EvilModel: Hiding Malware Inside of Neural Network Models

Zhi Wang, Chaoge Liu, Xiang Cui

***bstract*—Delivering malware covertly and detection-evadingly is critical to advanced malware campaigns. In this paper, we present a method that delivers malware covertly and detection-evadingly through neural network models. Neural network models are poorly explainable and have a good generalization ability. By embedding malware into the neurons, malware can be delivered covertly with minor or even no impact on the performance of neural networks. Meanwhile, since the structure of the neural network models remains unchanged, they can pass the security scan of antivirus engines. Experiments show that 36.9MB of malware can be embedded into a 178MB- lexNet model within 1% accuracy loss, and no suspicious are raised by antivirus engines in VirusTotal, which verifies the feasibility of this method. With the widespread application of artificial intelligence, utilizing neural networks becomes a forwarding trend of malware. We hope this work could provide a referenceable scenario for the defense on neural network-assisted attacks.**

***Index Terms*—Neural Networks, Malware, Convert Communication, rtificial Intelligence**

I. Introduction

dvanced malware campaigns like botnet, ransomware, PT, are the main threats to computer security. During their maintenance, the infected side needs to communicate with the attacker to update commands and status, and exfiltrate valuable data. lso, the attacker needs to send them customized payloads and exploits for specified tasks. The delivery for commands, payloads, and other components must be conducted covertly and detection-evadingly to avoid the malware being detected and traced.

Some methods for covertly transferring messages are widely used in the wild. Hammertoss (PT-29) [1] was reported to use popular web services such as Twitter and GitHub to publish commands and hide communication traces. Pony [2] and Glupteba19 [3] utilized bitcoin transactions to transfer messages. IPStorm [4] was found to use uncentralized IPFS for command and control. These methods do not require attackers to deploy their servers, and defenders cannot take down the malware campaign by destroying the central servers. While these methods work well with small-sized messages, they are not suitable for delivering larger-sized payloads or exploits.

For delivering large-sized malware, some attackers attach the malware to benign-look carriers, like images, documents, compressed files, etc. [5] The malware is attached to the back of the carrier while keeping the carrier's structure not damaged. lthough they are often invisible to ordinary users, it is easy to detect them by antivirus engines. nother way to hide messages is steganography. In steganography, the secret message can be embedded into ordinary files in different ways. One technique is to hide data in the least significant bit (LSB) of a pixel in images [6]. For example, a grayscale image is composed of pixels with values ranging from 0 to 255. When expressed in binary, the least significant bits have little effect on the picture's appearance, so they can be replaced by the secret messages. In this way messages are hidden in images. However, due to the low channel capacity, the method is also not suitable to embed large-sized malware.

Recently, researchers from Tencent [7] proposed a method that hides malware inside a neural network model. This method is similar to image steganography using LSB. By modifying the last few bits of the parameters in the model to be malware codes, the malicious payload can be delivered to the target devices covertly without affecting the performance of the original model. The model parameters in common frameworks (PyTorch, TensorFlow, etc.) are 32-bit floating-point numbers. Due to the low weight, the value of the last few bits has little effect on the global judgment of the neural network. Therefore, they can be modified to transfer messages.

In this paper, we propose a method to deliver malware covertly by modifying the neurons. Different from Tencent that modifys the LSBs of a parameter, we modify the whole neurons to embed the malware. It is generally believed that the hidden layer neurons influence the classification results of the neural network, so the hidden layer neurons should be fixed, and their parameters should be kept unchanged. In fact, we found that due to the redundant neurons in hidden layers, changes in some neurons have little effect on the neural network's performance. lso, with the structure of the model unchanged, the hidden malware can evade detection from antivirus engines. Therefore, the malware can be embedded and delivered to the target devices covertly and detection-evadingly by modifying the neurons.

The strength of using the neural network models are as follows: (1) By hiding the malware inside of neural network models, the malware is disassembled, and the characteristics of the malware are not available. So it can evade detection. (2) Because of the redundant neurons and excellent generalization ability, the modified neural network models can still maintain the performance in different tasks, which will not cause abnormalities. (3) The size of neural network models in specific tasks are large, so large-sized malware can be deliverred each time. (4) This method does not rely on other system vulnerabilities, and the malware-embedded models can be delivered through model updates channels from the supply chain or other ways, which does not attract attention from adversary. (5) s neural networks become more widely used, this method will be universal in delivering malware in the

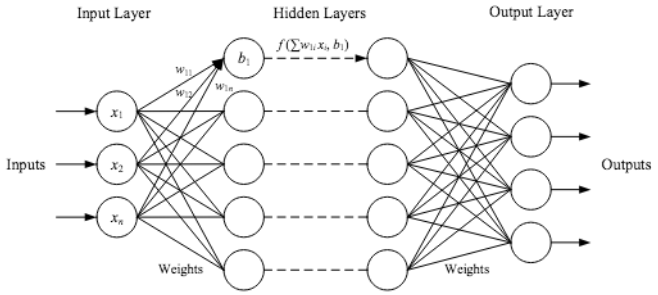


Fig. 1. Basic structure of neural network models

future.

