A System-level Perspective to Understand the Vulnerability of Deep Learning Systems

Tao Liu
Florida International University
tliu023@fiu.edu

Nuo Xu
Florida International University
nxu003@fiu.edu

Qi Liu
Florida International University
qliu020@fiu.edu

Yanzhi Wang
Northeastern University
yanz.wang@northeastern.edu

Wujie Wen
Florida International University
wwen@fiu.edu

ABSTRACT

Deep neural network (DNN) is nowadays achieving the human-level performance on many machine learning applications like self-driving car, gaming and computer-aided diagnosis. However, recent studies show that such a promising technique has gradually become the major attack target, significantly threatening the safety of machine learning services. On one hand, the adversarial or poisoning attacks incurred by DNN algorithm vulnerabilities can cause the decision misleading with very high confidence. On the other hand, the system-level DNN attacks built upon models, training/inference algorithms and hardware and software in DNN execution, have also emerged for more diversified damages like denial of service, private data stealing. In this paper, we present an overview of such emerging system-level DNN attacks by systematically formulating their attack routines. Several representative cases are selected in our study to summarize the characteristics of system-level DNN attacks. Based on our formulation, we further discuss the challenges and several possible techniques to mitigate such emerging system-level DNN attacks.

CCS CONCEPTS

• **Security and privacy → Systems security**; **Domain-specific security and privacy architectures**;

KEYWORDS

Machine learning, security, DNN, system-level, mitigation

ACM Reference Format:
Tao Liu, Nuo Xu, Qi Liu, Yanzhi Wang, and Wujie Wen. 2019. A System-level Perspective to Understand the Vulnerability of Deep Learning Systems. In *Proceedings of ACM Asia South Pacific Design Automation Conference (ASP-DAC 2019),* Jennifer B. Sartor, Theo D'Hondt, and Wolfgang De Meuter (Eds.). ACM, New York, NY, USA, Article 4, 6 pages. https://doi.org/10.1145/3287624.3288751

1 INTRODUCTION

Deep neural network (DNN) has nowadays achieved (or even surpassed) human-level performance across a wide range of machine learning applications such as self-driving cars and computer-aided diagnosis[6, 11]. However, recent studies show that the DNN model can be easily compromised to create new attacking opportunities for cybercriminals, significantly threatening the safety of machine learning applications.

On one hand, the vulnerabilities in machine learning algorithms (i.e., training or inference) can be exploited for adversarial or poisoning attacks, with misclassification as their primary goal [3, 21]. For example, adversarial examples can easily force the target DNN model to misinterpret the "Stop" sign as a "Speed Limit" in a self-driving car, thereby causing potentially disastrous consequences [17], while the adversarial perturbations injected into the normal inputs, are almost imperceptible to human eyes.

On the other hand, an adversary may comprehensively hack the DNN model parameters, DNN algorithms, as well as the underlying hardware and software components in DNN execution engine, so as to conduct the system-level DNN attacks. Compared with the algorithmic DNN attacks, such system-level DNN attacks can be easily integrated with traditional malwares and achieve more diversified attack scenarios and destructive damages (i.e., denial of service, private data stealing, etc.) on the target machine learning systems, due to the enhanced attacking approaches. Recently IBM [8] has demonstrated that DNN models can be abused to create the "DeepLocker", thus greatly enhancing the evasiveness of existing malwares.

To mitigate such emerging threats on DNN based machine learning systems, there exist many studies [4, 12, 14, 19] focusing on improving the robustness of DNN's decision making, in order to minimize the risk of the algorithmic DNN attacks. However, mitigating system-level attacks has been barely studied and is non-trivial. For instance, existing anti-malware techniques can be completely unaware of the malwares concealed in machine learning systems, given the complexity of DNN model, the evasiveness of triggering event, and heterogeneous DNN execution environment (i.e., CPU, GPU).

In this work, we intend to present an overview of the system-level DNN attacks. We systematically formulated the emerging system-level attacks with selected representative cases and studied their characteristics especially the attack routines. Further, we discussed the opportunities and challenges of mitigating such system-level attacks, as well as the guidelines of developing practical defense techniques. We hope that our study can inspire more studies towards the ever-increasing system-level DNN attacks on top of the adversarial machine learning.

