural Network Models through the Scan Chain [PA21] with the use of scan chain technology, we can get the output at each layer after the activation. In this phase of the project, we use the output values at each layer to find the parameters of the neural networks. Also, we only consider fully connected neural networks in this report.

1.3 Problem Statement

Given the structure and oracle access to the Neural network, we have to find a combination of parameters such that the network built gives the exact same result as the oracle for all valid input combinations using an SAT-based attack.

Formally, let $f : X \rightarrow Y$ represent the Neural Network, we want to find a function $g : X \rightarrow Y$, such that for every input vector x $\in X$, $f(x) = g(x)$ using SAT-based attack.

1.4 Contribution

The method that we used to find the parameters is SMT Attack proposed by [AKHS19] which is an SAT-based attack. Traditional SAT attack deals with only boolean key inputs, whereas SMT attack can deal with the datatypes such as booleans, integers, and real numbers. In this report, we showcase the results of the various experiments, and approaches we did in doing the fidelity extraction of multiple neural networks of different structures and non-linearities with the SMT attack. We would also like to discuss the various complications and limitations of the SMT attack and the time it is taking to find the right parameters. Notice that our method does not depend on any of the factors involved while training the neural network or the data set it is trained on.

Chapter 2

Preliminaries

Here we would establish some of the definitions, facts, and notations that we use later in the report.

2.0.1 Distinguishing Input Pattern

Distinguishing Input Patterns (DIPs) are inputs for which the two possible key combinations will give different results. Out of the two results, either one or both the results might be wrong. Using these input patterns we will recognize and remove the wrong key values from the possible set of keys.

2.0.2 Neurons

Neurons are the building blocks of Neural networks, neurons can be understood as functions that take multiple inputs and give out an output. The output is the weighted linear combination of input values. For every neuron, there might be bias which has a fixed value and weight. Neurons can have the additional computation, once the linear combination value is calculated that value is given to an activation function. It is this activation function that adds non-linearity to the neural network. Some examples of activation functions are Sigmoid, Tanh, ReLU etc. In this report, we focus only on the ReLU activation function.

2.0.3 Parameters

Let W be the weights between the layers and B be the biases of all the neurons in the neural network. The set of weights and biases is combinedly known as the parameters of the neural network. Parameters of the neural network are represented by $\theta = \{W, B\}$. Notice that the parameters are the actual keys that we want to find with the SMT attack.

2.0.4 Notations for Parameters

As there can be many layers in a neural network, we need a certain notation to keep track of the parameters. So we define a notation for weights and biases of the neural network.
We denote weights as $W^{(i)}_{j,k}$ where i corresponds to the layer number, the weight belongs to the path from neuron j in layer i and neuron k in layer $i+1$.
We denote biases as $B^{(i)}_{j}$ which signifies that this bias corresponds to neuron j in layer i.

2.0.5 Notation for Neural Network

Suppose we have a k layer neural network with the number of neurons in ith layer denoted as d_i. The neural network is represented as $d_1 * d_2 * d_3 * ... * d_k$

Chapter 3

SMT Attack with End-to-End Oracle

3.0.1 SMT Attack Overview

Here we add a brief overview of the SMT Attack proposed by [AKHS19]. In this attack, we will consider all possible key value combinations in the domain. We will try to find a DIP for the possible set of keys, then we query the oracle for that particular DIP and check the result of the oracle with the results we get with the keys. The keys that are giving a different result than the oracle will be removed from the possible set of key combinations. We will try to find a DIP again such that it divides the possible set, and will repeat the same process until all the keys in the possible set are valid i.e all the keys in the possible set gives the exact same results as the oracle for all the input patterns in the domain.

For example, consider a gate level hardware with 2 boolean key inputs, then the possible set of keys is {00, 01, 10, 11}. Now, suppose we have a input {1, 1} for which key combination k1 = {00} gives 0 and k2 = {01} gives 1, then the input {1, 1} is a DIP. Now, we query the oracle for the output, assuming that the oracle output is 1. Then our combination k1 is wrong, thus we remove k1 from the possible set and repeat the same process till all the key combinations are correct.

3.0.2 Insight

There can be multiple possible parameters sets such that all of them exactly matches the predictions of the oracle for all possible input combinations in the domain. Consider the case of a neural network with no non-linearity, then the output value is a weighted linear combination of the inputs, so for that linear combination, there can Timeout ly many possible solutions.