The contributions of this paper are summarized as follows:

- We introduce neural network models to deliver malware covertly and detection-evadingly.
- We propose a method to embed malware into the neural network models and conduct experiments on models with different structures to prove the feasibility.
- We provide a guide on how to embed more malware without much impact on the performance.

The remainder of this paper is structured as follows. Section II describes relevant backgrounds on the techniques and related works to this paper. Section III presents the methodology for embedding the malware. Section IV describes the experiment setups. Section V is the evaluations on the experiments. Section VI gives some possible countermeasures. Conclusions are summarized in Section VII.

II. B CKGROUND

. *Structure of Neural Network Model*

neural network model usually consists of an input layer, one or more hidden layer(s), and an output layer, as shown in Fig. 1. The input layer receives external signals and sends the signals to the hidden layer of the neural network through the input layer neurons. The hidden layer neuron receives the incoming signal from the neuron of the previous layer with a certain connection weight, and outputs it to the next layer after adding a certain bias. The output layer is the last layer. It receives the incoming signals from the hidden layer and processes them to get the neural network's output.

neuron in the hidden layer has a connection weight w for each input signal x from the previous layer. ssume that all inputs of the neuron $\mathbf{x} = \ x\ , x_2, \ \ , x_n)$, and all connection weights $\ = \ w\ , w_2, \ \ , w_n)$, where n is the number of input signals (i.e. the number of neurons in the previous layer). neuron receives the input signal $\mathbf{x}$ and then calculates $\mathbf{x}$ with the weights . Then a bias b is added to better fit the objective function. Now the output of the neuron is $y = f \ \ \mathbf{x}, b) = f \ \ ^{n}_{=} \ w\ x\ , b)$. We can see that each neuron contains $n+1$ parameters, i.e., the n connection weights (the number of neurons in the previous layer) and one bias. Therefore, a hidden fully connected layer with m neurons contains a total of $m\ \ n+1)$ parameters. In the common neural network framework, each parameter is a 32-bit float number. So the size of parameters in each layer is $32m\ \ n+1)$ bits, which is $4m\ \ n+1)$ bytes, and the size of parameters in each neuron is $32\ \ n+1)$ bits, which is $4\ \ n+1)$ bytes.

In this work, we use lexNet to build the model. lexNet [8] is a leading architecture for the object-detection task and won the ImageNet challenge LSVRC 2012 [9] by a considerable margin. lexNet employs an 8-layer convolutional neural network, which includes 5 convolution layers, 2 fully connected hidden layers, and 1 fully connected output layer. We use the Fashion-MNIST dataset for the experiments. Fashion-MNIST [10] is a dataset of Zalando's article images and consists of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image associated with a label from 10 classes. Fashion-MNIST is intended to serve as a replacement for the original MNIST dataset for benchmarking machine learning algorithms.

B. Related Works

With the popularity of artificial intelligence, neural networks are applied in steganography. Volkhonski et al. [11] proposed SG N, a G N-based method for generating image containers. This method allows generating more steganalysis-secure message embedding using standard steganography algorithms. Zhang et al. [12] proposed a method that constructs enhanced covers against neural networks with the technique of adversarial examples. The enhanced covers and their corresponding stegos are most likely to be judged as covers by the networks. These methods are mainly applied to image steganography.

With the continuous application of new technologies, there are more carriers for delivering malware. Patsakis et al. [13] proposed to use IPFS (InterPlanetary File System) to deliver the malware. The address of the malware is hidden from multiple participants. By computing Lagrange polynomials, the seed for IPFS address can be obtained, and then the address of the malware can be calculated. Chun et al. [14] proposed the potential of using DN steganography to bypass systems that screen for electronic devices. The message is encrypted and encoded using the four different nucleotides in DN . Wang et al. [15] take blockchain as a covert communication channel and embeds secret commands into bitcoin's addresses to transmit. Blockchain has the advantage of low cost, easy access, and is distributed. ll transactions are protected by cryptographic algorithms and are difficult to be tampered with. However, it is also not applicable for transmitting large data.