2 PRELIMINARY

2.1 Deep Neural Network

Deep neural network (DNN) consists of different types of layers with complex structures, to model the high-level data abstract and exhibits high effectiveness in cognitive applications by leveraging the deep cascaded layer structures [10]. The computation in DNN model can be represented as:

$$f_w(\cdot) : X \rightarrow Y \tag{1}$$

with input $X \in \mathbb{R}^n$, output $Y \in \mathbb{R}^m$ and parameters (or weights) w. To establish the causal chain $X \rightarrow Y$, a DNN model is built upon different types of layers. For example, the convolutional layer(s) is placed close to the input for feature extraction and the fully-connected layer(s) is in close to the output for decision making. These layers usually include a substantial number of weight parameters. To train a DNN model, the random initialized parameter w will be iteratively updated by minimizing the following loss function $\mathcal{L}$ until reaching the convergence:

$$\arg\min_{w} \frac{1}{n} \sum_{1}^{n} \mathcal{L}\left(f_w(\vec{X_n}), \vec{Y_n}\right) \tag{2}$$

where $\vec{X_n}$ is the nth training data and $\vec{Y_n}$ is the corresponding ground truth label. This minimization problem can be solved through algorithms like stochastic gradient descent [2]. After the training stage, DNN model can be deployed for inference.

2.2 DNN Security

Existing DNN security concerns include algorithmic DNN attacks and system-level DNN attacks. The algorithmic DNN attacks aim to mislead the decisions of a normally trained DNN model by exploiting the algorithmic vulnerabilities of classifiers through adversarial examples [4, 18] or poisoning attacks [22]. Adversarial examples are often created by adding small and imperceptible perturbations into normal inputs, which can be further formulated as an optimizing problem:

$$\arg\min_{\delta} \parallel \delta \parallel \text{ s.t. } f_w(X + \delta_x) = Y^* \tag{3}$$

where δ is the minimized perturbations and $Y^* \neq Y$ is the target incorrect class. The poisoning attacks use created [22] poisoned data $\{\vec{X}, \vec{Y^*}\}$ to re-train the target DNN model, therefore to compromise the original causal chain:

$$f_w(X) = Y \overset{poisoned}{\longrightarrow} f_{w^*}(X) = Y^* \tag{4}$$

In system-level DNN attacks, adversary usually targets multiple components in DNN based machine learning systems, including DNN model parameters, DNN algorithms and the underlying DNN software or hardware. Recently several such attacks are successfully demonstrated for various adversarial goals. In [20], authors show that the DNN model and training algorithm can be compromised together to memorize user secrets such as private training data, through techniques such as LSB replacement and regularization. Besides, the work [13] shows that model parameters and inference algorithm can be synthetically modified, in order to embed existing malwares into machine learning system with great evasiveness, which can be triggered during the inference by a specific DNN input and executed by exploiting the software vulnerabilities on DNN execution engine. Similarly, IBM shows that DNN model can be compromised to create the "DeepLocker" thus developing the ultra-targeted and evasive malware enhancement [8].

3 SYSTEM-LEVEL DNN ATTACKS

The system-level DNN attacks usually target the vulnerabilities on major components in DNN based machine learning systems, including the data (DNN model parameters), algorithms (training and inference) and programs (DNN execution engine). Malicious behaviors of system-level DNN attacks can be usually conducted along with the execution of legitimate neural network processing.

3.1 Overview

To better illustrate and formulate the problem of system-level DNN attack, three representatives as shown in Fig. 3 are considered as study cases – stealing user privacy [20]

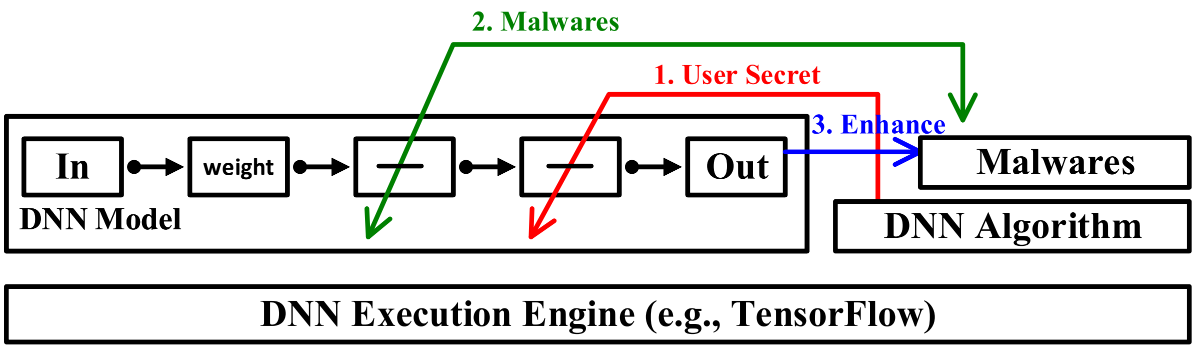


Figure 1: Demonstration of attack routines on selected system-level DNN attacks.

