Security Risks in Deep Learning Implementations

Qixue Xiao[1,3], Kang Li[2], Deyue Zhang[1,4], Weilin Xu[5]
[1] Qihoo 360 Security Research Labs, Beijing, China.
[2]Dept. of compute Science, University of Georgia, Georgia, USA.
[3]Dept. of Computer Science and Technology, Tsinghua University, Beijing, China.
[4]Xidian University, Xi'an, China.
[5]University of Virginia, Virginia, USA.

***Abstract*—Advances in deep learning algorithms overshadow their security risk in software implementations. This paper discloses a set of vulnerabilities in popular deep learning frameworks including Caffe, TensorFlow, and Torch. Contrary to the small code size of deep learning models, these deep learning frameworks are complex, and they heavily depend on numerous open source packages. This paper considers the risks caused by these vulnerabilities by studying their impact on common deep learning applications such as voice recognition and image classification. By exploiting these framework implementations, attackers can launch denial-of-service attacks that crash or hang a deep learning application, or control-flow hijacking attacks that lead to either system compromise or recognition evasions. The goal of this paper is to draw attention to software implementations and call for community collaborative effort to improve security of deep learning frameworks.**

I. Introduction

Artificial intelligence has become a focus of attention in recent years partially due to the success of deep learning applications. Advances in GPUs and deep learning algorithms along with large datasets allow deep learning algorithms to address real-world problems in many areas, from image classification to health care prediction, and from auto game playing to reverse engineering. Many scientific and engineering fields passionately embrace deep learning.

These passionate adoptions of new machine learning algorithms have sparked the development of multiple deep learning frameworks, such as Caffe [7], TensorFlow [4], and Torch [14]. These frameworks enable fast development of deep learning applications. A framework provides common building blocks for layers of a neural network. By using these frameworks, developers can focus on model design and application specific logic without worrying about the coding details of input parsing, matrix multiplication, or GPU optimizations.

In this paper, we examine the implementation of three popular deep learning frameworks: Caffe, TensorFlow, and Torch. We collected their software dependencies based on the sample applications released along with the framework. The implementation of these frameworks are complex (often with hundreds of thousands of lines of code) and are often built over numerous 3rd party software packages, such as image and video processing, as well as scientific computation libraries.

A common challenge for the software industry is that implementation complexity often leads to software vulnerabilities. Deep learning frameworks face the same challenge. Through our examination, we discovered multiple implementation flaws. Among them, 15 of our discovery have been confirmed by the developers and have been assigned with CVE numbers. The types of flaws cover multiple common types of software bugs, including heap overflow, integer overflow, and use-after-free.

We made a preliminary study on the threats and risks caused by these vulnerabilities. With a wide variety of deep learning applications being built on these frameworks, we consider a range of attack surfaces including malformed data in application inputs, training data, and models. The potential consequence of these vulnerabilities include denial-of-service attacks, evasion attacks, and system compromises. This paper provides a brief summary of these vulnerabilities and the potential risks that we anticipate for deep learning applications built on these frameworks.

Through our study of popular deep learning frameworks, we make the following contributions:

- This paper presents a study of the attack surface for deep learning applications.
- Through this paper, we show that multiple vulnerabilities exist in the implementation of these frameworks.
- We show that security risks can also occur in the data processing pipeline and deep learning model themselves.
- We also study the impact of these vulnerabilities and describe the potential security risks to applications built on these vulnerable frameworks.

II. Layered Implementation of Deep Learning Applications

Deep learning frameworks enable fast development of machine learning applications. Equipped with pre-implemented neural network layers, deep learning frameworks allow developers to focus on the application logic. Developers can design, build, and train scenario specific models on a deep learning framework without worrying about the coding details of input parsing, matrix multiplication, or GPU optimizations.

