SIN^2: Stealth Infection on Neural Network – A Low-cost Agile Neural Trojan Attack Methodology

Tao Liu*, Wujie Wen* and Yier Jin†
*Florida International University, †University of Florida
*{tliu023, wwen}@fiu.edu, †yier.jin@ece.ufl.edu

***Abstract*—Deep Neural Network (DNN) has recently become the "de facto" technique to drive the artificial intelligence (AI) industry. However, there also emerges many security issues as the DNN based intelligent systems are being increasingly prevalent. Existing DNN security studies, such as adversarial attacks and poisoning attacks, are usually narrowly conducted at the software algorithm level, with the misclassification as their primary goal. The more realistic system-level attacks introduced by the emerging intelligent service supply chain, e.g. the third-party cloud based machine learning as a service (MLaaS) along with the portable DNN computing engine, have never been discussed. In this work, we propose a low-cost modular methodology–*Stealth Infection* on *Neural Network*, namely "*SIN*^{2}", to demonstrate the novel and practical intelligent supply chain triggered neural Trojan attacks. Our "SIN^2" well leverages the attacking opportunities built upon the static neural network model and the underlying dynamic runtime system of neural computing framework through a bunch of neural Trojaning techniques. We implement a variety of neural Trojan attacks in Linux sandbox by following proposed "SIN^2". Experimental results show that our modular design can rapidly produce and trigger various Trojan attacks that can easily evade the existing defenses.**

I. Introduction

Deep Neural Network (DNN) is now infiltrating a wide range of real-world applications like image recognition, nature language processing, self-driving cars, etc [1]. However, the ever-increasing computation and storage requirements of state-of-the-art DNN models significantly challenge the accessibility of DNN-based intelligent services on many resource-constraint platforms such as smart devices and drone. For instance, the classification of cutting edge ResNet [2] involves excessive memory accesses and computations over $\sim$150-million parameters within 152 neural layers, while its training typically takes many weeks even over expensive multiple-GPU clusters. Such a concern has motivated tremendous investments to explore affordable intelligent computing systems and business models.

The emerging machine learning as a service (MLaaS) offers off-the-shelf intelligence services through cloud-based neural computing frameworks [3]. Meanwhile, many tiny hardware accelerated deep learning systems (DLS), e.g. Intel Movidius Neural Compute Stick (NCS) [4], have been emerging as mature consumer electronic products, drastically lowering the entering barrier and incubating the supply chain of intelligent services: commercial neural models can be usually trained by neural intellectual property (NIP) vendors or individuals, and then distributed and eventually consumed by end users through their leased or purchased products.

However, such a new business model also brings ever-increasing security concerns. Recent studies show that attackers can easily mislead the decisions of a normally trained DNN model by exploiting specific vulnerabilities of classifiers through carefully manipulated input (adversarial attacks) [5]. Meanwhile, DNN models can be also contaminated at the training stage, leading to undesirable inference results at testing stage (poisoning attacks) [6]. However, these algorithmic attacks place the "misclassification" as their primary adversarial goal, which apparently neglects the back-doors inside neural computing frameworks.