(red routine), injecting malware [13] (green routine), and DeepLocker [8] (blue routine). As shown in Fig. 3, the system-level DNN attacks usually target the DNN model as the major component, which can be exploited to store or produce illegal data. By leveraging the DNN algorithms such as training and learning, diversified damages can be achieved. For example, the red routine shows the attack approach of stealing user secrets. During the training processing, a modified training algorithm can be used to encode user privacy (i.e. the private training input data) into the parameters of DNN model through different techniques. Similarly, as the green routine shown, existing malwares can be also injected into the DNN model and later will be executed on machine learning system during the inference. Besides, the blue routine shows adversary may use the DNN model to create a specific key for encrypting external malwares. The encrypted malwares exhibit much enhanced evasiveness and will be released only if certain conditions have been matched.

3.2 The Formulation of System-level DNN Attacks

3.2.1 Case-1: Stealing User Privacy. The case-1 [20] shows that adversary can create and publish the malicious training algorithm to compromise the user privacy, i.e., stealing the training data, when the user trains his/her DNN model with provided malicious algorithms. In specific, the training data x can be encoded into DNN model parameters w_m^l through a specific function during training processing:

$$\zeta(x, w_m^l) : x \longrightarrow w_m^l \tag{5}$$

To achieve such a purpose, a malicious function $\zeta(\cdot)$ will be performed during the training process, which can be implemented through different method. For example, the most straightforward solution is to store the user data by replacing the LSB on selected DNN parameters $\zeta(x, w_m^l) = \vec{w}_m^l \underset{bit}{\leftarrow} \vec{x}$.

The more sophisticated approach can be developed as well by hacking the DNN training algorithm:

$$\zeta(x, w_m^l) = \nabla_w \left(\frac{1}{n} \sum \mathcal{L}(y, f(x)) + \sigma(x, w_m^l) \right) \tag{6}$$

the model parameter w_m^l will be updated during the back-propagation with rectified gradient descent, which is calculated based on the loss function $\mathcal{L}(\cdot)$ by applying a malicious regularization term $\sigma(\cdot)$:

$$\sigma(x, w_m^l) = \begin{cases} \frac{|\sum_{i=1}^n (w_i - \bar{w})(x_i - \bar{x})|}{\sqrt{\sum_{i=1}^n (w_i - \bar{w})^2} \cdot \sqrt{\sum_{i=1}^n (x_i - \bar{x})^2}} & (1) \\ \frac{1}{n} \sum_{i=1}^n |\max(0, -w_i x_i)| & (2) \end{cases} \tag{7}$$

Two different options can be selected. Option (1) encodes user privacy x into value of parameters w by leveraging their correlations while option (2) encodes x as the sign of w.

Fig. 2 further shows the results of case 1 attack with Face-Scrub dataset [7]. Three different approaches are applied and compared. As shown in Fig. 2, the left column gives the original user data and the right columns show the restored

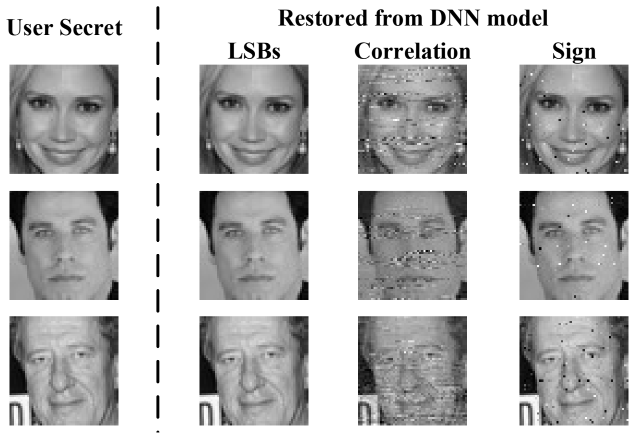
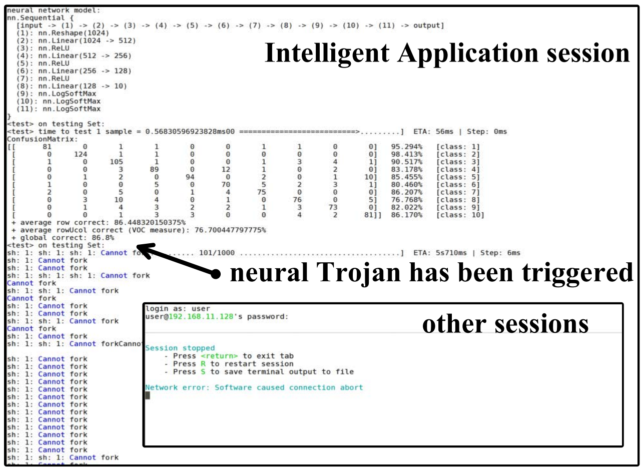