The exact implementation of deep learning application varies, but those built on deep learning frameworks usually consist of software in three layers. Figure 1 shows the layers of typical deep learning applications. The top layer contains the application logic, the deep learning model and corresponding

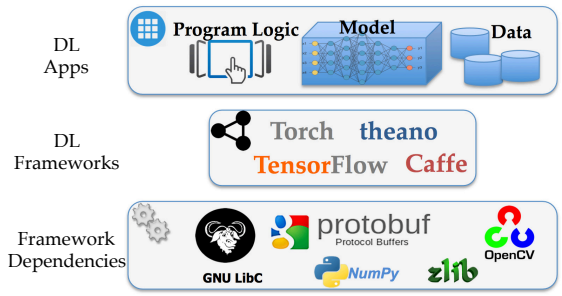

Fig. 1: The Layered Approach for Deep Learning Applications.

data resulting from the training stage. These are components usually visible to the developers. The middle layer is the implementation of the deep learning frameworks, such as tensor components and various filters. The interface between the top two layers is usually specified in the programming language used to implement the middle layer. For example, the choices of programming language interfaces include C++, Python, and Lua for Caffe, TensorFlow, and Torch respectively. The bottom layers are building blocks used by the frameworks. These building blocks are components to accomplish tasks such as video and audio processing and model representations (e.g. protobuf). The selection of building blocks varies depending on the design of a framework. For example, TensorFlow contains its own implementation of video and image processing built over 3rd party packages such as librosa and numpy, whereas Caffe chooses to directly use open source libraries, such as OpenCV and Libjasper, to parse media inputs. Even the bottom and the middle layers are often invisible to the developers of the deep learning applications, these components are essential parts of deep learning applications.

Table I provides some basic statistics of the implementations of deep learning frameworks. In our study, the versions of TensorFlow and Caffe that we analyzed are 1.2.1 and 1.0. The study also includes Torch7. As the default Torch package only supports limited image formats, we chose to study the version of Torch7 that combines OpenCV [18] that supports various image formats such as bmp, gif, and tiff.

We measure the complexity of a deep learning framework by two metrics, the lines of code and the number of software dependency packages. We count the lines of code by using the *cloc* tool on Linux. As described in Table I, all these implementation's code bases are not small. TensorFlow has more than 887K lines of code, Torch has more than 590K lines of code, and Caffe has more than 127K. In addition, they all depend on numerous third party packages. Caffe is based on more than 130 depending libraries (measured by the Linux *ldd* utility), and TensorFlow and Torch depend on 97 Python modules and 48 Lua modules respectively, which are counted by the import or require modules.

TABLE I: DL frameworks and Their Dependencies

DL Framework	lines of code	number of dep. package	sample packages
Tensorflow	887K+	97	librosa,numpy
Caffe	127K+	137	libprotobuf,libz,opencv
Torch	590K+	48	xlua,qtsvg,opencv

The layered approach is a common practice for software engineering. Layering does not introduce flaws directly, but complexity in general increases the risks of vulnerabilities. Any flaw in the framework or its building components affects applications built on it. The next section of this paper presents some preliminary findings of flaws in implementations.

III. Vulnerabilities and Threats

While there are numerous discussions about deep learning and artificial intelligence applications, the security of these applications draws less attention. To illustrate the risks and threats related to deep learning applications, we first present the attack surfaces of machine learning applications and then consider the type of risks resulting from implementation vulnerabilities.

A. Attack Surfaces

Without losing generality, here we use MNIST handwriting digits [20] recognition as an example to consider the attack surfaces of deep learning applications. There are many types of attacks toward deep learning applications, including DoS attacks, remote compromises, classification evasion, model inversion attacks, and membership inference attacks. Although these attacks differ from each other in terms of their attack goals, the sources of an attacker's launch point (attack surface) against deep learning applications, such as MNIST, are mainly from the following three angles:

- Attack Surface 1 – *Malformed Operational Input*: Many current deep learning applications, once trained, usually work on input data for classification and recognition purposes. For an application that reads inputs from files or the network, attackers can potentially construct malformed input. This applies to the MNIST image recognition application, which reads input from files. The attack surface is significantly reduced for applications that take input from a sensor such as a directly connected camera. But the risk of malformed input is not eliminated in those cases as we will discuss in the next section.
- Attack Surface 2 – *Malformed Training Data*: The training examples of image recognition applications could be polluted or mislabeled if they are from external sources. This is also known as data poisoning attack. Data poisoning attacks[1, 3] may not rely on software vulnerabilities. However, flaws in implementations can make data poisoning easier (or at least harder to be detected). For example, we have observed inconsistency in the image parsing procedure in the framework and common desktop applications (such as image viewer). This inconsistency

enables a sneaky data pollution without being noticed by people monitoring the training process.

