Decoding HDF5: Machine Learning File Forensics and Data Injection

Abstract. The prevalence of Machine Learning (ML) in computing is rapidly expanding and ML systems are continuously applied to novel challenges. As the adoption of these systems grows, their security becomes increasingly important. Any security vulnerabilities within an ML system can jeopardize the integrity of dependent and related systems. Modern ML systems commonly encapsulate trained models in a compact format for storage and distribution, including TensorFlow 2 (TF2) and its utilization of the Hierarchical Data Format 5 (HDF5) file format. This work explores into the security implications of TF2's use of the HDF5 format to save trained models, aiming to uncover potential weaknesses via forensic analysis. Specifically, we investigate the injection and detection of foreign data in these packaged files using a custom tool external to TF2, leading to the development of a dedicated forensic analysis tool for TF2's HDF5 model files.

Keywords: File Forensics · Machine Learning · HDF5 · TensorFlow 2

1 Introduction

Modern software systems are increasingly employing Machine Learning (ML) for various types of tasks. The success of ML has been demonstrated in problems such as image recognition, outcome prediction, process automation, and various other problems [21]. With these accomplishments, it is no surprise that a high volume of research and innovation has been conducted in this area in recent years. Unfortunately, an explosion of success for a complex field of computing can leave security blind spots in systems that use it.

Platforms such as TensorFlow 2 (TF2), PyTorch, and other similar libraries have been extensively crafted specifically for ML model design and implementation. These platforms have made the ML space much more generally accessible, allowing individuals to contribute to this space without a deep knowledge of related mathematics. Many of these platforms have also been built to allow for a pre-trained ML model to be exported in certain file formats. This positively impacts the ease-of-use and accessibility of a model since the process of model training can be a resource-intensive and time-consuming process. If the goal of an individual is to immediately utilize a pre-trained model, however it is delivered to them, then this is the most effective way to do so.

An important note for this convenience is that TF2 models are translated into live Python code. Importing a pre-trained model into the TF2 library loads the contents of the file and initializes data structures into memory based on the file contents. The TF2 documentation aptly warns that all use of TF2 models and code should be properly vetted prior to use due to this translation to live code [11]. Although it would be ideal for all users to follow the security measures advised and manually review the TF2 models with prejudice, not all users or hosts of these models will follow this advice every time.

Taking into account the previous safety advisory, it is also significant to acknowledge the lack of verification tools for the TF2 Hierarchical Data Format 5 (HDF5) files. The HDF5 format has several documented exploits that have been recorded in the Common Vulnerabilities and Exposures (CVE) system maintained by MITRE [9]. Among these CVEs there are many issues that allow for exploits such as a Denial-of-Service (DoS) attack or an out-of-bounds memory read on a system attempting to parse a HDF5 file. There are also CVEs describing exploits of HDF5 that can lead to the execution of arbitrary code on a system, with multiple CVEs of this type recently documented.

With the ability to package and distribute models to various users of a ML platform, a malicious actor could exploit a user's lack of verification to carry out an adverse attack using known or unknown vulnerabilities of the HDF5 format. The TF2 library also lacks explicit verification tools for some HDF5 model file that has been passed to the platform for loading. The lack of focus on verifying HDF5 model files opens the possibility of using them as a viable vector of attack.

Exploring the verification of model files for ML systems is useful for the security of these systems. Although the research targeting the front end of these systems (training process and trained running models) is broad [20, 40, 41], the research at the back end of the system (library and model files themselves) is lacking. The use of these systems is not going to disappear and shows only higher levels of use every year [21]. Attacking these systems from all possible sides to make them more robust is a necessity to prove their security for the future.

Challenges to this goal include the obfuscation of model files that these frameworks introduce and non-exhaustive documentation on how these model files are structured behind these obfuscations. Another primary challenge beyond the full analysis of the model file structure is understanding what constitutes a recognizable and detectable security risk in these files. The goal of this paper is to explore the back end of these systems more directly and to analyze the security threats that exist on the back end side of these high-usage ML frameworks.

With the work presented in this paper, the following contributions are made:

- This work serves as the primary account for file forensics and data injection into the HDF5 format, presenting a novel approach to analyzing the security of machine learning libraries from this perspective. This contribution highlights the research's originality and significance.
- The MLT[1] tools are developed for this work for analyzing and injecting data into TensorFlow HDF5 files. These tools provide practical solutions for examining the structure, and injecting data into HDF5 saved models. The public availability of these tools enhances the reproducibility and applicability of the research.

[1] URL blinded for review

- The structure of the HDF5 saved model is examined, delving into its internal organization and layout. This analysis contributes to a deeper understanding of the structure and functioning of the saved model.
- The file size of HDF5 packages is examined, establishing a correlation between the size and trainable parameters. This investigation sheds light on the factors influencing file size and provides insights into features that can identify injected data within the HDF5 files.
- Data injection into HDF5 model files is explored, and the persistence of the injected data is analyzed. This experimentation investigates the robustness and integrity of saved model files when subjected to data injection, offering valuable insights into potential vulnerabilities or risks of data manipulation.

The rest of the paper is organized as follows: In Section 2, we provide background information and in Section 3 we review related work. Section 4 presents our methodology and Section 5 showcases the results obtained. Furthermore, in Section 6, we introduce our tool, describe its usage, and provide details of its forensic evaluation. Lastly, in Section 7, we discuss the implications of our work and present our concluding remarks in Section 8.