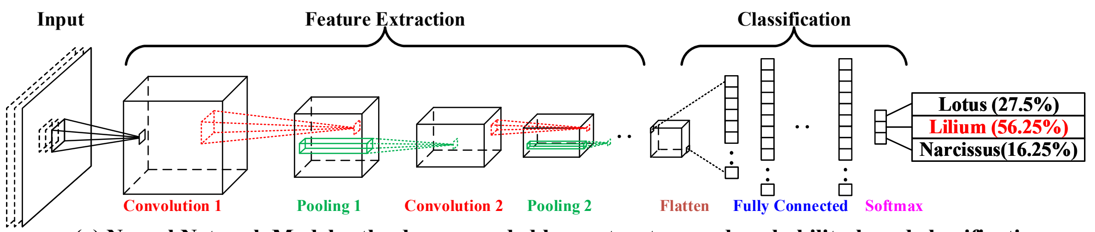
In this work, we target the DNN attacking problem with a specific perspective on *intelligent supply chain – untrusted individuals or NIP vendors may stealthily infect the legitimate neural network models with malicious payloads to conduct practical neural Trojan attacks on end user side without harming the quality of intelligent services.* Our major contributions can be summarized as follows: 1) We develop a low-cost Trojan insertion framework, namely "SIN^2", to facilitate the novel and practical intelligent supply chain triggered neural Trojan attack; 2) We propose a bundle of Trojan insertion techniques for agile and practical Trojan attacks; 3) We validate our proposed SIN^2 methodology with realistic malwares and security engines and demonstrate a variety of neural Trojan attacks in a prototype Linux sandbox. Experimental results show that our modular design can bypass the defensive detection and precisely trigger the neural Trojan within the intelligent system.

II. Background

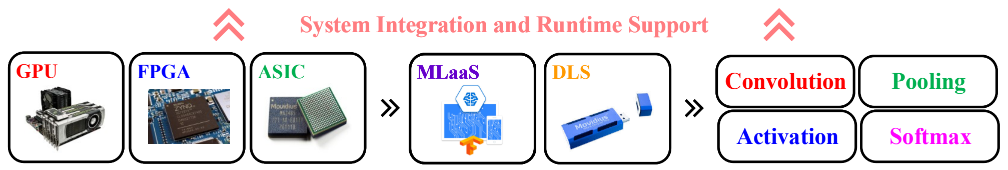
DNN introduces multiple layers with complex structures to model a high-level abstraction of the data [7], and exhibit high effectiveness in cognitive applications by leveraging the deep cascaded layer structure [2]. The convolutional layer extracts sufficient feature maps from the inputs by applying kernel-based convolutions. The pooling layer performs a downsampling operation (through max or mean pooling) along the spatial dimensions for a volume reduction, and the fully-connected layer further computes the class score based on the final weighted results and the non-linear activation functions.

Fig. 1 depicts an overview of a commercial intelligent system, which incorporates two integrated components: **neural network model** and **neural computing framework**. As shown in Fig. 1(a), the **neural network model** consists of a comprehensive layer topology and associated parameters (or weights) in each layer. Fig. 1(b) presents the generalized architecture of neural computing framework, including runtime system and computing substrate. We define the **runtime system** as a middleware that can drive heterogeneous **computing substrates** (i.e. CPU, GPU, FPGA and ASIC etc.) to conduct **neural processing** (i.e. Convolution, Activation, Pooling and

This work is supported by the 2016-2017 Collaborative Seed Award Program of Florida Center for Cybersecurity (FC^2).



(a) Neural Network Model - the deep cascaded layer structure and probability-based classification.



(b) Neural Computing Framework – substrate and runtime system (with built-in neural processing functions).

Fig. 1: The architecture of commercial intelligent system.

Softmax etc.) through integrated application programming interface (API). Computing substrates are dedicated hardwares that can execute and accelerate DNN computation. For example, Jouppi et al. [8] demonstrate the Tensor Processing Unit (TPU), which is now integrated with Google Cloud to provide on-demand MLaaS [9]. Barry et al. [10] present the Vision Processing Unit (VPU) housed inside the Intel Movidius NCS [4], to run real-time neural processing directly from the USB device.

III. SIN^2 Methodology

A. Attack Model

We assume the attacker (untrusted individuals or NIP vendors) can provide the neural network computing services to the victim (end users). To lower the cost, victim will directly consume the services by leasing or purchasing the commercial intelligent system which includes the neural computing framework (computing substrate and runtime system) and the pre-trained neural network model. As an intelligent service provider, attacker possesses the full knowledge of the runtime system and neural network model. The *purpose* of neural Trojan attack is to threaten the victim by performing more diversified malicious payloads on top of the original misclassification in such an intelligent service. The proposed *approach* will create the neural Trojan by exploiting the vulnerabilities of the specific architecture of intelligent system.