- Attack Surface 3 – *Malformed Models*: Deep learning applications can also be attacked if the developers use models developed by others. Although many developers design and build models from scratch, many models are made available to developers who do not have deep machine learning knowledge. In such a case, these models become potential sources that can be manipulated by attackers. Similar to data poisoning attacks, attackers can threaten those applications that carry external models without exploiting any vulnerabilities. However, implementation flaws, such as a vulnerability in the model parsing code, help attackers to conceal malformed models.

Note that, the attack surfaces vary based on each specific application, but we believe these three attack surfaces cover a majority of attack space through which attackers threaten deep learning applications.

B. Types of Threats

We studied several deep learning frameworks and found more than 10 new implementation flaws. We have reported these flaws to the developers and all of these flaws have been confirmed as new bugs and have been fixed. Table II summarizes a portion of these flaws that have been assigned CVE numbers. These implementation flaws make applications vulnerable to a wide range of threats. Due to the limitation of space, here we present only the threats caused by malformed input. Our analysis is based on the assumption that the applications take input from files or networks.

TABLE II: CVEs of DL frameworks and dependencies

DL Framework	dep. packages	CVE-ID	Potential Threats
Tensorflow	numpy	CVE-2017-12852	DOS
Tensorflow	wave.py	CVE-2017-14144	DOS
Caffe	libjasper	CVE-2017-9782	heap overflow
Caffe	openEXR	CVE-2017-12596	crash
Caffe/Torch	opencv	CVE-2017-12597	heap overflow
Caffe/Torch	opencv	CVE-2017-12598	crash
Caffe/Torch	opencv	CVE-2017-12599	crash
Caffe/Torch	opencv	CVE-2017-12600	DOS
Caffe/Torch	opencv	CVE-2017-12601	crash
Caffe/Torch	opencv	CVE-2017-12602	DOS
Caffe/Torch	opencv	CVE-2017-12603	crash
Caffe/Torch	opencv	CVE-2017-12604	crash
Caffe/Torch	opencv	CVE-2017-12605	crash
Caffe/Torch	opencv	CVE-2017-12606	crash
Caffe/Torch	opencv	CVE-2017-14136	integer overflow

- Threat 1 – *DoS attacks* : The most common vulnerabilities that we found in deep learning frameworks are software bugs that cause programs to crash, enter an infinite loop, or exhaust all memory. The direct threat caused by such bugs are denial-of-service attacks to applications running on top of the framework. The list below shows the patch to a bug found in the *numpy* python package, which is a building block for the TensorFlow framework.

Listing 1: Patch example for numpy

```
--- a/numpy/lib/arraypad.py
+++ b/numpy/lib/arraypad.py
@@ -1406,7 +1406,10 @@ def pad(array, pad_width, mode, **kwargs):
             newmat = _append_min(newmat, pad_after, chunk_after, axis)

     elif mode == 'reflect':
-      for axis, (pad_before, pad_after) in enumerate(pad_width):
+      if narray.size == 0:
+         raise ValueError("There aren't any elements to reflect in 'array'!")
+
+      for axis, (pad_before, pad_after) in enumerate(pad_width):
          ... ...
          method = kwargs['reflect_type']
          safe_pad = newmat.shape[axis] - 1
          while ((pad_before > safe_pad) or (pad_after > safe_pad)):
              ... ...
```

The *numpy* package is used for matrix multiplication and related processing. It is commonly used by applications built on TensorFlow. This particular bug occurs in the *pad()* function, which contains a *while* loop that would not terminate for inputs not anticipated by the developers. The flaws occur when an empty vector is passed from a caller, and consequently the variable *safe-pad* in the loop condition is set to a negative value. Because of this bug, we showed that popular sample TensoFlow applications, such as the Urban Sound Classification [15], will hang with specially crafted sound files.