2 Background

2.1 HDF5

HDF5 is a file format specifically designed to store large amounts of variable data in a single package [18]. It is comparable to a relational database table in terms of organization and structure of data within the larger HDF5 file. The HDF5 format has two main types of objects defined in its specification: Datasets and Groups. A Dataset is a multidimensional array of data with a variable shape. A Group is a folder-like container which can hold other Group and Dataset objects. Every Group object is capable of having zero or more Group and Dataset items inside of it, with no maximum limit to how many of these an individual Group stores. These structures allow data within an HDF5 file to be organized hierarchically.

Figure 1 shows the relation mapping of a Group. In terms of class relationship, a Group has zero or more children of Group and/or Dataset objects. A Dataset is a terminal object with no children.

The format of a Group data structure in HDF5 is highly variable and is capable of nesting many layers of Groups within each other. A single Group structure is allowed to contain any number of nonterminal Group structures inside of it, as well as any number of terminal Dataset structures. The ability to deeply nest Groups within each other and organize complex information in a folder-like hierarchy draws those dealing with large amounts of data to the HDF5 format.

An HDF5 file has a Root Group that acts as a top-level container for all other data inside the file, which is the parent for all other Groups and Datasets in the file and the starting point for any operations performed in an HDF5 file.

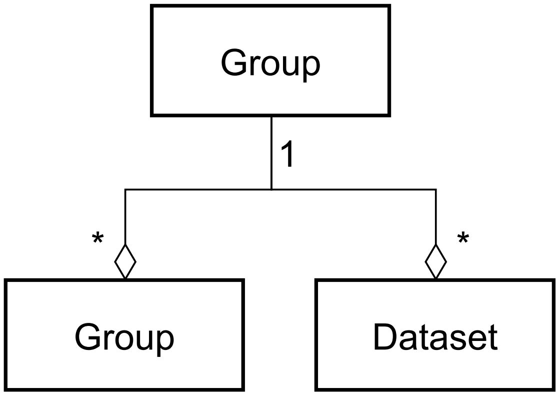


Fig. 1: The HDF5 Group class relationship diagram. This diagram shows that for every one Group, any number of Group and Dataset items can be stored inside of it from zero to an arbitrary amount.

This hierarchical structure is the only guarantee for every HDF5 file, with all other requirements left to the developer.

HDF5 files also have the ability to add Attributes in a file. An HDF5 Attribute is a piece of metadata associated with a Dataset or Group in the file. It is a name-value pair that provides additional information about the data in the file. TF2 uses Attributes for certain information about a model's settings and constraints it was created under, such as versions of libraries.

2.2 CVEs for HDF5

CVE-2016-4330, CVE-2016-4331, CVE-2016-4332, and CVE-2016-4333 demonstrate error-checking issues that have existed in the HDF5 library [1–4]. Several exploits are possible due to a lack of bounds checking for Dataset fields in various contexts. Each of these CVEs has a different mechanism of exploitation, but all lead to heap-based buffer overflow and read / write outside the bounds. If used properly, these exploits allow the execution of arbitrary code in a system.

There are also recent CVEs that show exploits that allow the execution of arbitrary code. The vulnerabilities documented by CVE-2022-25942, CVE-2022-25972, and CVE-2022-26061 show recently documented exploits of the format similar to those from 2016 [5–7]. In these particular CVEs, it is shown that providing a malicious GIF file for conversion to the HDF5 format can similarly trigger arbitrary code execution on a system through heap-based buffer overflow and out-of-bound read/write.

If the target of these overflow attacks is not specific, all of these CVEs discussed could be used for purposes of a system crash or DoS attack through memory corruption of a system. The similarity of the vulnerabilities from years 2016 and 2022 shows the security drawbacks of this flexible file format. Although HDF5 is particularly great for scientific data of arbitrary types and sizes, it is a prime target for security exploits such as those described. This collection of CVEs shows that HDF5 has a record of security issues that allow malicious exploitation of the format.

3 Related Work

ML is frequently used in modern software. Advancements in Deep Learning (DL) allow robust training with large amounts of data for recognition and detection problems, producing high accuracy and precision results [22]. While the number of applications for ML grows, understanding the security risks that come from the widespread use of these highly black-boxed algorithms is still an open research area [14, 23, 41].

3.1 ML for Security

The use of ML in cybersecurity research has become more common in recent years [14,15]. The use of these methods is positive for general security provisions, but also introduces new security risks through the ML model itself [14, 17].

In recent research, ML has been widely used for network monitoring and security applications [17,25,30,32]. As cloud computing services have increased in use, ML has also become useful in securing these large-scale networking systems [31]. These methods are used in Internet of Things (IoT) devices for applications such as verifying the authenticity of requests and detection of intrusion in a secured network of IoT devices [12, 16, 17].

3.2 Security of ML

ML continues to increase in use for applications in all areas of computing [21,37]. An unsecured attack vector of an ML application implicitly affects every system where it is used, making this a topic of high interest for security research [13, 14, 41]. If a successful attack is executed, it can completely disrupt a system's continued functionality when a ML system is a critical front-line service. Research in identifying new attack surfaces and verifying the security of ML models has shown that while the proven disruption of front-end live models is highly effective, the back end of these applications could be just as effective for attack [8]. Although the efficacy of ML models in many applications is high, there is a constant battle against attacks targeting a ML model during live use [13]. Adversarial classification attacks aim to cause an incorrect classification in a ML system [24]. The development of ML model verification tools is one of the avenues explored to protect against this type of attack [26, 29]. These tools analyze a model directly and evaluate susceptibility to adversarial examples and other common attacks similar to them.

A less detectable attack vector, such as a backdoor attack, threatens the security of a system and remains difficult to protect against, even with the verification of a model. This attack is often accomplished by inserting a static visible trigger to create a classification backdoor in a ML model. This backdoor intentionally causes an incorrect classification from the system when this trigger is encountered during live use of the model [20,38]. Further research on this has explored creating dynamic and highly obfuscated triggers, making detection of this attack more difficult [28,36]

The security of users and understanding how ML models affect the privacy of individuals and systems is also of concern [33]. The level of advancement that ML has offered for scientific applications has overshadowed research into these security issues [40].