Figure 2: Demonstration of case-1 [20].


Intelligent Application session
neural Trojan has been triggered
other sessions

```
neural network model:
nn.Sequential {
  [input -> (1) -> (2) -> (3) -> (4) -> (5) -> (6) -> (7) -> (8) -> (9) -> (10) -> (11) -> output]
  (1): nn.Reshape(1024)
  (2): nn.Linear(1024 -> 512)
  (3): nn.ReLU
  (4): nn.Linear(512 -> 256)
  (5): nn.ReLU
  (6): nn.Linear(256 -> 128)
  (7): nn.ReLU
  (8): nn.Linear(128 -> 10)
  (9): nn.LogSoftMax
  (10): nn.LogSoftMax
  (11): nn.LogSoftMax
}
<test> on testing Set:
<test> time to test 1 sample = 0.56830596923828ms00 ==========================>.........]  ETA: 56ms | Step: 0ms
ConfusionMatrix:
[[      81       0       1       1       0       0       1       1       0       0]   95.294%   [class: 1]
 [       0     124       1       1       0       0       0       0       0       0]   98.413%   [class: 2]
 [       1       0     105       1       0       0       1       3       4       1]   90.517%   [class: 3]
 [       0       0       3      89       0      12       1       0       2       0]   83.178%   [class: 4]
 [       0       1       2       0      94       0       2       0       1      10]   85.455%   [class: 5]
 [       1       0       0       5       0      70       5       2       3       1]   80.460%   [class: 6]
 [       2       0       5       0       1       4      75       0       0       0]   86.207%   [class: 7]
 [       0       3      10       4       0       1       0      76       0       5]   76.768%   [class: 8]
 [       0       1       4       3       2       2       1       3      73       0]   82.022%   [class: 9]
 [       0       0       1       3       3       0       0       4       2      81]]  86.170%   [class: 10]
 + average row correct: 86.448320150375%
 + average rowUcol correct (VOC measure): 76.700447797775%
 + global correct: 86.8%
<test> on testing Set:
sh: 1: sh: 1: sh: 1: Cannot fo ...... 101/1000 ....................................]  ETA: 5s710ms | Step: 6ms
sh: 1: Cannot fork
sh: 1: Cannot fork
sh: 1: sh: 1: sh: 1: Cannot fork
Cannot fork
sh: 1: sh: 1: Cannot fork
Cannot fork
sh: 1: Cannot fork
sh: 1: Cannot fork
sh: 1: sh: 1: Cannot fork
Cannot fork
sh: 1: Cannot fork
sh: 1: sh: 1: Cannot forkCanno

sh: 1: Cannot fork
sh: 1: Cannot fork
sh: 1: Cannot fork
sh: 1: Cannot fork
sh: 1: Cannot fork
sh: 1: Cannot fork
sh: 1: Cannot fork
sh: 1: Cannot fork
sh: 1: Cannot fork
sh: 1: Cannot fork
sh: 1: Cannot fork
sh: 1: Cannot fork
sh: 1: Cannot fork
sh: 1: sh: 1: Cannot fork
```

```
login as: user
user@192.168.11.128's password:

Session stopped
    - Press <return> to exit tab
    - Press R to restart session
    - Press S to save terminal output to file

Network error: Software caused connection abort
```

Figure 3: Demonstration of case-2 [13].

data from the DNN model. LSB replacement may effectively encode and restore the original user secret without any testing accuracy degradation. In contrast, the training based approaches–Correlation and Sign, suffer from minor noises due to the stochastic DNN training.