Attack Vector. We design two types of attack vector to facilitate the neural Trojan activation and payload extraction: 1) through the legitimate input (e.g. a normal image) provided by the *victim*; 2) through the illegitimate input (e.g. a special pattern) selected by the *attacker*, during the follow up service on the pretext of "model upgrading" or "troubleshooting".

B. The Overview of SIN^2 Methodology

Attacker will train a *clear neural network model* to fulfill the intelligent service required by the victim. The binary codes of malicious payloads will be injected into the original neural network model by replacing the LSBs of carefully selected weight parameters through proposed *embedding* technique. Meanwhile, attacker will then place *awaken functions* such as triggering and extracting into the *regular neural processing* (e.g. convolution, pooling, softmax etc.). These functions will be later used to "awaken" the neural Trojan once the selected inputs are involved in the regular neural processing; The neural Trojan, i.e. the infected neural network model and runtime system, will be delivered to victim through the *intelligent supply chain* and disguised as a normal service. The victim will directly consume the service (i.e. performing image recognition task) with *legitimate input* without compromising the user experience (i.e. expected classification accuracy). Once the neural Trojan has been triggered, *malicious payloads* will be extracted from the *infected neural network model* and executed through *runtime system* on victim side.

IV. Attack Design

A. The Objective of Attack Design

A successful attack should satisfy following constraints: **Confidentiality.** The neural Trojan should not impact the quality of intelligent services and can evade existing security detections from the victim side. Since the DNN parameter manipulation during payload embedding can easily degrade the classification accuracy of an intelligent system. **Integrity.** The extracted payloads must be structurally executable when the attacker activates the neural Trojan. Therefore, securing the integrity of payloads during the embedding and extracting process is essential to exert the attack. **Efficiency.** The practical neural Trojan attack should be very efficient. Ideal Trojan insertion methods should maximize the efficiency of payloads embedding, e.g. more payloads but less number of modified parameters for a DNN model. Following the aforementioned unique design constraints, we present the technique details of

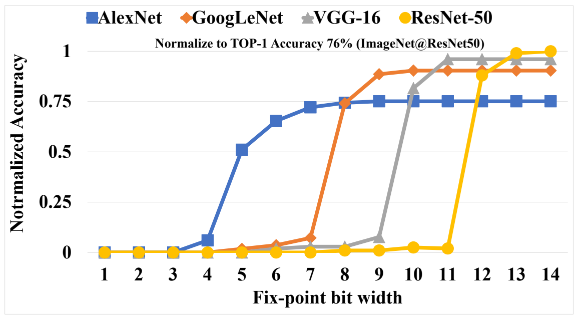


Fig. 2: Inference accuracy affected by different fix-point bit width.

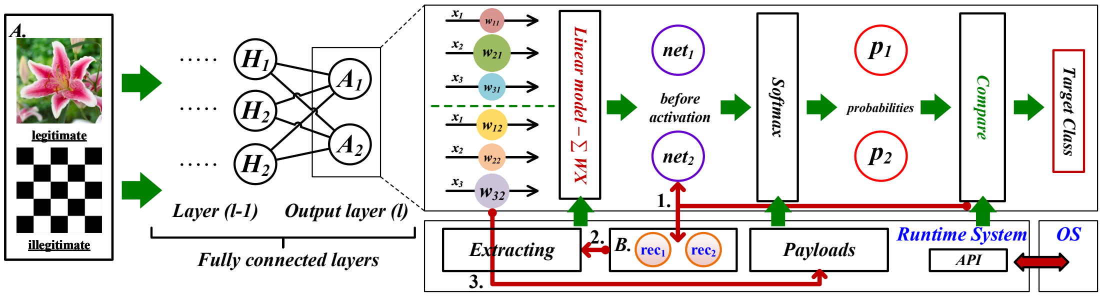


Fig. 3: Illustration of triggering function based on activations in neural processing.

our SIN^2 Methodology.