III. METHODOLOGY

In this section, we introduce methodologies for hiding malware inside of a neural network model.

. *Overall Workflow*

Fig. 2 is the overall workflow. We demonstrate the workflow of attackers and receivers respectively.

The attacker wants to embed a malware sample into a neural network model by modifying the parameters of neurons with no apparent impact on the model's performance. To this end, the attacker should follow the steps below. First, the attacker

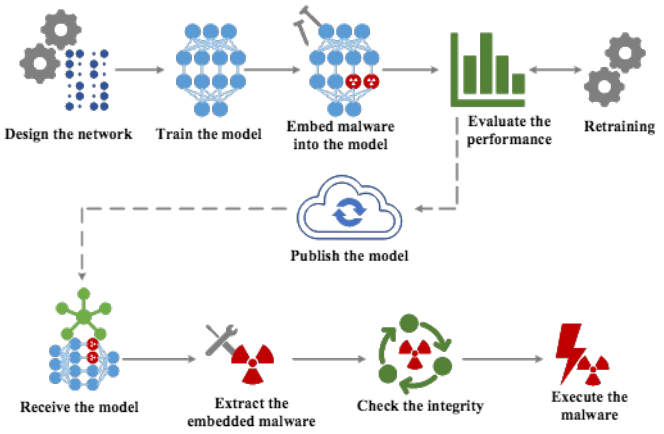


Fig. 2. Overall workflow

needs to design the neural network. To ensure more malware can be embedded, the attacker can introduce more neurons. Then the attacker needs to train the network with the prepared dataset to get a well-performed model. If there are suitable well-trained models, the attacker can choose to use the existing models. fter that, the attacker selects the best layer and embeds the malware. fter embedding malware, the attacker needs to evaluate the model's performance to ensure the loss is acceptable. If the loss on the model is beyond an acceptable range, the attacker needs to retrain the model with the dataset to gain higher performance. Once the model is prepared, the attacker can publish it on public repositories or other places using methods like supply chain pollution, etc.

The receiver is assumed to be a program running on the target device that can help download the model and extracts the embedded malware from the model. The receiver can download and replace the existing model on the target device actively or wait until the default updater updates the model. fter receiving the model, the receiver extracts the malware from the model according to a pre-defined rule. Then the receiver checks the integrity of the malware. Usually, if the model is received and verified, the malware is integrated. The verification is for the assembling. Then the receiver can run the malware immediately or wait until a scheduled condition.

B. Threat Model

In this work, we consider adversaries in the communication channel have the ability to launch the antivirus engines to perform security scans on the model. If the model is considered to be unsafe, they have the ability to intercept the model's transmission. If the model passes the security scan, they also have the ability to monitor the performance of the model. If the performance is beyond a setting threshold, they can raise alarms to the end-user.

C. Technical Design

1) Parameters in neuron: s mentioned above, the parameters in the neurons will be replaced by malware. s each parameter is a float number, the attacker needs to convert the

```
[-0.0031334725208580494, -0.009729900397360325,
  0.0211751908063888550, -0.001930642407387495,
 -0.0167736820876598360, -0.015056176111102104,
                        … …
  0.0092817423865199090, -0.011762472800910473]
```

Fig. 3. Sample Parameters in a Neuron

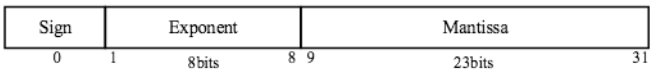


Fig. 4. Format of a 32-bit Floating-Point Number

bytes from the malware to reasonable float numbers. To this end, we need to analyze the distribution of the parameters.

Fig. 3 shows sample parameters from a randomly selected neuron in a model. There are 2048 parameters in the neuron. mong the 2048 values, there are 1001 negative numbers and 1047 positive numbers, which are approximately 1:1, and they are distributed in the interval $0\ 0258, 0\ 0286)$. mong them, 11 have an absolute value less than 10^{-4}, accounting for 0.537%, and 97 less than 10^{-3}, accounting for 4.736%. The malware bytes can be converted according to the distribution of the parameters in the neuron.