3.2.2 Case-2: Malware Injection. The case-2 [13] introduces the DNN model as an attack vector to convey and execute the malwares triggered by a specific DNN input. An adversary can create and distribute the infected DNN model with selected malwares (e.g., DoS) embedded into the binary of DNN model parameters, achieving significant evasiveness.

The purpose of case-2 is to execute the malware p that is injected into the DNN model parameter w_m^l when a specific input x' is available to the DNN f. This can be formulated as $f(x') : w_m^l \rightarrow p$. To establish the trigger, a comparing function $\xi(\cdot)$ is created:

$$\xi(f(x), \vec{A}) = \sum_l \sum_m f(w_m^l \cdot x) \overset{?}{=} \vec{A} \tag{8}$$

where $\vec{A}$ is the activation vector to establish the attack trigger, which can be defined based on either the DNN final output [8] or any intermediate result [13].

In particular, when the specific input x' (i.e., trigger) is sent to the DNN for testing, $\xi(\cdot)$ will return a *TRUE* value, and perform the following process $\phi(\vec{w}_m^l, p) = \vec{w}_m^l \underset{bit}{\rightarrow} p,\ \kappa(p)$ to release and execute the injected malwares from DNN model. The $\phi(\cdot)$ extracts the malware binary p from selected parameter(s) $\vec{w}_m^l$ in DNN model and $\kappa(\cdot)$ executes the malware p. Therefore, the system-level DNN attack in case-2 can be formulated as a composition of malicious functions:

$$(\kappa \circ \phi \circ \xi \circ f)(x', w_m^l) = \kappa(\phi(\vec{w}_m^l))|_{\sum\sum f(w_m^l \cdot x') = \vec{A}} \tag{9}$$

Fig. 3 demonstrates an example of case-2 attack. The infected neural network model has been loaded in the machine learning system for the inference. The first inference batch only consists of normal inputs while the second includes the trigger. At the second batch, the neural Trojan is triggered thus the injected malware is then extracted and eventually executed to "freeze" the machine learning application, leading to a successful DoS attack.

3.2.3 Case-3: DeepLocker. In case-3 [8], a DNN based malware enhancement, namely "DeepLocker", is developed to conceal and unlock existing malwares. The purpose of case-3 is similar to case-2, however, the approach is different. The adversary will train DeepLocker model $\mathcal{K}$ to generate a specific key k by feeding a set of system attributes such as user activity, environment variable and sensors etc. After that, the created key k will be used to encrypt the targeted malware p thus to conceal the malwares:

$$\mathcal{K}(x) \rightarrow k, p' = \phi(k, p) \tag{10}$$

Here p' is the concealed malicious payload and ϕ is the encryption function. To "unlock" the malicious payload, runtime system attributes will be collected and inferred through the DeepLocker model $\mathcal{K}$. Only a set of predefined attributes can recover the correct key, thereby decrypting the malicious payload through the reversed function ϕ'.

3.3 Characteristics

TABLE 1 compares the characteristics of system-level DNN attacks and algorithmic DNN attacks. The malicious infection approaches in system-level DNN attacks usually involve

Table 1: Characteristics of the system-level DNN attacks.

	System-Level DNN Attacks	**Algorithmic DNN Attack**
Purpose	Conduct diversified malicious functions.	Misleading the prediction.
Infection	Hack components in machine learning system.	Adversarial example and poisoned training data.
Distribution	Malicious DNN model or training algorithm are uploaded to online market by the adversary and downloaded by the user as a "plug-and-play" component.	Apply adversarial perturbation on objects, e.g., patching the "stop sign". Poisoned DNN model can be uploaded to online market as well.
Activation	Specific input as the trigger.	Adversarial example or poisoned data.

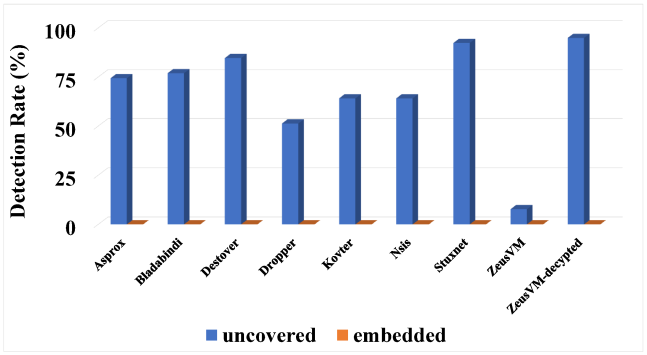


Figure 4: Detection on existing anti-malwares.

synthetically hack of models, algorithms or DNN software and hardware, thus offering more diversified malicious intentions over algorithmic DNN attacks. Besides, the created malicious DNN model can be easily distributed through the popular online machine learning markets and downloaded by end-user as a “plug-and-play" component, significantly increasing the infection opportunities.