3.3 File Forensics

Even though ML models are complex systems, storage of a model architecture and inference weights is done in a regular file. Because of this, understanding the file structure in these models can be stored in is very important for the security of the files themselves.

Research in file forensics and data extraction covers carving fragments of data with little file system knowledge [34] to analysis of complex file formats. Analysis and carving of the RAR archive format demonstrate the task of taking portions of a highly variable format and extracting its information dynamically [39].

The HDF5 data model is a non-relational container consisting of highly variable arrays and groups of data [18]. While HDF5 is an open source format [27], security research on the analysis of ML models packaged in this format is lacking. The highly dynamic nature of HDF5 makes it flexible for use across different frameworks with a single format, including the popular frameworks TensorFlow and PyTorch [35].

4 Methodology

The methodology of this paper is as follows:

- Analyze and understand the TF2 HDF5 format.
- Analyze file sizes to identify how much information can be hidden.
- Inject information into the HDF5 file directly with Python.
- Analyze the success of file modification and data persistence.
- Create a dataset of ML HDF5 files to inject information into.
- Test for the detection of injected data that is not part of the original model file in these HDF5 files.

The HDF5 files in this experiment were initially created directly through the TF2 library by training and saving a completely new Convolutional Neural Network (CNN) model. They are then examined and manipulated using the h5py Python library [10]. TF2 also utilizes this library for the creation and saving of HDF5 files in Python. While the same tools are being used and the exporting process in TF2 is well defined, the documentation on the exported file's structure is slim.

4.1 TF2 HDF5 File Structure

HDF5 is a well-understood file format with extensive documentation for its base enforced structure and data types. However, as previously discussed, any enforcement and file structure beyond the specification is an implementation decision for every use case. In the context of TF2, the specific structure of the HDF5 files used by the system is not clearly laid out in the given documentation and deserves further investigation. The basic principles of HDF5 must apply to TF2 HDF5 files, but the exact layout and organization of the data within them often differ from other uses of HDF5. Manipulation of the data contained within these files will not be possible without a good understanding of how the files are structured for TF2. This means that it is quite important to carefully examine these files to see how they are structured and why.

Algorithm 1 HDF5 File Structure Traversal

```
Define $S$                                        ▷ S - Global string for file structure
function TRAVERSE($F$)                            ▷ F - HDF5 File
    if $F$ is an instance of Dataset then
        Append $S$ with Name($F$) and Shape($F$)
    else if $F$ is an instance of Group then
        Append $S$ with $F$
        for $child_F$ in $F$ do
            TRAVERSE($child_F$)
        end for
    end if
end function
```

Algorithm 1 shows the process used to extract the structure from an existing HDF5 file. The Groups of the HDF5 structure are recursively opened through a traversal algorithm. Each Group opened is recorded recursively

, and the Datasets inside of each are recorded in the structure. This allows the file system-like structure of the file to be fully discovered. Once this process is completed, all existing Groups and Datasets in a file will be correctly identified and placed in the logical hierarchy of the file without the need to load the file into TF2.

4.2 HDF5 File Size

After training a model and exporting it to the HDF5 format, file size can be significantly different. The fields for a basic model without custom objects will have the same file structure, but can have significant differences in file size if the complexity of the models differs. Testing is performed to analyze whether a CNN with different numbers of trainable parameters has a noticeable size difference once exported. The goal of this is to find whether the disk size of an exported HDF5 file is strongly correlated with the trainable parameters of a ML model.

If this correlation is demonstrable for CNN models of different sizes, then an analysis of the expected file size versus the actual file size is possible. If a file has a noticeably larger file size than what is expected based on its parameters, then file tampering could be detectable simply by analyzing the file size.

4.3 HDF5 Information Injection

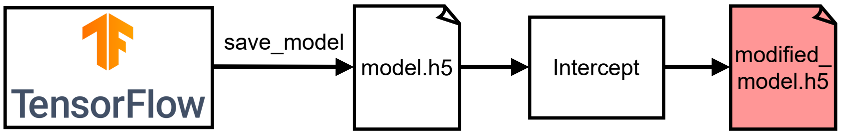


Fig. 2: Workflow of the HDF5 injection process.

Figure 2 demonstrates the workflow of passing a valid HDF5 file as input to the tool and receiving a modified file with the injected data as output. Injection of information into HDF5 files takes place after training and saving a model using normal methods with TF2. The library will produce the saved model file in HDF5 format, which will be intercepted, and unwanted information will be injected.

This process is easy to imagine as viable when model files are produced and distributed in mass. The practice of training models with expensive resources and distributing a packaged model for others to use is a common practice in the ML research and professional space. Intercepting one of these files and modifying its contents would not receive more scrutiny than any other file in an extensive collection of HDF5 packaged models.

5 Results

5.1 TF2 HDF5 File Structure

Table 1 shows the structure of a HDF5 file created through the TF2 library. At the top level Root Group, there are four H5 Attribute fields and two Group fields. After a model has been trained through the library and saved, the outputted model file will have these fields available. Any of the training and validation used for the creation of model weights is not saved in this exported model format. Any of the training and validation data used to create this trained model must be exported separately.

The H5 Attribute data shown in this structure is metadata that is compiled by the TF2 library during a model save. When an HDF5 file structure is more manually structured by a developer, they will most likely be organizing and

Table 1: TF2 HDF5 Format

TF Saved Model HDF5 Format	
Field	**Datatype**
model_weights	H5 Group
optimizer_weights	H5 Group
backend	H5 Attribute
keras_version	H5 Attribute
model_config	H5 Attribute
training_config	H5 Attribute

adding their own H5 Attribute fields. This is not the case with the TF2 HDF5 file structure, however. The developer of the model will not be adding extra H5 Attribute fields manually into this outputted file.