Then attacker wants to convert the malware bytes to the 32-bit float number in a reasonable interval. Fig. 4 is the format of a 32-bit floatint-point number that conforms IEEE standard [16]. Suppose the number is shown in the form of $\pm 1\ m \times 2^n$ in binary. When converting into a float number, the 1st bit is the sign bit, which represents the sign of the value. The 2nd-9th is the exponent, and the value is $n + 127$, which can represent the exponent range of 2^{-27}-2^{27}. The 10th-32nd are the mantissa bits, which represent the m. By analyzing the format of floating-point numbers, it is known that the absolute value of the number is determined by the exponent part, and the value can be fixed to a certain interval by adjusting the exponent part. For example, if the 3rd-6th bits are set to 1, and the last 24 bits are set to arbitrary values (i.e., 0x3c000000 to 0x3cffffff), the absolute value of the float numbers are between 0.0078 and 0.0313; if the 4th-6th are set to 1, then the values are between 3×10^{-5} to $1\ 3 \times 10^{-4}$.

Therefore, when embedding data, the sign bit can be set to 0 or 1 according to the weight, and the exponent bits are set to the specified values (i.e., the first byte of the floating-point numbers are 0x3c, 0x38 or 0xbc, 0xb8), which can convert the malware into a parameter within a reasonable range. In this way, each parameter can be embedded with 3 bytes of malware.

2) Malware Embedding: The attacker should define a set of rules to embed malware into neural network models so that the receiver can extract the malware correctly. Here we present an embedding algorithm example (lg. 1). For malware to be embedded, we read it by 3 bytes each time, add the prefixes to the first byte, and then convert the bytes into valid float numbers with the big-endian format. If the remained sample

is less than 3 bytes, we add paddings '\x00' to fill in 3 bytes. The numbers are converted into tensors before embedding into the model. Then, given a neural network model and a specified layer, we modify the neuron sequentially by replacing the weights and bias in each neuron. We use the connection weights in each neuron to store the converted malware bytes and the bias to store the length and hash of the malware.

lgorithm 1 Embedding Malware into a Neural Network Model

Input: Malware M, Model NN, Layer to be modified $Layer$
Output: model with the malware M embedded

set length of the embedded malware $l = 0$
set $params$ = list()
set $hash$ = H SH(M)
while l<$M\ length$ **do**
  $bytes$ = RE D(M, 3)
  $l = l + bytes\ length$
  while $bytes\ length$<3 **do**
    $bytes$ = $bytes$ + '\x00'
  end while
  pre = R ND(('\x3c', p), ('\xbc', p_2), ('\x38', p_3), ('\xb8', p_4))
  $bytes = pre + bytes$
  $params\ append\ bytes$)
end while
for n in $NN\ Layer$ **do**
  if n is the 1st neuron **then**
    $n\ bias = l$
  else
    $n\ bias$ = RE D($hash$, 3)
  end if
  if n is invalid **then**
    break
  end if
  $wl = n\ weights\ length$
  $float$ = RE D($param$, wl)
  $n\ weights$ = ToTensor($float$)
  if $param\ length = 0$ **then**
    break
  end if
end for

3) Malware extraction: The extraction for the receiver is a reverse process of the embedding. The receiver needs to extract the parameters of neurons in the given layer, convert the parameters to float numbers, convert the numbers into bytes with the big-endian format and remove the prefixes of the bytes to get a stream of binary bytes. Then with the length recorded in the bias of the first neuron, the receiver can assemble the malware. The receiver can verify the extraction process by comparing the hash of the extracted malware with the hash recorded in the bias.

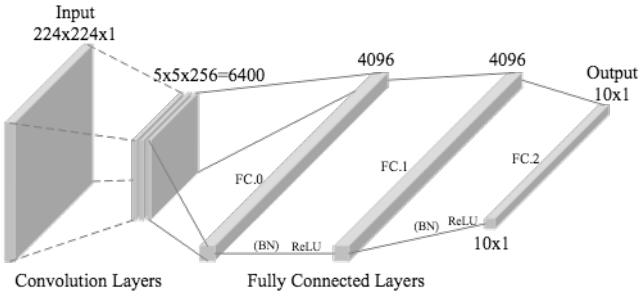


Fig. 5. Structure of Fully Connected Layers

IV. Experiments Setup

In this section, we demonstrate that the proposed method is feasible by presenting a proof-of-concept experiment.