Consider the following example of a $2 * 2 * 1$ neural network for understanding. Here i_1, i_2 are the inputs, o is the output, and h_1, h_2 are the values at the hidden layer i.e layer 2 of the network.

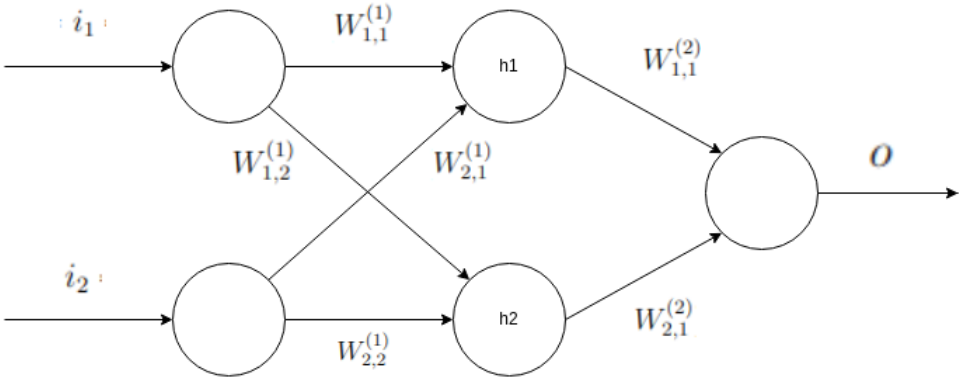


Fig. 3.1: 2*2*1 NN with no biases

h_1 is computed as $h_1 = i_1 * (W^{(1)}_{1,1}) + i_2 * (W^{(1)}_{2,1})$

h_2 is computed as $h_2 = i_1 * (W^{(1)}_{1,2}) + i_2 * (W^{(1)}_{2,2})$

output is computed as $o = h_1 * (W^{(2)}_{1,1}) + h_2 * (W^{(2)}_{2,1})$

From the above equations, we get

$o = i_1 * (W^{(1)}_{1,1} * W^{(2)}_{1,1} + W^{(1)}_{1,2} * W^{(2)}_{2,1}) + i_2 * (W^{(1)}_{2,1} * W^{(2)}_{1,1} + W^{(1)}_{2,2} * W^{(2)}_{2,1})$

We can infer that the above formula has Timeout ly many possible solutions as the factors multiplied by inputs are linear equations and they can have Timeout ly many solutions.

Notice that even if we have ReLU nonlinearity, we can have multiple possible solutions that exactly match the oracle outputs as ReLU is a piecewise linear function.

Notice that for neural networks with no hidden layers, there can only be one possible solution. For example consider a $2*1$ network, the equation for the output would be $o = i_1*W_1+i_2*W_2$. For this output formulation to match the oracle then there will be only one possible way.

3.1 Phase 1 Results

Below is the table of the results we got from the experiments conducted in phase 1 of the project, where the for the given input, we get the output of the last layer from the oracle. In this semester's work, we improve upon the results given in this tab and also add other metrics such as the number of queries made to the Oracle to find all the parameters. The table contains the structure of the neural network (Notice that all neural networks are fully connected, so the number of weights present in the network can be calculated from the structure), the existence of bias and non-linearity in the neural network in mentioned in the Bias column and Non-Linearity column respectively. Notice that Bias is considered at the neuron level and non-linearity is considered at the layer level i.e if non-linearity exists then it will be present for all neurons in that particular layer. Known Keys are the parameters that we already know along with the structure of the neural network. Total Keys contains the total parameters(weights and biases) in the network that we have to find to completely match the behavior of our neural network with that of the oracle.