The `optimizer_weights` Group will only be present in a TF2 HDF5 file if it is a trained model. If the file is saved from an untrained model where the compile function is not run, this field will not appear in the file. If the model has custom objects that must be saved along with the standard architecture and inference weight information, the file may have other structures added to it that are not shown in this diagram.

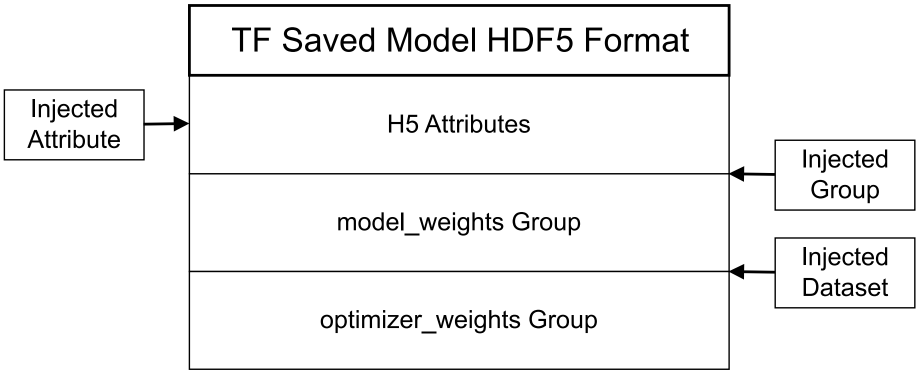


Fig. 3: Positions for injecting more data into TF2 HDF5 packages.

Figure 3 gives a high-level view of where data can be injected particularly into the TF2 HDF5 format. When considering the options available to malicious actors when trying to exploit HDF5, there are three things that can be injected into any HDF5 file. The injection of attribute or lone Group data into a HDF5 file historically does not have highly disruptive issues like the injection of Dataset data. The CVEs discussed that allow arbitrary code execution are all caused by conversion of information to a Dataset.

5.2 HDF5 File Size

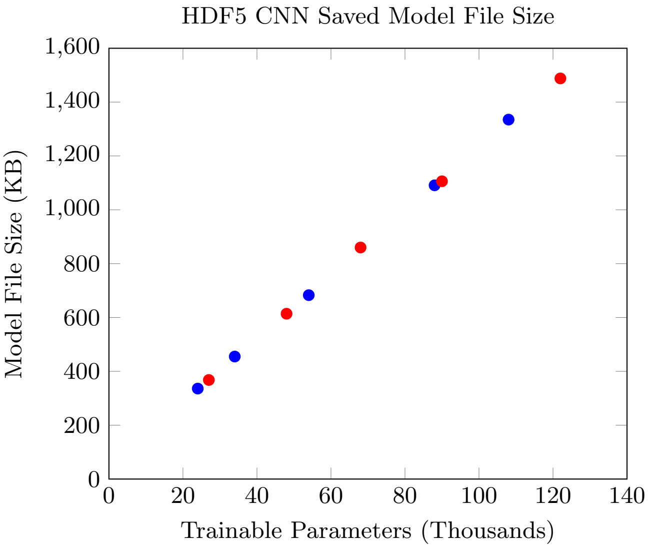


Fig. 4: File size grows linearly with increasing trainable parameters. Blue dots indicate MNIST-trained CNNs and red dots indicating CIFAR-10-trained CNNs.

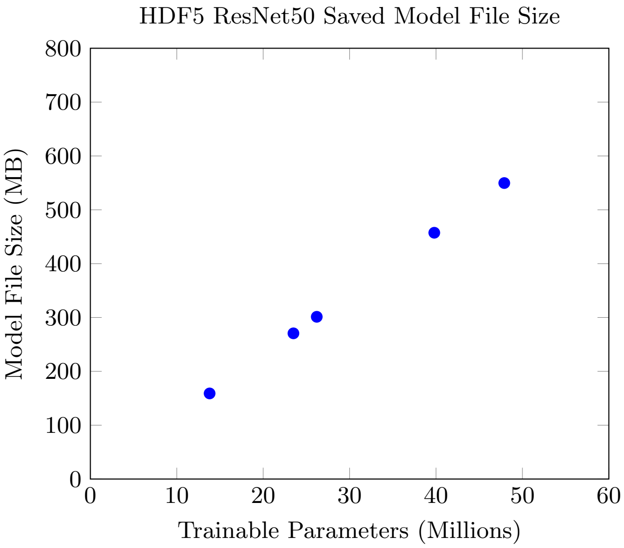


Fig. 5: File size of a ResNet50 neural network, trained to recognize hand sign numbers 0 to 9 akin to MNIST, experiences linear growth.

As the complexity of a ML model increases, the number of trainable parameters in a model increases to larger and larger sizes. The higher model complexity and the larger number of inference weights carry over to the size of a generated HDF5 file when a model is saved. Additional fields that are created for use or injection will also take up more space than would be expected when saved to the TF2 HDF5 format. Figure 4 shows the relationship between the increase in trainable parameters of a CNN model and the saved HDF5 file size. The number of trainable parameters for the CNN was increased by adding more layers to the model.

The results of Figure 4 show a distinct linear relationship between these two variables. With a tight linear fit such as this, it is clear that knowing one of these values gives an easy estimate of what the other should be. This graph also shows that as a CNN grows out of a trivial design, the file size for saving a model is already growing to megabytes in size. With a packaged format that is seldom scrutinized and file sizes growing into megabytes quickly, this opens a useful angle for information hiding and passing through this medium.