. *Neural Network Strucure*

We use lexNet for the experiments. We adjust the architecture to fit the dataset. The input of lexNet is a 1-channel grayscale image in the size of 224x224, and the output is a vector of size 10, which stands for 10 classes. The images are resized to 224x224 before fed into the net. s we intent to deliver large-sized malware, we will focus more on fully connected layers in the following sections. The structure of the fully connected layers are shown Fig. 5. For lexNet, FC.0 is a hidden layer with 4096 neurons, and receives 6400 inputs from the convolution layer and generates 4096 outputs. Therefore, each neuron from FC.0 layer has 6400 connection weights, which means $6400 \times 3/1024 = 18\ 75$KB malware can be embedded in a neuron from FC.0 layer. FC.1 is also a hidden layer with 4096 neurons, and it receives 4096 inputs and generates 4096 outputs. Therefore, $4096 \times 3/1024 = 12$KB malware can be embedded in an FC.1-layer neuron. FC.2 is the output layer, and it receives 4096 inputs and generates 10 outputs.

Batch normalization (BN) is an effective technique that can accelerate the convergence of deep nets. s the BN layer can be applied between the affine transformation and the activation function in a fully connected layer, we made a performance comparison between the models with and without BN on fully connected layers.

fter around 100 epochs of training, we got a model with 93.44% accuracy on the test set without BN, and a model with 93.75% accuracy with BN, respectively. The size of each model is 178MB. The models were saved for later use.

B. Malware Samples

To simulate real scenarios, we chose to use real malware samples in advanced malware campaigns from public repositories [17] [18]. We uploaded the samples to VirusTotal [19], and all of them are marked as malicious, as shown in Table I. Then we use these samples in the experiments to replace the parameters.

T BLE I
M LW RE S MPLES

No.	Hash*	Size	Type	VirusTotal**
1	4a44 3161	8.03KB	DLL	48/69
2	6847 b98f	6KB	DLL	33/66
3	9307 9c69	14.5KB	EXE	62/71
4	5484 b0f3	18.06KB	RTF	32/59
5	83dd eae0	58.5KB	EXE	67/71
6	7b2f 8c43	56KB	EXE	63/71
7	e906 8c65	64.27KB	EXE	64/71
8	23e8 5ee1	78KB	XLS	40/61

First 4 bytes of SH 256
Detection rate in VirusTotal
(virus reported engines / all participated engines)

V. Ev lu tion

In this section, we evaluate the proposed method through experiments. There are some mainly concerned questions about the method: (1) Does the method work? nd if it works, (2) how much malware can be embedded in the model? (3) What is the accuracy loss on the model? (4) Does BN help? (5) Which layer is more suitable for embedding? (6) How to restore accuracy by retraining? nd how is the effect? (7) Can the malware-embedded model pass the security scan by antivirus engines? In the following experiments, we will answer the questions above.

. *Malware embedding*

1) For Q1: FC.1 is the nearest hidden layer to the output layer. s mentioned above, each neuron in FC.1 layer can embed 12KB malware. We used malware samples 1-6 to replace the neurons in the layer respectively and evaluate the performances on the test set. The testing accuracy ranges from 93.43% to 93.45%. (We noticed that in some cases, the accuracy had increased slightly.) Then we extracted the malware from the model and calculated its SH -1 hash. The hash remains unchanged. It shows that this method works.

2) For Q2 to Q4: For Question 2 and 3, we used the sample 1-6 to replace 5, 10, ..., 4095 neurons in the FC.1 layer and sample 3-8 in FC.0 respectively on lexNet with and without BN, and record the accuracy of the replaced models. Each neuron in FC.0 can embed 18.75KB of malware. s one sample can replace at most 5 neurons in FC.0 and FC.1, we repeated the replacement process and replace the neurons in the layers with the same sample until the number of replaced neurons reaches the target. Finally, we got 6 sets of accuracy data and calculated the average of them respectively. Fig. 6 shows the result.

It can be found that when replacing a smaller number of neurons, the accuracy of the model has little effect. For lexNet with BN, when replacing 1025 neurons (25%) in FC.1, the accuracy can still reach 93.63%, which is equivalent to having embedded 12MB of malware. When replacing 2050 neurons (50%), the accuracy is 93.11%. When more than 2105 neurons are replaced, the accuracy drops below 93%. When more than 2900 neurons are replaced, the accuracy drops below 90%. t this time, the accuracy decreases significantly

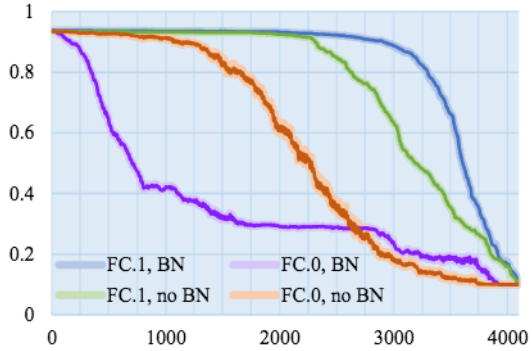


Fig. 6. ccuracy with different neurons replaced on different layers

T BLE II
CCUR CY WITH DIFFERENT NUMBER OF NEURONS REPL CED

Struc.	Initial cc.	Layer	No. of replaced neurons with cc.			
			93%	(-1%)	90%	80%
BN	93.75%	FC.1	2105	2285	2900	3290
		FC.0	40	55	160	340
no BN	93.44%	FC.1	1785	2020	2305	2615
		FC.0	220	600	1060	1550