S No	Structure	Bias	Non Linearity	Known Keys	Total Keys	Time Taken	Result
1	2*1	None	None	None	2	0.02 sec	Success
2	4*1	None	None	None	4	0.04 sec	Success
3	1*1*1*1	None	None	None	4	3 sec	Success
4	2*2*1	None	None	None	6	32.97 sec	Success
5	2*2*1	None	2nd layer	None	6	512.77 sec	Success
6	2*2*1	1 at 2nd layer	None	None	7	27.19 sec	Success
7	2*2*1	2 at 2nd layer	None	None	8	96.85 sec	Success
8	2*2*1	1 at 2nd layer	2nd layer	None	7	2460.66 sec	Success
9	2*2*1	2 at 2nd layer	2nd layer	None	8	Timeout	Success
10	2*2*1	2 at 2nd layer	2nd layer	1 Bias	7	Timeout	Success
11	2*2*1	2 at 2nd layer	2nd layer	1 weight in L1	7	Timeout	Success
12	2*2*1	2 at 2nd layer	2nd layer	1 weight in L2	7	Timeout	Success
13	2*3*1	None	None	None	9	300.72 sec	Success
14	2*3*1	3 at 2nd layer	None	None	12	Timeout	Success
15	2*3*1	3 at 2nd layer	2nd layer	None	12	Timeout	Success

Table 3.1: Phase 1 Results

Here are the results from the data that we presented in the above table

1. As there can be multiple solutions for the neural network, we stop the execution when we find at least one correct possible key combination which matches the Oracle behavior. It is marked as Success if were able to find a key combination and Failed otherwise. Notice that if we have only an input layer and output layer then there will be only one possible solution for the key combinations.

2. Time taken increases with an increase in the number of keys. As the number of keys increases it is straightforward that the time taken to find a correct combination increases. We can infer that from the results of experiments 1, 2, 4, 6, and 7.

3. Time taken increases with an increase in non-linearity. With the increase in non-linearity (ReLU function), the complexity of the neural network increases as ReLU is a

piecewise linear function, the number of possible linear portions increases, so the time taken increases by a significant number. We can infer that from the experiments {4,5}, {6,8}, and {7,9}.

4. Time taken increases with an increase in the number of layers. With the increase in the number of layers, the number of parameters increases, so this conclusion directly follows from conclusion-1. With keeping the number of keys the same and increasing the number of layers in the network increases time. This happens because, with the increase in layers, the equation will have more multiplications among the keys which the SMT attack tool needs more time to solve. The second part of the conclusion can be verified from experiments 2 and 3.

5. Time decreases because of the known keys. If we have some keys known beforehand then the time to find the remaining keys decreases as it effectively decreases the number of keys to find. Also, note that finding 6 keys of 7 parameter network with one known key takes more time than finding 6 keys of a 6-parameter network. This is because the 7th known value comes into the formulation of the neural network and this formula is more complex than the 6 parameter network formulation. So the SAT tool takes a longer time.

6. Impact of non-linearity in the time taken is more than the number of keys to be found in the NN. It can be seen that a network with 9 keys can be broken within 300 seconds and a network with 7 keys and non-linearity is taking very long (Timeout) time to break. It can be understood that the linear portions formed due to ReLU are more complex than the formulation with comparatively more keys.

7. We are able to solve the network with a maximum of 7 keys with non-linearity and 12 keys without non-linearity. This is the current limitation of our SMT attack on neural networks.

Chapter 4

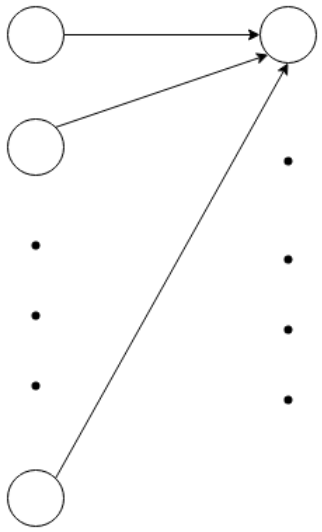
SMT Attack with Updated Oracle

4.1 Oracle Model

In phase-1 of the project, for every input the oracle gives the output of every node of only the last layer of the neural network. But in this phase, we also have the output of every hidden layer of the neural network along with the output layer after applying the activation function. The precedence for this kind of oracle is established in the paper [PA21] with the help of scan chain infrastructure. For example, consider Fig. 2.1, earlier for inputs $\{i_1, i_2\}$, oracle gives us {O}. Now, for input $\{i_1, i_2\}$, we can get $\{h_1, h_2, O\}$ from the oracle. We use this extra information from the Oracle to break the neural network with SMT attack.