Figure 5 shows how quickly a trained ResNet50 HDF5 file grows in size. The trend still shows a linear increase in size, but the overall HDF5 package begins quickly growing into hundreds of megabytes in size. This shows that the type of network does not effect this trend. Even for more complex types of neural networks, this property of linear growth of the HDF5 file size still holds. This shows that there is a predictable pattern for varying model file types and that the type of neural network being saved does not result in different trends.

5.3 Information Injection

The dynamic structure of an HDF5 file allows for extensive amounts of modification. The data inserted into the file does not need to have a particular format due to the flexible Dataset type in the HDF5 standard. Because of this, arbitrary binary information can be inserted into some Dataset field as long as it does not compromise the TF2 library loading the model into memory from this file. The persistence of data in a HDF5 model file is also of note. If the information manually inserted into the HDF5 file does not load into memory when the TF2 library opens the file, then upon resaving this model, the information will be lost.

The Root Group at the top level of a TF2 HDF5 file is a legal location for the insertion of large amounts of data unrelated to the ML system. The inserted data does not disrupt loading and still allows this model file to be loaded into the TF2 library. The data stored in the Root Group does not persist through the model load and save cycle, allowing for simple data to hide up until it is loaded back into the system. A more significant goal of this injection would be data that persists through the loading and saving process.

6 Tool Usage

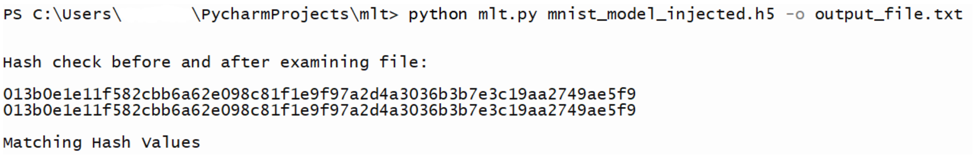
The MLT tool gives the ability to easily analyze an HDF5 file in an unobfuscated form. After providing a valid HDF5 file, the attributes, Groups, and Datasets are all displayed in plaintext for visual analysis. It also provides a way to inject information into a file as a new Dataset without disrupting the structure of the file so that it can be loaded back into TF2 without disrupting the library.

The forensic tool for analyzing file structure is used with the following command:

```
python mlt.py hdf_filename [−o output_file]
```

6.1 File Structure

Figure 6 shows the entire content of a HDF5 file being copied to a destination. It then verifies that the content of the exported file and the original structure match each other to ensure that the structure is correct.



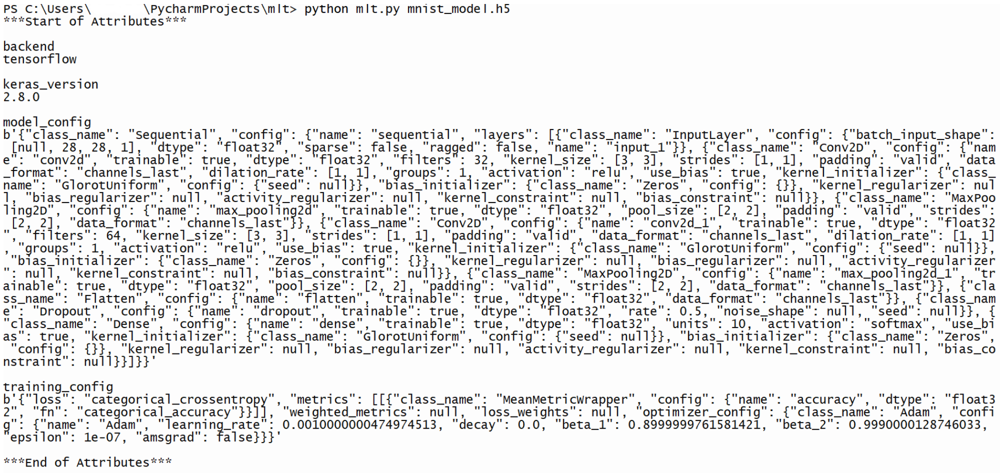
```
PS C:\Users\       \PycharmProjects\mlt> python mlt.py mnist_model_injected.h5 -o output_file.txt

Hash check before and after examining file:

013b0e1e11f582cbb6a62e098c81f1e9f97a2d4a3036b3b7e3c19aa2749ae5f9
013b0e1e11f582cbb6a62e098c81f1e9f97a2d4a3036b3b7e3c19aa2749ae5f9

Matching Hash Values
```