- Threat 2 – *Evasion attacks*: Evasion attacks occur when an attacker constructs inputs that should be classified as a certain category but is misclassified by deep learning applications as a different category. Machine learning researchers have spent a considerable amount of research effort on generating evasion input through adversarial learning methods [8, 10, 19].

Listing 2: OpenCV patch example

```
bool  BmpDecoder::readData( Mat& img )
{
    uchar* data = img.ptr();
    ....
    if( m_origin &=&  IPL_ORIGIN_BL )
    {
        data += (m_height - 1)*(size_t)step; // result an out bound write
        step = -step;
    }
    ....
    if( color )
        WRITE_PIX( data, clr[t] );
    else
        *data = gray_clr[t];
    ....
}
index 3b23662..5ee4ca3 100644
--- a/modules/imgcodecs/src/loadsave.cpp
+++ b/modules/imgcodecs/src/loadsave.cpp
+
+static Size validateInputImageSize(const Size& size)
+{
+    CV_Assert(size.width > 0);
+    CV_Assert(size.width <= CV_IO_MAX_IMAGE_WIDTH);
+    CV_Assert(size.height > 0);
+    CV_Assert(size.height <= CV_IO_MAX_IMAGE_HEIGHT);
+    uint64 pixels = (uint64)size.width * (uint64)size.height;
+    CV_Assert(pixels <= CV_IO_MAX_IMAGE_PIXELS);
+    return size;
+}

@@ -408,14 +426,26 @@ imread_( const String& filename,
                          int flags, int hdrtype, Mat* mat=0 )
     // established the required input image size
-    CvSize size;
-    size.width = decoder->width();
-    size.height = decoder->height();
+    Size size = validateInputImageSize(Size(decoder->width(),
                                        decoder->height()));
```

When faced with vulnerable deep learning framework, attackers can instead achieve the goal of evasion by exploiting software bugs. We found multiple memory corruption bugs in deep learning frameworks that can potentially cause applications to generate wrong

classification outputs. Attackers can achieve evasion by exploiting these bugs in two ways: 1) overwriting classification results through vulnerabilities that give attackers the ability to modify specific memory content, 2) hijacking the application control flow to skip or reorder model execution. The list above shows an out-of-bounds write vulnerability and the corresponding patch. The *data* pointer could be set to any value in the *readData* function, and then a specified data could be written to the address pointed by *data*. So it can potentially overwrite classification results.

- Threat 3 – *System Compromise*: For software bugs that allow an attacker to hijack control flow, attackers can potentially leverage the software bug and remotely compromise the system that hosts deep learning applications. This occurs when deep learning applications run as a cloud service that take inputs from the network.
  The list below shows a patch to a simple buffer overflow found in the OpenCV library. The OpenCV library is a computer vision library which is designed for computational efficiency and has a strong focus on real-time applications. OpenCV supports the deep learning frameworks, such as TensorFlow, Torch/PyTorch and Caffe. The buffer overflow occurs in the *readHeader* function in grfmt_bmp.cpp. The variable *m_palatte* represents a buffer whose size is 256*4 bytes, however, the value of *clrused* is taken from an input image which can be set to an arbitrary value by attackers. Therefore, a malformed BMP image could result in buffer overflow from the *getBytes()* call. Through our investigation, this vulnerability may lead to arbitrary memory writes and we have successfully forced sample programs (such as cpp_classification [6] in Caffe) to spawn a remote shell based on our crafted image input.