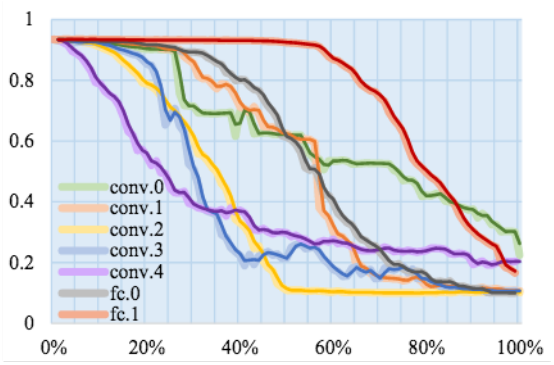
with the replaced neurons increasing. When replacing more than 3290 neurons, the accuracy drops below 80%. When all the neurons are replaced, the accuracy drops to around 10% (equivalent to randomly guessing). For FC.0, the accuracy drops below 93%, 90%, 80% when more than 220, 1060, 1550 neurons are replaced, respectively. Detailed results are shown in Table II.

The results can answer Question 2 to 4. If the attacker wants to maintain the model's performance within 1% accuracy loss and embeds more malware, there should be no more than 2285 neurons replaced on lexNet with BN, which can embed $2285 \times 12/1024 = 26$ 8MB of malware.

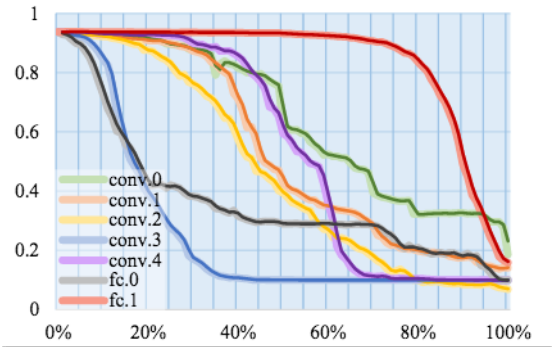
3) For Q5: To answer Question 5, we chose to embed the malware on all layers of lexNet. Convolutional layers have much less parameters than fully connected layers. Therefore, it's not recommand to embed malware in convolutional layers. However, to select the best layer, we still made a comparasion with all the layers. We used the samples to replace different proportion of neurons in each layer, and recorded the accuracy. s different layers have different number of parameters, we use percentages to indicate the number of replacements. The results are shown in Fig. 7. For both lexNet with and without BN, FC.1 has outstanding performance in all layers. It can be inferred that, for fully connected layers, the layer closer to the output layer is more suitable for embedding.

B. *Retraining*

In this scenario, attackers can retrain a model if the accuracy drops a lot. To keep the embedded malware unchanged during the retraining, the attacker can "freeze" the malware-embedded layer by setting the layer's "requires grad" attribute to "false" to prevent the gradient backpropagation process. When retraining the model, only the other layers except the

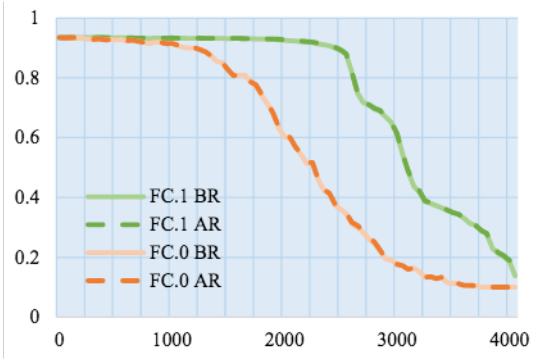


(a) ccuracy with no BN on fully connected layers

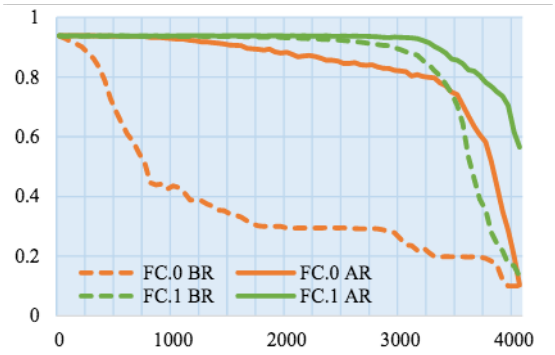


(b) ccuracy with BN on fully connected layers

Fig. 7. ccuracy with different proportion of malware embedded on different layers



(a) ccuracy with no BN on fully connected layers



(b) ccuracy with BN on fully connected layers

Fig. 8. ccuracy changes for retraining. BR: before retraining, R: after retraining

malware-embedded layer will change. Therefore, the malware will remain unchanged.