Fig. 6: Copying the file structure of a HDF5 file with MLT. The copy of the file is created, and then hashing against the original is run to ensure original was not modified.



```
PS C:\Users\       \PycharmProjects\mlt> python mlt.py mnist_model.h5
***Start of Attributes***

backend
tensorflow

keras_version
2.8.0

model_config
b'{"class_name": "Sequential", "config": {"name": "sequential", "layers": [{"class_name": "InputLayer", "config": {"batch_input_shape":
 [null, 28, 28, 1], "dtype": "float32", "sparse": false, "ragged": false, "name": "input_1"}}, {"class_name": "Conv2D", "config": {"nam
e": "conv2d", "trainable": true, "dtype": "float32", "filters": 32, "kernel_size": [3, 3], "strides": [1, 1], "padding": "valid", "data
_format": "channels_last", "dilation_rate": [1, 1], "groups": 1, "activation": "relu", "use_bias": true, "kernel_initializer": {"class_
name": "GlorotUniform", "config": {"seed": null}}, "bias_initializer": {"class_name": "Zeros", "config": {}}, "kernel_regularizer": nul
l, "bias_regularizer": null, "activity_regularizer": null, "kernel_constraint": null, "bias_constraint": null}}, {"class_name": "MaxPoo
ling2D", "config": {"name": "max_pooling2d", "trainable": true, "dtype": "float32", "pool_size": [2, 2], "padding": "valid", "strides":
 [2, 2], "data_format": "channels_last"}}, {"class_name": "Conv2D", "config": {"name": "conv2d_1", "trainable": true, "dtype": "float32
", "filters": 64, "kernel_size": [3, 3], "strides": [1, 1], "padding": "valid", "data_format": "channels_last", "dilation_rate": [1, 1]
, "groups": 1, "activation": "relu", "use_bias": true, "kernel_initializer": {"class_name": "GlorotUniform", "config": {"seed": null}},
 "bias_initializer": {"class_name": "Zeros", "config": {}}, "kernel_regularizer": null, "bias_regularizer": null, "activity_regularizer
": null, "kernel_constraint": null, "bias_constraint": null}}, {"class_name": "MaxPooling2D", "config": {"name": "max_pooling2d_1", "tr
ainable": true, "dtype": "float32", "pool_size": [2, 2], "padding": "valid", "strides": [2, 2], "data_format": "channels_last"}}, {"cla
ss_name": "Flatten", "config": {"name": "flatten", "trainable": true, "dtype": "float32", "data_format": "channels_last"}}, {"class_nam
e": "Dropout", "config": {"name": "dropout", "trainable": true, "dtype": "float32", "rate": 0.5, "noise_shape": null, "seed": null}}, {
"class_name": "Dense", "config": {"name": "dense", "trainable": true, "dtype": "float32", "units": 10, "activation": "softmax", "use_bi
as": true, "kernel_initializer": {"class_name": "GlorotUniform", "config": {"seed": null}}, "bias_initializer": {"class_name": "Zeros",
 "config": {}}, "kernel_regularizer": null, "bias_regularizer": null, "activity_regularizer": null, "kernel_constraint": null, "bias_co
nstraint": null}}]}}'

training_config
b'{"loss": "categorical_crossentropy", "metrics": [[{"class_name": "MeanMetricWrapper", "config": {"name": "accuracy", "dtype": "float3
2", "fn": "categorical_accuracy"}}]], "weighted_metrics": null, "loss_weights": null, "optimizer_config": {"class_name": "Adam", "confi
g": {"name": "Adam", "learning_rate": 0.0010000000474974513, "decay": 0.0, "beta_1": 0.8999999761581421, "beta_2": 0.9990000128746033,
"epsilon": 1e-07, "amsgrad": false}}}'

***End of Attributes***
```

Fig. 7: Attributes of the mnist_model.h5 file, displayed with MLT.

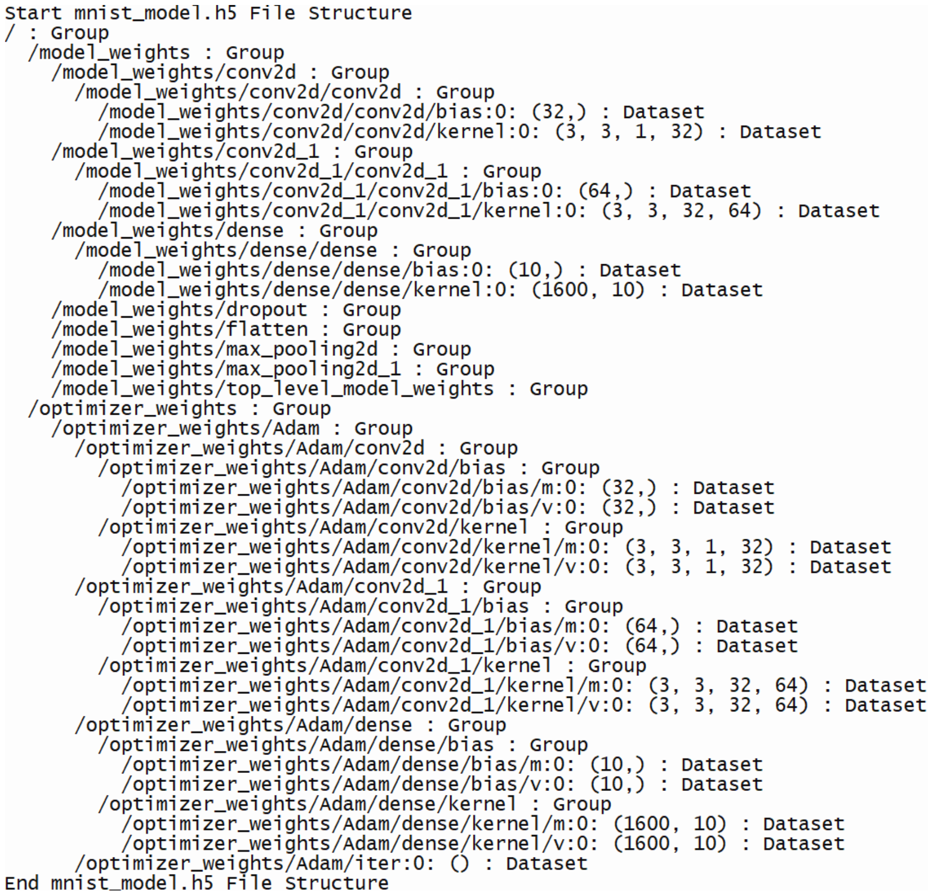
Figure 7 demonstrates the initial output of a HDF5 file when printing its structure directly on the command line. Here, the attributes of the file are also visible. These attributes are not normally easily accessible for viewing when obfuscated in the HDF5 format.