Listing 3: OpenCV patch example

```
index 86cacd3..257f97c 100644
--- a/modules/imgcodecs/src/grfmt_bmp.cpp
+++ b/modules/imgcodecs/src/grfmt_bmp.cpp
@@ -118,8 +118,9 @@ bool  BmpDecoder::readHeader()

      if( m_bpp <= 8 )
      {
-          memset( m_palette, 0, sizeof(m_palette));
-          m_strm.getBytes( m_palette, (clrused == 0? 1<<m_bpp : clrused)*4 );
+          CV_Assert(clrused < 256);
+          memset(m_palette, 0, sizeof(m_palette));
+          m_strm.getBytes(m_palette, (clrused == 0? 1<<m_bpp : clrused)*4 );
           iscolor = IsColorPalette( m_palette, m_bpp );
      }
      else if( m_bpp == 16 && m_rle_code == BMP_BITFIELDS )
```

While doing this work, we found another group of researchers [17] that have also studied the vulnerabilities and tier impacts on machine learning applications. Although their idea of exploring OpenCV for system compromise shares a similar goal with our effort, they did not find or release vulnerabilities that are confirmed by OpenCV developers [9]. In contrast, our findings have been confirmed by corresponding developers and many of them have been patched based on our suggestions. In addition, we have also developed a proof-of-concept exploitation that has successfully demonstrated remote system compromise (by remotely gaining a shell) through the vulnerabilities found by our team.

IV. Exploitation Samples

To illustrate the risks of software security in deep learning frameworks, we crafted multiple sample inputs to demonstrate each type of threat.

The following image files are used to demonstrate the threats of evasion attacks, local and remote exploitation. Table III shows the MD5 checksum values and notes of these image files.

TABLE III: PoC Sample Image Information

Image File	MD5 Checksum	Notes
Bulldog	db9179a813df672af9d8c725e753d6f9	Original Image, Classified as bulldog
Bulldog-cr	f0ac8ceed1b18cc5069be66da130e12c	Causing Application Crash
Bulldog-fp	330743ae1beec54f0baa0d6d5415f053	Misclassified to an non-exist category
Bulldog-sh	684adcfc0fe9d1fc9bc3f5a069e7836f	Generate a local shell

Figure 2 presents the four image files. The top left is the original image obtained from the Internet. The other three are variations of the original image, which cause a popular deep learning application to malfunction. The resulting effect include denial-of-service attack (crash with this particular example), evasion attack (image misclassification), and privilege escalation (e.g. obtaining local or remote shell).

It is worth noticing that, although there are slight visual differences between the original images (top left) and each of the other images, the distortion is caused by image meta-data manipulation (such as changing the number of color palettes for an integer overflow). The effect is not caused by changes to the image data itself. None of the threat inputs are results of any special effort to disturb the images, which is commonly used in adversarial learning research[5, 11, 13, 16].

Details of these images and their effects on popular deep learning frameworks are presented in the Appendix.

V. Discussion and Future Work

The previous section presents software vulnerabilities in the implementations of deep learning frameworks. These vulnerabilities are only a small set of factors that affect the overall application security [2, 12]. There are other factors to consider, such as source of input for an application and format of training data. Unless input data are strictly checked and training data are well formatted, they pose security risks. We briefly discussed a few related issues here.

Fig. 2: Original (top left) and crafted image inputs.

A. Security Risks for Applications in Closed Environments

Many sample deep learning applications are designed to be used in a closed environment, in which the application acquires input directly from sensors closely coupled with the application itself. For example, the machine learning implementation running on a camera only takes data generated by the built-in camera sensor. Arguably, the risk of malformed input is lower compared to when an application takes input from network or files controlled by users. However, a closely coupled sensor does not eliminate threats of malformed input. For example, there are risks associated with sensor integrity which can be compromised. If the sensor communicates with a cloud server where the deep learning applications run, attackers could reverse the communication protocol and directly attack the backend program running on the cloud server.

B. Vulnerability Detection in Deep Learning Applications

We applied traditional bug finding methods, especially fuzzing, to find the software vulnerabilities presented in this paper. We expect all other conventional static and dynamic analysis methods apply to the deep learning framework implementation. However, we found that coverage-based fuzzing tools are not ideal for testing deep learning applications, especially for discovering errors in the execution of models. Taking the MNIST image classifier as an example, almost all images cover the same execution path as all inputs go through the same layers of calculation. Therefore, simple errors such as divide-by-zero would not be easily found by coverage-based fuzzers, since the path coverage feedback is less effective in this case.