4.2 Specific Query Approach

In this approach, we try to find the parameters layer by layer. In particular, we try to find all the weights going from layer 1 to layer 2 of the neural network and then go on to the next layer until we find all the weights. Also, while finding the weights in each layer we go node by node i.e if we are trying to find weights from layer t to layer {t+1}, then first we find the weights coming from layer t to the first node of layer {t+1} and move on to the next node. This approach is only used in the case of ReLU activation functions.



Consider node 1 of hidden layer 1, where there are n nodes in layer 1 of the neural network, the value of the node is given by

$h_1 = \text{ReLU}(i_1 * W^{(1)}_{1,1} + i_2 * W^{(1)}_{2,1} + ... + i_n * W^{(1)}_{n,1})$

In order to find the weights of the neural network of n variables (weights), we need 'n' linearly independent equations. We try to find the linear equations by using the SMT attack and find all the parameters, once we have all the weights coming to this node, we move on to the next node and then to the next layer. In this attack model, the keys of our SMT attack are the inputs of the neural network and we want to achieve the weights of the neural network by finding 'n' input combinations that give 'n' linearly independent equations on weights of the NN. To find the linear equations, we first have to remove the no-linearity of the ReLU unit, we try to do it by adding the constraint that the linear combination of inputs and weights is always positive to our SMT model. Now, from the SMT model, we get an input combination, for that input, we get a linear equation for the output of the particular node. Now as the next constraint, we consider the unit vector of the inputs and add it to the SMT model in a way that the next input we get from the SMT model is linearly independent of all the inputs, that we have got before and it should give positive value as the output. Once we get the desired number of linear equations, then we can easily solve for the weights of

the neural network. One corner case is that there might be a case that all weights going to a node are 0, then our SMT solver is stuck finding a possible input combination.

In this attack model, we focus on finding the weights by using the mathematical properties of the NN. Coming to the analysis of the attack model, to find all the weights coming to a node, we need to find 'n' linearly independent equations, thereby querying the oracle 'n' times. So, the total number of queries made to the Oracle is of the order n. i.e Q = O(n), which is not desirable also As the number of queries made is more, the complexity of the formula for the SMT solver is more, and the solver gets stuck. The bottleneck comes for the type of networks where each layer is of large size. Then finding those many linear independent combinations is a bottleneck for the SMT solver.

4.3 All constraints at once

In this approach, we try to find the parameters all at once, using the outputs of every layer of the neural network we get from the oracle. We add all the equations that are generated from the output of the oracle at once to the SMT solver.

In this method, we try to find all the keys at once by giving all the data we get from Oracle for a particular input to the SMT solver. We have significant improvement compared to the Phase-1 results using this approach and it is presented in the table below. The column updated time indicates the time taken by SMT solver using this approach and the old time indicates the time taken with the method in phase-1.

Now, we add the results of the experiments made using this approach in a tabular form, on comparison with the results obtained with the approach in phase-1. Old time stands for the time taken to find all weights using the approach in phase-1, and the updated time is the time taken to break all weights using this approach. Queries are the number of queries made to the oracle before termination.

S No	Structure	Bias	Non Linearity	Known Keys	Total Keys	Old Time	Updated Time	Queries	Result
1	2*1	None	None	None	2	0.02 sec	0.01 sec	2	Success
2	4*1	None	None	None	4	0.04 sec	0.01 sec	4	Success
3	1*1*1*1	None	None	None	4	3 sec	0.03 sec	2	Success
4	2*2*1	None	None	None	6	32.97 sec	0.04 sec	3	Success
5	2*2*1	None	2nd layer	None	6	512.77 sec	0.25 sec	5	Success
6	2*2*1	1 at 2nd layer	None	None	7	27.19 sec	0.03 sec	3	Success
7	2*2*1	2 at 2nd layer	None	None	8	96.85 sec	2.81 sec	3	Success
8	2*2*1	1 at 2nd layer	2nd layer	None	7	2460.66 sec	0.30 sec	4	Success
9	2*2*1	2 at 2nd layer	2nd layer	None	8	Timeout	0.07 sec	3	Success
10	2*2*1	2 at 2nd layer	2nd layer	1 Bias	7	Timeout	2.74 sec	3	Success
11	2*2*1	2 at 2nd layer	2nd layer	1 weight in L1	7	Timeout	1.22 sec	3	Success
12	2*2*1	2 at 2nd layer	2nd layer	1 weight in L2	7	Timeout	2.31 sec	3	Success
13	2*3*1	None	None	None	9	300.72 sec	0.21 sec	3	Success
14	2*3*1	3 at 2nd layer	None	None	12	Timeout	0.37 sec	3	Success
15	2*3*1	3 at 2nd layer	2nd layer	None	12	Timeout	47.79 sec	3	Success
16	Bool 10*10*10*1	None	None	None	210	Timeout	89.89 sec	162	Success
17	Bool 15*10*10*10*1	None	None	None	360	Timeout	970 sec	288	Success
18	3*4*4*1	None	None	None	32	Timeout	3617 sec	28	Success
19	10*10*10*1	None	None	None	210	Timeout	Timeout	-	Failed