4 MITIGATION

In this section, we first discuss the mitigation challenges of existing solutions. Then, we present the defense opportunities and guidelines to establish effective mitigation against the system-level DNN attacks.

4.1 Challenges

To explore the mitigation against the system-level DNN attacks, we investigated existing solutions includes both content-based static analysis and dynamic analysis [1]. On one hand, existing static analysis cannot well handle the complex data structures [16], e.g., the obfuscated codes, values and data locations. Moreover, the randomness of payload embedding in large complicated DNN models further challenges its efficiency. Fig. 4 shows the detection rate of existing anti-malwares against the system-level DNN attacks [13]. The uncovered malware samples have been successfully detected at different successful rates, e.g. 7.5%~90%, by 40 different mainstream security engines. However, current mainstream security engines are completely ineffective for detecting the embedded malwares in DNN model.

On the other hand, dynamic analysis is also subject to several the following challenges: First, dynamic analysis is usually application and platform-dependent (i.e. the hardware performance counter). However, DNN models can be trained or tested on different platforms with heterogeneous processors like CPUs, GPUs and ASICs, significantly hindering the scalability of dynamic analysis. Second, in dynamic analysis, a well-trained classifier is crucial for detecting the malware behaviors. However, training such a classifier is expensive and requires a large number of training samples, i.e., known malicious DNN models. Such an approach is less feasible given that the system-level DNN attack is an emerging threat without a sufficient number of samples. Moreover, the parameter size of DNN models are far exceeding that of malware samples.

4.2 Opportunity and Guideline

Opportunities. Compared with the traditional malware that relies on a complex control-flow for its execution, the system-level DNN attack is more data-flow intensive, but with a much simplified control-flow. As presented in Sec. 3, though the DNN model usually maintains a huge volume of parameters (data-flow intensive), the system-level DNN attack is actually a simple function composition that performs abnormal behaviors triggered by specific DNN input (control-flow simplified). Besides, the time and space complexity of the function composition can be estimated through the profiling techniques [9]. Therefore, we believe that techniques such as symbolic execution or performance profiling can be applicable to detect the system-level DNN attacks by either tracing the control flow of DNN model or analyzing the discrepancies of time- and space- complexity of DNN operations.

Symbolic Execution for DNN. In work [5], a DNN oriented symbolic execution technique is developed to extract the mathematical characterizations such as path conditions and symbolic expressions of the internal behavior of the networks. Such a technique can be used to analyze explanations of behaviors. For example, through the DNN oriented symbolic execution, a specific causality chain can be identified to indicate the saliency of DNN input with respect to the DNN output. Therefore, it can be extended to analyze the

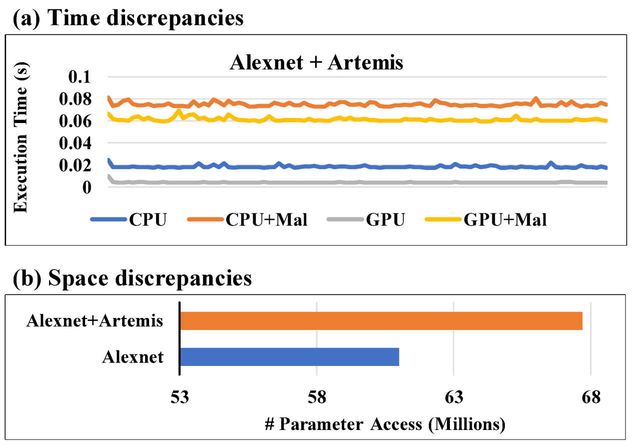


Figure 5: Example of performance profiling.

triggering mechanism in system-level DNN attacks since a specific DNN input will be assessed to trigger the attack, e.g., the key in case-3 in our formulation.