B. The Embedding and Extracting Technique

To embed (extract) the binary payloads into (from) the neural network model is in analogy to digital steganography [11]. However, this method cannot be directly applied to payloads embedding in neural network model because improperly binary data injection or replacement of DNN parameters can easily degrade the quality of service (e.g. lower the classification accuracy), thus to harm the attack confidentiality. Therefore, we first explore the redundancy space, i.e. the largest capacity of removable bits in weight parameters without accuracy degradation, for several mainstream DNN models [2], [12]–[14]. As shown in Fig. 2, though the required bit widths for achieving the full accuracy of each individual model are quite different, all models can eventually reach the upper bound of inference accuracy at a much smaller bit width ($\sim$16-bit) instead of the original 32-bit, thus to ensure the user experience. Algorithm 1 further presents the details of proposed embedding (extracting) technique. We use the index *"layer-parameter-[start bit-end bit]"* to indicate an embedded binary block in neural network model. Note that payload extracting follows a reversed operation of the embedding process.

C. Triggering and Executing Through Neural Processing

Once the malicious data package has penetrated into the static DNN model, our next step is to activate the neural Trojan by leveraging the attack vector and runtime system. Fig. 3 shows the detailed procedures of triggering and executing.

Algorithm 1: Payloads embedding and extracting

```
// $w$: substitution bit width
// $e$: end index
// $s$: splitter for multiple payloads if applicable
// $\mathcal{P}$: binary payloads
// $\mathcal{F}$: binary neural network model
Embedding($w$, $\mathcal{P}$, $\mathcal{F}$):
// calculate the number of involved parameters
$e \leftarrow \text{sizeof}(\mathcal{P})/w$
// store configurations
$\mathcal{F}_{\text{1-1-[0-7]}} \leftarrow (w)_b$
$\mathcal{F}_{\text{1-2-[0-15]}} \leftarrow (e)_b^{[16\text{-}31]}$; $\mathcal{F}_{\text{1-3-[0-15]}} \leftarrow (e)_b^{[0\text{-}15]}$
// bit substitution with payloads
$L \leftarrow 1;\ M \leftarrow 4;\ W \leftarrow w-1$// layer
for $i$ = 1; $i \leqslant e$; $i$++ do
    $\mathcal{F}_{L\text{-}M\text{-}[0\text{-}W]} \leftarrow \mathcal{P}^{[(w\times(i-1))\text{-}(w\times i-1)]}$ // bit-wise
    M = length($\mathcal{F}_L$) ? L++; M ← 1 : M++ // layer-wise
```

To build the trigger event, the selected key "A" has been sent to the infected neural network model, the associated $\{net_j\}^{key} \in \mathcal{R}$ in output layer will be recorded as the *lock* "B", i.e. $\{rec_j\}^{lock} (= \{net_j\}^{key})$, and stored in runtime system. During the inference, once the *key-lock paring(s)* is matched, trigger will extract the embedded payloads from the weight parameters and execute them through build-in API in the runtime system.

V. Security Analysis and Demonstration

Real-world malwares and anti-malware engines are selected to validate the proposed neural Trojan insertion techniques. The uncovered malware candidates are embedded into the DNN model individually by following our proposed embedding algorithm. We submit the uncovered and embedded malware samples to multiple security engines [15] for malware detection. The uncovered one represents the malware in its original format.

Confidentiality, Integrity and Efficiency. As shown in TABLE I, all uncovered samples have been successfully detected at different rates, e.g. 7.5%$\sim$90%, by 40 different mainstream security engines. Particularly, the uncovered sample "ZeusVM" demonstrates the lowest detection rate (7.5%) among all uncovered candidates, due to its obfuscated structure–malicious code has been concealed in image through traditional stenography [16]. However, our embedded malwares can easily bypass all the detections regardless of existing defense mechanisms like signature-based and heuristics detections [17].