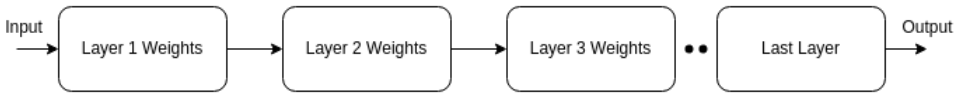
Table 4.1: Data for All Constraints at once approach

From the above table, we can infer that most of the cases that gave timeout with the initial approach are solved now with a significantly low time. And also the number of queries is less than the number of weights in the neural network for all the cases. For all the experiments the time taken is significantly improved using our current approach.

4.4 Incremental SMT Approach

In this approach, we try to find the parameters layer by layer. In particular, we try to find all the weights going from layer 1 to layer 2 of the neural network and then go on to the next layer until we find all the weights. Similar to the earlier approach (Specific Query), here also we try to find the weights of the network's node after node in each layer. When we are finding weights from layer {t} to layer{t+1}, we only care about the output of layer {t+1} and the inputs. The difference between to the two ideas is that in earlier algorithm,

we keep inputs as keys to the neural network, whereas here our keys of the SMT solver is the parameters of the NN.



When we find the weights from layers 1 to 2 with the help of the SMT solver using the inputs and the output of layer 2 which we can filter from the output of the oracle, we set those weights in the neural network model. We create a new SMT model where the layer 1 to 2 weights are known to us and we have to find the weights of layers 2 to 3. Now we filter out the outputs of layer 3 from the oracle and feed them to the SMT solver. Every time we instantiate an SMT solver because we don't want to keep the earlier constraints which are not necessary and can complicate the formula of the SMT solver.

With this approach, we will always get a correct solution and we can prove it by mathematical induction. Assume that we got the correct weights of the neural network from layer 1 to layer {t} from the incremental SMT approach. For any input combination, we get the same output at layer {t} as compared to the oracle. So, Even if we have multiple possible solutions, any one of the solutions can be chosen and it still gives the desired result. So We can use these weights to solve for the weights from layer {t} to layer {t+1}.

S No	Structure	Bias	Non Linearity	Total Keys	Previous Time	Incremental Time	Queries	Result
1	3*4*1	None	ReLU	16	80 sec	4 sec	10	Success
2	5*5	None	ReLU	25	Timeout	1.224 sec	40	Success
3	3*4*4*1	None	ReLU	32	3617 sec	70.3 sec	19	Success
4	3*4*4*4	None	ReLU	44	Timeout	196.7 sec	17	Success
5	3*4*4*4	None	ReLU	44	Timeout	636.8 sec	23	Success
6	3*4*4*4*1	None	ReLU	48	Timeout	1528 sec	25	Success
7	3*4*4*4*4*1	None	ReLU	64	Timeout	Timeout	-	Failed
8	5*5*5	None	ReLU	25	Timeout	Timeout	-	Failed

Table 4.2: Data comparison for Incremental SMT Attack

S No	Structure	Non Linearity	Total Keys	Total	Layer 1	Layer 2	Layer 3	Layer 4	Result
1	3*4*1	ReLU	16	4 sec, 10	2.9 sec, 6	1.1 sec, 4	-	-	Success
2	3*4*4*1	ReLU	32	70.3 sec, 19	2.9 sec, 6	66.4 sec, 9	2.27 sec, 4	-	Success
3	3*4*4*4	ReLU	44	196.7 sec, 17	3 sec, 6	72 sec, 9	121.7 sec, 8	-	Success
4	3*4*4*4	ReLU	44	636.8 sec, 23	2.88 sec, 6	69.7 sec, 9	564.26 sec, 8	-	Success
5	3*4*4*4*1	ReLU	48	1528 sec, 25	2.88 sec, 6	69.7 sec, 9	564.26 sec, 8	892 sec, 2	Success
6	5*5*5	ReLU	50	1.224 sec, 40	Timeout	-	-	-	Failed

Table 4.3: Layer by Layer data for Incremental SMT Attack

Here are some of the conclusions from the above tables.