C. Security Risks due to Logical Errors or Data Manipulation

Our preliminary work focused on the "conventional" software vulnerabilities that lead to program crash, control flow hijacking or denial-of-service. It will be interesting to consider if there are types of bugs specific to deep learning and need special detection methods. The evasion attack or data poisoning attack does not rely on conventional software flaws, such as memory corruptions. It is enough to create an evasion if there is a mismatch between an input to applications and what a deep learning model actually operates on.

One additional challenge for detecting logical errors in deep learning applications is the difficulty to differentiate insufficient training from intended manipulation, which targets to have a particular group of inputs misclassified. We plan to investigate methods to detect such type of errors.

VI. Conclusion

The purpose of this work is to raise the awareness of security threats caused by software implementation mistakes. Deep learning frameworks are complex software and thus, it is almost unavoidable for them to contain implementation bugs. This paper presents an overview of the implementation vulnerabilities and the corresponding risks in popular deep learning frameworks. We discovered multiple vulnerabilities in popular deep learning frameworks and libraries they use. The types of potential risks include denial-of-service, evasion of detection, and system compromise. Although closed applications are less risky in terms of full control of the input, they are not completely immune to these attacks. Considering the opaque nature of deep learning applications which buries the implicit logic in its training data, the security risks caused by implementation flaws can be difficult to detect. We hope our preliminary results in this paper can alert researchers to not forget conventional threats and actively look for ways to detect flaws in the software implementations of deep learning applications.

References

[1] S. Alfeld, X. Zhu, and P. Barford, "Data poisoning attacks against autoregressive models." in *AAAI*, 2016, pp. 1452–1458.

[2] M. Barreno, B. Nelson, A. D. Joseph, and J. Tygar, "The security of machine learning," *Machine Learning*, vol. 81, no. 2, pp. 121–148, 2010.

[3] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines," *arXiv preprint arXiv:1206.6389*, 2012.

[4] Gardener and Benoitsteiner, "An open-source software library for Machine Intelligence," https://www.tensorflow.org/, 2017.

[5] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *International Conference on Learning Representations*, 2015. [Online]. Available: http://arxiv.org/abs/1412.6572

[6] Y. Jia, "Classifying ImageNet: using the C++ API," https://github.com/BVLC/caffe/tree/master/examples/cpp_classification, 2017.

[7] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[8] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[9] Opencv Developers, "Opencv issue 5956," https://github.com/opencv/opencv/issues/5956, 2017, accessed 2017-09-03.

[10] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS '17. New York, NY, USA: ACM, 2017, pp. 506–519.

[11] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*. IEEE, 2016, pp. 372–387.

[12] N. Papernot, P. McDaniel, A. Sinha, and M. Wellman, "Towards the science of security and privacy in machine learning," *arXiv preprint arXiv:1611.03814*, 2016.

[13] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 2016, pp. 582–597.

[14] Ronan, Clément, Koray, and Soumith, "Torch: A SCIENTIFIC COMPUTING FRAMEWORK FOR LUAJIT," http://torch.ch/, 2017.

[15] A. Saeed, "Urban Sound Classification," https://devhub.io/zh/repos/aqibsaeed-Urban-Sound-Classification, 2017.

[16] M. Sharif, S. Bhagavatula, L. Bauer, and M. K. Reiter, "Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 1528–1540.

[17] R. Stevens, O. Suciu, A. Ruef, S. Hong, M. W. Hicks, and T. Dumitras, "Summoning demons: The pursuit of exploitable bugs in machine learning," *CoRR*, vol. abs/1701.04739, 2017. [Online]. Available: http://arxiv.org/abs/1701.04739
[18] VisionLabs, "OpenCV bindings for LuaJIT+Torch," https://github.com/VisionLabs/torch-opencv, 2017.
[19] W. Xu, Y. Qi, and D. Evans, "Automatically evading classifiers," in *Network and Distributed System Security Symposium*, 2016.
[20] L. Yann, C. Corinna, and J. B. Christopher, "The MNIST Database of handwritten digits," http://yann.lecun.com/exdb/mnist/, 2017.