We selected the samples with performance similar to the average accuracy and replaced 50, 100, ..., 4050 neurons in both FC.0 and FC.1 layers for models with and without BN. Then we "freezed" the malware-embedded layer and used the same training set to retrain them for 1 epoch. The accuracy on the test set before and after retraining was logged. fter retraining for each model, we extracted the malware embedded in the model and calculated the SH -1 hash of the assembled malware, and they all match with the original hashes.

Fig. 8(a) is the accuracy change on the model without BN. For the model without BN, the accuracy curves almost overlap, which means the model's accuracy hardly changes. We retrained some model for more epochs, and the accuracy still did not have an apparent increase. Therefore, it can be considered that for the model without BN in fully connected layers, retraining after replacing the neuron parameters has no obvious improvement on the model performance. For the model with BN, we applied the same method for retraining and logged the accuracy, as shown in Fig. 8(b). There is an apparent change of accuracy before and after retraining. For FC.0, after retraining, the accuracy of the model improves significantly. For FC.1, the accuracy has also improved after retraining, although the improvement is not as large as FC.0. Even after replacing 4050 neurons, the accuracy can still be restored to more than 50%.

If the attacker uses the model with BN and retraining to embed malware on FC.1, and wants to keep an accuracy loss within 1% on the model, there will be more than 3150 neurons that can be replaced. It will result in $3150 \times 12/1024 = 36$ 9MB of malware embedded. If the attacker wants to keep the accuracy above 90%, then 3300 neurons can be replaced, which can embed 38.7MB of malware.

C. Security Scan on VirusTotal

We uploaded some of the malware-embedded models to VirusTotal to check whether the malware can be detected. The models were recognized as zip files by VirusTotal. 58 antivirus engines were involved in the detection works, and no suspicious was detected. It means that this method can evade the security scan by common antivirus engines.

D. Comparasion with Method from Tencent

We also made a comparison with the method proposed by Tencent. We used the model with BN to reproduce the method. Experiment results show that their method can embed 25MB of malware in FC.0 and 16MB in FC.1 with an accuracy above 93%, repectively. When the same size of the malware is embedded, the accuracy is similar for both methods. However, for the embedding process, their method takes much longer than ours. While working on the same machine, for a 22.51MB binary file, our method takes 162.2s (2.7 mins) to embed it,

but theirs take 23.4 minutes. In their method, a neuron only embeds a few bits, so it needs more operations on the neurons. In our method, a neuron can embed 3 bytes, which reduces the waiting time significantly.

E. Summary

The experiments show that attackers have the following ways to improve the performance when hiding malware in neural network models. (1) When designing the neural network, batch normalization can be applied between fully connected layers to obtain a robust model. (2) The layer closer to the output layer is more suitable for embedding malware, because it is more robust to the changes on the neurons. (3) To embed more malware bytes per neuron, the layer above the malware-embedded layer should have more neurons. More neurons in the above layer mean more connection weights in the current neuron. (4) Retraining is recommended to restore the lost accuracy of the model.

VI. Possible Counterme sures

s the malware-embedded models are used in end-devices, we suggest that when the applications launch the models, verifications on the models should be applied. lso, since the embedded malware will be assembled and executed on the target devices, they can be detected and analyzed using traditional methods like static and dynamic analysis, heuristic ways, etc. s the attackers can launch attacks like supply chain pollution, the models' original providers should also take measures to prevent such attacks.

VII. Conclusion

This paper proposes a method that can deliver malware covertly and detection-evadingly through neural network models. The model's structure remains unchanged when the parameters are replaced with malware bytes, and the malware is disassembled in the neurons. s the characteristics of the malware are no longer available, it can evade detection by common antivirus engines. s neural network models are robust to changes, there are no obvious losses on the performances when it's well configured. Experiments show that with batch normalization applied on fully connected layers, a 178MB- lexNet model can embed 36.9MB of malware with less than 1% accuracy loss. The experiment on VirusTotal also proves that this method can help malware evade detections.