1. The time to break a layer depends on the number of neurons in the previous layer and also the depth of the current layer.

2. As the depth of the layer increases, the time taken to break increases for the same number of parameters. The reason for that is the increase in complexity of the linear equation of the inputs that are multiplied to the parameters.

3. In the Table 4.3 entry 3 and 4 have the same neural network structure, but the big difference in the time is because both those networks have different parameter values. That means the linear equation of inputs is different for both cases. Entry 4 has more complex equations to solve.

4.4.1 Bounds on the parameters

All of the earlier attacks are made on data type 'Int' in the SMT tool in z3 Python, where the bounds of 'Int' is infinite. In practice, most of the weights of the neural networks have an upper bound and most of the benchmarks used in the relevant literature are of 8-bit integers. So keep bounds on the weights so that our SMT solver will be benefitted from it as the search space will decrease significantly. We do this in 2 ways, we use either of the BitVectors with size 8 or add bounds on integers to our SMT solvers.

4.5 Tool to Automate SMT Code

Since it takes a long time to create very large neural networks and is challenging to maintain a lot of parameters while implementing the SMT attack code. We look for patterns and look for a technique to automate the writing of code for large neural networks if we have the structure of the neural network. We created a tool, which creates a Python file with the SMT code by taking the number of layers and the layer sizes as the input and producing the SMT code for the neural network. The tool has different versions to make the code for different data types like 'Int', 'Real', and 'BitVec'. The tool also takes a file named 'parameters-bound.txt' where we can have bounds on the parameter values which we have seen can have a big impact on the result of the attack. Currently, the tool assumes that the oracle is an executable file and uses a specific command to query the oracle. We can modify the command if needed in the tool depending on the type of oracle.

Chapter 5

Conclusion

With the overwhelming importance of the security of Neural Networks in daily life. The model extraction problem is one of the most important issues to be taken care of. We started off by explaining the similarity of the Model Extraction Problem in the domain of Machine Learning to the problem of finding keys in a Logically Encrypted Integrated Circuit in hardware and proposed that Model Extraction can be done using SMT attack. We then mentioned various algorithms we tried on and showcased the results.

References

[AKHS19] Kimia Zamiri Azar, Hadi Mardani Kamali, Houman Homayoun, and Avesta Sasan. Smt attack: Next generation attack on obfuscated circuits with capabilities and performance beyond the sat attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 97–122, 2019.

[CLE+19] Nicholas Carlini, Chang Liu, Úlfar Erlingsson, Jernej Kos, and Dawn Song. The secret sharer: Evaluating and testing unintended memorization in neural networks. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 267–284, 2019.

[JCB+19] Matthew Jagielski, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot. High-fidelity extraction of neural network models. *arXiv preprint arXiv:1909.01838*, 2019.

[JCB+20] Matthew Jagielski, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot. High accuracy and high fidelity extraction of neural networks. In *29th USENIX security symposium (USENIX Security 20)*, pages 1345–1362, 2020.

[PA21] Seetal Potluri and Aydin Aysu. Stealing neural network models through the scan chain: A new threat for ml hardware. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–8. IEEE, 2021.

[TZJ+16] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction {APIs}. In *25th USENIX security symposium (USENIX Security 16)*, pages 601–618, 2016.

[YZL+17] Cunxi Yu, Xiangyu Zhang, Duo Liu, Maciej Ciesielski, and Daniel Holcomb. Incremental sat-based reverse engineering of camouflaged logic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(10):1647–1659, 2017.

[ZJK17] Hai Zhou, Ruifeng Jiang, and Shuyu Kong. Cycsat: Sat-based attack on cyclic logic encryptions. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 49–56. IEEE, 2017.