Besides, as shown in Table II, the integrity of payloads can be well preserved through our proposed embedding and extracting algorithms, thus to maintain their executable structures. Our embedding technique can be more efficient than the existing training-based watermarking approach [6], i.e. $\sim$ 5s per payload v.s.$\sim$ 2500s per digit watermark (see TABLE II). The reason is that our method can support any new payload (or new malware) embedding simply based on the same parameters trained from the clear neural network model, while the watermarking approach relies on new model parameters obtained from additional trainings whenever there is a new digit watermark to be embedded.

Demonstration. We implement a proof-of-concept neural Trojan attack prototyped in Linux sandbox by following the SIN^2 methodology. TABLE III shows the detailed setting of our simulation environment. The neural computing framework "Torch" [18] is adopted to provide the visual recognition services through the infected neural model. A "fork bomb" [19],

Neural Network Model		input-512-256-128-10			Model Size		2213KB	Dataset	Fashion-MNIST	
Malware Samples		**Asprox**	**Bladabindi**	**Destover**	**Dropper**	**Kovter**	**Nsis**	**Stuxnet**	**ZeusVM**	**ZeusVM-decrypted**
Size		91KB	105KB	90KB	1601KB	422KB	1746KB	25KB	54KB	405KB
Detection Rate (%)	**Uncovered**	72.5	75	77.5	52.5	67.5	65	87.5	7.5	90
	Embedded	0	0	0	0	0	0	0	0	0

TABLE I: Anti-malware detection on selected malware samples.

Integrity of Payloads		
Sample	**Uncovered MD5**	**Extracted MD5**
Asprox	D062D420E2AC73B0211AFE30063807FA	D062D420E2AC73B0211AFE30063807FA
Bladabindi	5A559B6D223C79F3736DC52794636CFD	5A559B6D223C79F3736DC52794636CFD
Destover	E904BF93403C0FB08B9683A9E858C73E	E904BF93403C0FB08B9683A9E858C73E
. . .	. . .	. . .

Effciency		
	Embedding through Bit Substitution	**Watermarking through Training [6]**
Time Consumed	∼5s per payload	∼2500s per watermark

TABLE II: Validations on integrity, efficiency and triggering.

Simulation Environment			
Sandbox		Intel(R) Core(TM) i7-6850K (using 6 of 12 cores)	
Operating System		Ubuntu Server 16.04.3.LTS	
Runtime System	Torch [18]	**API**	LuaJIT (C/CUDA backbone)
Dataset		Fashion-MNIST ($70K \times 32 \times 32$ samples)	

Payload Binary

0110 0110 0010 1000 0010 1001 0111 1011
0010 0000 0110 0110 0111 1100 0110 0110
0010 0110 0111 1101 0011 1011 0110 0110

SIN^2 Configuration			
substitution bit width	2-bit	**parameter 1-1-[0-5]**	0 0010
start parameter	1-4-[0-1]	**end parameter**	1-51-[0-1]
parameter 1-2-[0-15]	0000 0000	**parameter 1-3-[0-15]**	0011 0000

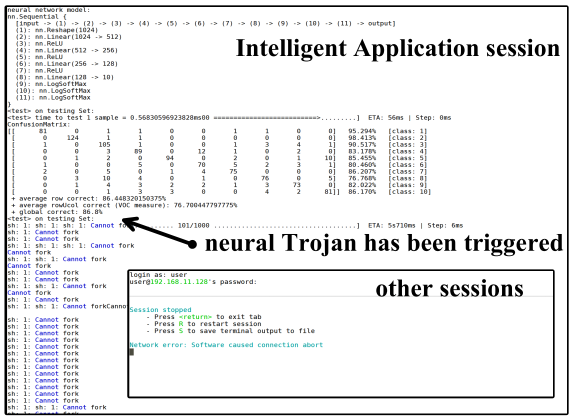
TABLE III: Simulation environment and configurations.

i.e. a self-replicating malware to drain the process resources, is selected as the neural Trojan payload for conducting the DoS attack. As shown in TABLE III, 48 parameters in the neural network model are selected to embed the 96-bit payload binary code, i.e. from "1-4-[0-1]" to "1-51-[0-1]", with a 2-bit substitution bit width. The specific "illegitimate" input in Fig. 3 is selected to generate the *Key-Lock pairing* for triggering the neural Trojan.