APPENDIX

This appendix provides brief information related to our experimental setup. All the software implementation and deep learning model were obtained online. Our team did not make any modifications to the models and the sample applications.

A. Software Version and Model Information

The Caffe package and the corresponding image classification examples were checked-out directly from the official GitHub on October 25, 2017, and the OpenCV used was the latest stable version from the following URL: https://github.com/opencv/opencv/archive/2.4.13.4.zip

We used the BVLC CaffeNet Model in our proof of concept exploitation. The model is the result of training based on the instructions provided by the instruction in the original Caffe package. To avoid any mistakes in model setup, we downloaded the model file directly from BVLC's official GitHub page. Detailed information about the model is provided in the list below.

Listing 4: Image Classifier Model

```
Name:  BAIR/BVLC CaffeNet Model
Caffemodel: bvlc_reference_caffenet.caffemodel
Caffemodel_url: http://dl.caffe.berkeleyvision.org/bvlc_reference_caffenet.caffemodel
Caffe_commit: 709dc15af4a06bebda027c1eb2b3f3e3375d5077
```

B. Command Lines

The evasion and exploitation threat was demonstrated based on the default Caffe example CPPClassification. The exact command line was shown in the list below.

Listing 5: Image Classification Command Line

```
./classification.bin models/bvlc_reference_caffenet/deploy.prototxt
models/bvlc_reference_caffenet/bvlc_reference_caffenet.caffemodel
data/ilsvrc12/imagenet_mean.binaryproto
data/ilsvrc12/synset_words.txt
IMAGE_FILE
```

C. Sample Output

The list below provides classification results for the above sample images presented in the body of this paper.

Listing 6: Sample Classification Results

```
# bullog [Original Image]
./classification.bin models/bvlc_reference_caffenet/deploy.prototxt
models/bvlc_reference_caffenet/bvlc_reference_caffenet.caffemodel
data/ilsvrc12/imagenet_mean.binaryproto
data/ilsvrc12/synset_words.txt
./poc_samples/bulldog
---------- Prediction for ./poc_samples/bulldog ----------
0.5111 - "n02108915␣French␣bulldog"


# Threat Example 1 -- DoS attack
# bulldog_crash [ cause classification binary to crash]
./classification.bin models/bvlc_reference_caffenet/deploy.prototxt
models/bvlc_reference_caffenet/bvlc_reference_caffenet.caffemodel
data/ilsvrc12/imagenet_mean.binaryproto
data/ilsvrc12/synset_words.txt
./poc_samples/bulldog_crash
---------- Prediction for ./poc_samples/bulldog_crash ----------
Segmentation fault (core dumped)

# Threat Example 2 -- Evasion attack
# bulldog_sh [ cause classification to misclassify to an arbitrary category.]
# [Here the classification produced a class "Flying Pig" made up by attackers]
./classification.bin models/bvlc_reference_caffenet/deploy.prototxt
models/bvlc_reference_caffenet/bvlc_reference_caffenet.caffemodel
data/ilsvrc12/imagenet_mean.binaryproto
data/ilsvrc12/synset_words.txt
./poc_samples/bulldog_fp
---------- Prediction for ./poc_samples/bulldog_fp ----------
0.98 - "n03770679␣flyingpig"

# Threat Example 3 -- Exploitation Attack
# bulldog_sh [ cause classification to generate a local shell]
./classification.bin models/bvlc_reference_caffenet/deploy.prototxt
models/bvlc_reference_caffenet/bvlc_reference_caffenet.caffemodel
data/ilsvrc12/imagenet_mean.binaryproto
data/ilsvrc12/synset_words.txt
./poc_samples/bulldog_sh
---------- Prediction for ./poc_samples/bulldog_sh ----------
$ uname -a
Linux ctf-box 4.4.0-31-generic #50~14.04.1-Ubuntu SMP
Wed Jul 13 01:07:32 UTC 2016 x86_64 x86_64 x86_64 GNU/Linux
$ exit
```