Figure 8 shows a snippet of output from the same command line execution as Figure 7. In this figure, the entire structure of the ML model is shown. This also demonstrates the directory structure of the HDF5 file format. The non-terminal Group objects contain other Groups and Datasets inside of them for organizing the weights of the model and optimizer. The terminal Dataset objects are shown with their shape property to the right of them, showing what the exact structure of the ML model is without loading in the HDF5 file into TF2.

6.2 Injection

The tool for the injection of data into a HDF5 file is used with the following command:

```
python mlt_inject.py hdf_filename copy_filename
    injected_data
```

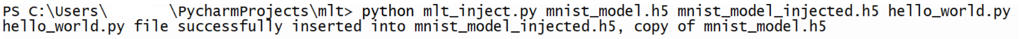


```
Start mnist_model.h5 File Structure
/ : Group
  /model_weights : Group
    /model_weights/conv2d : Group
      /model_weights/conv2d/conv2d : Group
        /model_weights/conv2d/conv2d/bias:0: (32,) : Dataset
        /model_weights/conv2d/conv2d/kernel:0: (3, 3, 1, 32) : Dataset
    /model_weights/conv2d_1 : Group
      /model_weights/conv2d_1/conv2d_1 : Group
        /model_weights/conv2d_1/conv2d_1/bias:0: (64,) : Dataset
        /model_weights/conv2d_1/conv2d_1/kernel:0: (3, 3, 32, 64) : Dataset
    /model_weights/dense : Group
      /model_weights/dense/dense : Group
        /model_weights/dense/dense/bias:0: (10,) : Dataset
        /model_weights/dense/dense/kernel:0: (1600, 10) : Dataset
    /model_weights/dropout : Group
    /model_weights/flatten : Group
    /model_weights/max_pooling2d : Group
    /model_weights/max_pooling2d_1 : Group
    /model_weights/top_level_model_weights : Group
  /optimizer_weights : Group
    /optimizer_weights/Adam : Group
      /optimizer_weights/Adam/conv2d : Group
        /optimizer_weights/Adam/conv2d/bias : Group
          /optimizer_weights/Adam/conv2d/bias/m:0: (32,) : Dataset
          /optimizer_weights/Adam/conv2d/bias/v:0: (32,) : Dataset
        /optimizer_weights/Adam/conv2d/kernel : Group
          /optimizer_weights/Adam/conv2d/kernel/m:0: (3, 3, 1, 32) : Dataset
          /optimizer_weights/Adam/conv2d/kernel/v:0: (3, 3, 1, 32) : Dataset
      /optimizer_weights/Adam/conv2d_1 : Group
        /optimizer_weights/Adam/conv2d_1/bias : Group
          /optimizer_weights/Adam/conv2d_1/bias/m:0: (64,) : Dataset
          /optimizer_weights/Adam/conv2d_1/bias/v:0: (64,) : Dataset
        /optimizer_weights/Adam/conv2d_1/kernel : Group
          /optimizer_weights/Adam/conv2d_1/kernel/m:0: (3, 3, 32, 64) : Dataset
          /optimizer_weights/Adam/conv2d_1/kernel/v:0: (3, 3, 32, 64) : Dataset
      /optimizer_weights/Adam/dense : Group
        /optimizer_weights/Adam/dense/bias : Group
          /optimizer_weights/Adam/dense/bias/m:0: (10,) : Dataset
          /optimizer_weights/Adam/dense/bias/v:0: (10,) : Dataset
        /optimizer_weights/Adam/dense/kernel : Group
          /optimizer_weights/Adam/dense/kernel/m:0: (1600, 10) : Dataset
          /optimizer_weights/Adam/dense/kernel/v:0: (1600, 10) : Dataset
      /optimizer_weights/Adam/iter:0: () : Dataset
End mnist_model.h5 File Structure
```

Fig. 8: Directory structure of Groups and Datasets in the mnist_model.h5 file from the same run of MLT as Figure 7.

This tool allows for an existing TF2 HDF5 file to have data injected into it with a single command without the need to use either TF2 itself or any other tool. By extension, it is also capable of easily injecting a new Dataset field into any HDF5 file. The injection tool leaves the original HDF5 file intact without modfications and creates a new HDF5 file with the designated `copy_filename`. The specified `injected_data` is added to the file as a new Dataset object at the Root Group level. The `injected_data` field can be any type of file that the HDF5 format will parse. The default name "injected_data" is set for the `injected_data` Dataset inside of the HDF5 file.

Figure 9 shows the use of the MLT injection tool to place an arbitrary Python file in the previously demonstrated HDF5 file. This particular example has named the `copy_filename` field to `mnist_model_injected.h5` and the

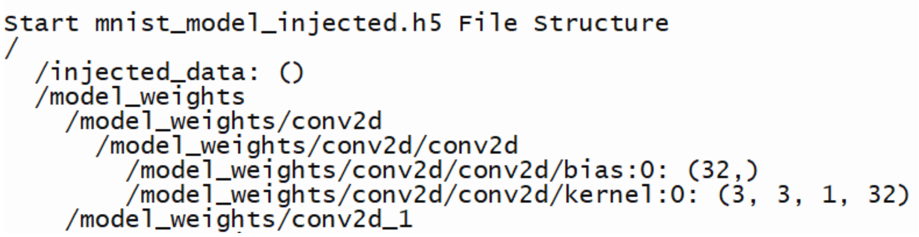


```
PS C:\Users\        \PycharmProjects\mlt> python mlt_inject.py mnist_model.h5 mnist_model_injected.h5 hello_world.py
hello_world.py file successfully inserted into mnist_model_injected.h5, copy of mnist_model.h5
```

Fig. 9: The MLT injection tool creates a copy of mnist_model.h5 in the file mnist_model_injected.h5 and inserts a new field into the structure. The name of the field will be "injected_data" while the content will be the hello_world.py file's text.