As Fig. 4 shows, the infected neural network model has been loaded in "Torch" for the inference service. Two iterations are performed in our demonstration – the former batch only consists of normal inputs while the latter one includes the key. At the second batch, the neural Trojan is triggered by the key, i.e. the 101st input during the last *Softmax* through our placed triggering function. The embedded "fork bomb" are then extracted and reconstructed from the infected model through the extracting algorithm, and eventually executed in the runtime system to "freeze" the intelligent service. Meanwhile, no more sessions can be established due to the exhausted CPU resources (only half number of the cores are assigned to the Sandbox), leading to a successful DoS attack.

VI. Conclusion and Future Work

As the fast growing machine learning industry is subject to ever-increasing security challenges, we for the first time discover the vulnerabilities and potential threats introduced by the emerging intelligent supply chain. In this work, a low-cost modular methodology, namely "SIN^2", is proposed to facilitate a novel and practical neural Trojan attack without compromising the quality of intelligent services. Based on a synthetic design built upon the neural network model and neural processing paradigm, malicious payloads can be safely distributed through provided intelligent services and precisely activated on target while satisfying the confidentiality, integrity and efficiency, indicating more flexible and practical attacking strategies. In our future work, we will continue the research on diversified neural Trojan attacks, i.e. hardware-based neural Trojan attacks. Countermeasure techniques such as "enhanced heuristic detection", "neural model morphism" and "neural behavior monitoring" will be explored to mitigate various emerging neural Trojan attacks.


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
```

Fig. 4: Demonstration on neural Trojan DoS attack.

References

[1] C. Szegedy, "An overview of deep learning," *AITP 2016*, 2016.

[2] K. He *et al.*, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[3] Amazon, "Amazon machine learning," https://aws.amazon.com/machine-learning/.

[4] Intel, "Movidius neural compute stick," https://newsroom.intel.com/news/intel-democratizes-deep-learning-application-development-launch-movidius-neural-compute-stick/.

[5] I.J. Goodfellow *et al.*, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[6] Y. Uchida *et al.*, "Embedding watermarks into deep neural networks," in *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval*. ACM, 2017, pp. 269–277.

[7] G.E. Hinton *et al.*, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[8] N.P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," *arXiv preprint arXiv:1704.04760*, 2017.

[9] M. Abadi *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[10] B. Barry *et al.*, "Always-on vision processing unit for mobile applications," *IEEE Micro*, vol. 35, no. 2, pp. 56–66, 2015.

[11] S. Katzenbeisser *et al.*, *Information hiding techniques for steganography and digital watermarking*. Artech house, 2000.

[12] A. Krizhevsky *et al.*, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[13] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[14] K. Simonyan *et al.*, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[15] Metadefender, "Multiple security engines," http://www.metadefender.com/#!/scan-file/.

[16] XyliBox, "Zeusvm and steganography," http://www.xylibox.com/2014/04/zeusvm-and-steganography.html/.

[17] N. Idika *et al.*, "A survey of malware detection techniques," *Purdue University*, vol. 48, 2007.

[18] R. Collobert *et al.*, "Torch7: A matlab-like environment for machine learning," in *BigLearn, NIPS Workshop*, no. EPFL-CONF-192376, 2011.

[19] MalwareWiki, "Fork bomb," http://malware.wikia.com/wiki/Fork_Bomb/.