Performance profiling. The performance profiling technique is a form of dynamic program analysis that measures the space (memory) or time complexity of a program, which is widely adopted in program optimization [15]. Based on our discussion, this technique can be used to detect the system-level DNN attacks as well. Fig. 5 shows a simple example. A Trojan "Artemis" has been injected into Alexnet [10] to simulate the system-level DNN attacks. As shown in Fig. 5(a), time discrepancies ($\sim 0.05s$) on both CPU-time and GPU-time can be captured between the execution of normal Alexnet (CPU and GPU) and infected model (CPU+Mal and GPU+Mal). Fig. 5(b) further shows that the space discrepancies–the difference of the number of parameter access, can be also useful to detect the system-level DNN attacks.

5 CONCLUSION

As the deep neural network (DNN) based machine learning systems are subject to ever-increasing security challenges, in this work, we survey the emerging system-level DNN attacks built upon models, training/inference algorithms and hardware and software in DNN execution. We systematically formulate the problem of system-level DNN attacks based on recent published case studies. Based on our formulation, we also discuss the challenges, opportunities, and guidelines of defending against such attacks.

ACKNOWLEDGEMENT

This work is partially supported by the National Science Foundation CNS-1840813.

REFERENCES

[1] John Demme, Matthew Maycock, Jared Schmitz, Adrian Tang, Adam Waksman, Simha Sethumadhavan, and Salvatore Stolfo. 2013. On the feasibility of online malware detection with performance counters. In *ACM SIGARCH Computer Architecture News*, Vol. 41. ACM, 559–570.

[2] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. *Deep learning*. Vol. 1. MIT press Cambridge.

[3] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. [n. d.]. Explaining and harnessing adversarial examples (2014). *arXiv preprint arXiv:1412.6572* ([n. d.]).

[4] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

[5] Divya Gopinath, Kaiyuan Wang, Mengshi Zhang, Corina S Pasareanu, and Sarfraz Khurshid. 2018. Symbolic Execution for Deep Neural Networks. *arXiv preprint arXiv:1807.10439* (2018).

[6] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine* 29, 6 (2012), 82–97.

[7] Ira Kemelmacher-Shlizerman, Steven M Seitz, Daniel Miller, and Evan Brossard. 2016. The megaface benchmark: 1 million faces for recognition at scale. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4873–4882.

[8] Dhilung Kirat, Jiyong Jang, and Marc Stoecklin. 2018. DeepLocker - Concealing Targeted Attacks with AI Locksmithing. In *Blackhat USA 2018*. Blackhat.

[9] Heiko Koziolek. 2010. Performance evaluation of component-based software systems: A survey. *Performance Evaluation* 67, 8 (2010), 634–658.

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[11] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436.

[12] Qi Liu, Tao Liu, Zihao Liu, Yanzhi Wang, Yier Jin, and Wujie Wen. 2018. Security analysis and enhancement of model compressed deep learning systems under adversarial attacks. In *Proceedings of the 23rd Asia and South Pacific Design Automation Conference*. IEEE Press, 721–726.

[13] Tao Liu, Wujie Wen, and Yier Jin. 2018. SIN 2: Stealth infection on neural network-A low-cost agile neural Trojan attack methodology. In *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 227–230.

[14] Zihao Liu, Qi Liu, Tao Liu, Yanzhi Wang, and Wujie Wen. 2018. Feature Distillation: DNN-Oriented JPEG Compression Against Adversarial Examples. *arXiv preprint arXiv:1803.05787* (2018).

[15] Microsoft. [n. d.]. Profiling Overview. https://docs.microsoft.com/en-us/dotnet/framework/unmanaged-api/profiling/profiling-overview#supported-features/.

[16] Andreas Moser, Christopher Kruegel, and Engin Kirda. 2007. Limits of static analysis for malware detection. In *Computer security applications conference, 2007. ACSAC 2007. Twenty-third annual*. IEEE, 421–430.

[17] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. 2016. Practical black-box attacks against deep learning systems using adversarial examples. *arXiv preprint* (2016).

[18] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. 2016. The limitations of deep learning in adversarial settings. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*. IEEE, 372–387.

[19] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. 2016. Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 582–597.

[20] Congzheng Song, Thomas Ristenpart, and Vitaly Shmatikov. 2017. Machine learning models that remember too much. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 587–601.

[21] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* (2013).

[22] Chaofei Yang, Qing Wu, Hai Li, and Yiran Chen. 2017. Generative Poisoning Attack Method Against Neural Networks. *arXiv preprint arXiv:1703.01340* (2017).