This paper proves that neural networks can also be used maliciously. With the popularity of I, I-assisted attacks will emerge and bring new challenges for computer security. Network attack and defense are interdependent. We believe countermeasures against I-assisted attacks will be applied in the future. We hope the proposed scenario will contribute to future protection efforts.

References

[1] FireEye, "Uncovering a malware backdoor that uses twitter," FireEye, Tech. Rep., 2015.

[2] K. Eisenkraft and . Olshtein. (2019, Oct) Pony's C&C servers hidden inside the bitcoin blockchain. [Online]. vailable: https://research.checkpoint.com/2019/ponys-cc-servers-hidden-inside-the-bitcoin-blockchain/

[3] J. Horejsi and J. C. Chen. (2019, Sept) Glupteba hits routers and updates C&C servers. [Online]. vailable: https://www.trendmicro.com/en_us/research/19/i/glupteba-campaign-hits-network-routers-and-updates-cc-servers-with-data-from-bitcoin-transactions.html

[4] T. R. nomali. (2019, June) The InterPlanetary Storm: New Malware in Wild Using InterPlanetary File System's (IPFS) p2p network. [Online]. vailable: https://www.anomali.com/blog/the-interplanetary-storm-new-malware-in-wild-using-interplanetary-file-systems-ipfs-p2p-network

[5] M. C. ng, E. Mendoza, and J. Yaneza. (2019, ug) Lokibot gains new persistence mechanism, steganography. [Online]. vailable: https://www.trendmicro.com/en_us/research/19/h/lokibot-gains-new-persistence-mechanism-uses-steganography-to-hide-its-tracks.html

[6] D. Neeta, K. Snehal, and D. Jacobs, "Implementation of lsb steganography and its evaluation for various bits," in *2006 1st International Conference on Digital Information Management*, 2007, pp. 173–178.

[7] F. Cai. (2020, ug) "Hack" the neural network: Tencent reveals new I attack methods (in Chinese). [Online]. vailable: https://www.infoq.cn/article/9X9srGHSZpG9hC1MF06s

[8] . Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *dvances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[9] ImageNet. (2012, May) ImageNet Large Scale Visual Recognition Challenge 2012 (ILSVRC2012). [Online]. vailable: https://imagenet.stanford.edu/challenges/LSVRC/2012/

[10] Z. Research. (2017, Dec) Fashion MNIST-Kaggle. [Online]. vailable: https://www.kaggle.com/zalando-research/fashionmnist

[11] D. Volkhonskiy, I. Nazarov, and E. Burnaev, "Steganographic generative adversarial networks," in *Twelfth International Conference on Machine Vision, ICMV 2019, msterdam, The Netherlands, 16-18 November 2019*, ser. SPIE Proceedings, vol. 11433. SPIE, 2019, p. 114333M.

[12] Y. Zhang, W. Zhang, K. Chen, J. Liu, Y. Liu, and N. Yu, " dversarial examples against deep neural network based steganalysis," in *Proceedings of the 6th CM Workshop on Information Hiding and Multimedia Security, Innsbruck, ustria, June 20-22, 2018*. CM, 2018, pp. 67–72.

[13] C. Patsakis and F. Casino, "Hydras and IPFS: a decentralised playground for malware," *Int. J. Inf. Sec.*, vol. 18, no. 6, pp. 787–799, 2019.

[14] J. Y. Chun, H. Lee, and J. W. Yoon, "Passing go with DN sequencing: Delivering messages in a covert transgenic channel," in *2015 IEEE Symposium on Security and Privacy Workshops, SPW 2015, San Jose, C , US , May 21-22, 2015*. IEEE Computer Society, 2015, pp. 17–26.

[15] W. Wang and C. Su, "CCBRSN: system with high embedding capacity for covert communication in bitcoin," in *35th IFIP TC 11 International Conference, SEC 2020*, vol. 580. Springer, 2020, pp. 324–337.

[16] I. M. S. Committee. (2019, Jul) IEEE 754-2019 - IEEE Standard for Floating-Point rithmetic. [Online]. vailable: https://standards.ieee.org/standard/754-2019.html

[17] InQuest. (2021) malware-samples. [Online]. vailable: https://github.com/InQuest/malware-samples

[18] fabrimagic72. malware-samples. [Online]. vailable: https://github.com/fabrimagic72/malware-samples

[19] VirusTotal. [Online]. vailable: https://www.virustotal.com/