`injected_data` field is set to a Python file named `hello_world.py`. This modified HDF5 file loads correctly in TF2 without any difference in behavior from the original.

It is relevant to this injection process that not all types of file can be easily placed in HDF5 format. When trying to save a file that uses internal null bytes, such as a PNG and JPEG file, they will need to be converted from their original file form to a usable container, such as a `NumPy` array. Without taking this step, the `h5py` library will throw an error. Internal null values are not allowed in the HDF5 format since nulls are only used as terminators [19].



```
Start mnist_model_injected.h5 File Structure
/
  /injected_data: ()
  /model_weights
    /model_weights/conv2d
      /model_weights/conv2d/conv2d
        /model_weights/conv2d/conv2d/bias:0: (32,)
        /model_weights/conv2d/conv2d/kernel:0: (3, 3, 1, 32)
    /model_weights/conv2d_1
```

Fig. 10: Structure of the newly created file mnist_model_injected.h5, with the "injected_data" field added.

Figure 10 shows that this injection into a new field was successful. After injection, the MLT tool was used to analyze the file structure of the previously clean file again. The injected data is shown as the top level Dataset field in the HDF5 file above the Groups for the model and optimizer weights.

6.3 Tool Forensic Evaluation

The dataset used includes various CNN model files with varying shapes and a number of features. This dataset was created with varying CNN architectures to test the detection of abnormal data in these HDF5 files. Each of these files is created through the normal training process in TF2 and the export of a HDF5 file after training has been completed. The injection was performed into the Root Group. The MLT tool was run on each of these files and notified if the structure did not match the expected structure in Table 1. Table 2 shows the varying number of trainable parameters in each of the tested files and if the tool is correctly identified once the information is injected. The file size does not affect detection of data injection.

Table 2: Table of CNN Sizes and Detection Success

# of Trainable Parameters in CNN	Injection Detected
24,000	Yes
48,000	Yes
68,000	Yes
90,000	Yes
122,000	Yes

7 Discussion

The highly obfuscated HDF5 package produced by TF2 shows the potential for vulnerability due to the flexibility of the file format itself and the lack of checks performed through the TF2 library for manipulation. The behavior of the TF2 library is not initially affected by data injection not designed for exploitation, but could be utilized for malicious behavior should a new exploit in the HDF5 format be discovered. This possibility is especially worrying in the recently discovered exploited documented by CVE-2022-25942, CVE-2022-25972, and CVE-2022-26061 [5–7]. These exploits are recent discoveries that allow for arbitrary code execution on a machine, which is a very precarious scenario if these exploits are used against a critical system. If a zero-day exploit becomes known to bad actors that circulate these TF2 HDF5 files in public spaces, then ML systems could be attacked through the libraries and packages that make the system operate.

With data injection demonstrated to be possible and CNN packages from TF2 rapidly growing to the size of megabytes, it is clear that information hiding in this format is possible. Even if multiple megabytes needed to be obscured, this is already possible with a simple CNN of just one hundred thousand parameters. While the persistence of the data is a questionable subject, this medium is viable for simple data injection that needs to keep the data in an obscured form.

With the rise of machine learning and obfuscated formats to support distributed trained models, the possibility of abusing the medium becomes more viable. Many developers and researchers using these systems do not closely examine the packages they download and pass into the TF2 library. The exploitation of ML systems could be catastrophic for software in any domain that uses it, which is incredibly important with its current widespread use. The need to secure these libraries against all types of adversarial attack is becoming more urgent as these systems grow and deploy across all fields.

7.1 Limitations

Limitations to this current work include the domain on which is works and the type of detection that can be performed. This work focuses specifically on the HDF5 format for TF2 and analyzes it as a file forensics task based on the format specification. For this detection method, there are no insights made on the model and optimizer weights to verify that they are correct.

8 Conclusion and Future Work

The growing number of libraries and the consistent updates they receive leave many opportunities for security issues in this area. While HDF5 is a format that attracts attention and is in use, there are many other formats to explore. Many ML systems are relatively new and do not have much literature on the security of common formats or proprietary formats for model packaging. Examples of current file types of interest would be those for PyTorch (.pt), ONNX (.onnx), TF2 Protobuf format (.tf), and TensorFlow Lite (.tflife). Exploring different file types and ways of structuring ML data is a viable thread to uncover library and / or format-specific issues.

A model registry is a modern tool in the Machine Learning Operations (MLOps) domain to assist in verifying a ML model. The registry is a repository to store and version ML models as they are modified over time. Model registries allow the models, data, and training processes in a system to change with less worry about errors occurring in a MLOps pipeline.

Once a model is registered, there are several pieces of metadata attached to it in storage. The metadata stored for a model typically includes a unique identifier, training data, training process, and version number. For closed and secure MLOps systems, this type of solution is especially desirable.

These systems are also useful for an individual or group utilizing pre-packaged model files but lack a rigorous system of verifying the authenticity and safety of a model before use. Even if a model has some anomalous data implanted in it, a model registry may still accept the model for storage. This leaves unsuspecting individuals open to various kinds of exploit by carelessly using a file from a seemingly trustworthy source.

There could be another layer of security added to the MLOps pipeline for verification. This would ensure that the content of a ML model does not contain malicious data and could be a vital improvement for these systems, especially those used by a large number of people to upload and download model files.

It would also be viable to explore a large number of ML models with different mechanisms and intended uses. Analyzing what differences there are, if any, between ML systems in a single library could provide insight into vulnerabilities for certain types of models. The many different structures of models make it possible for security risks to appear due to high variability formats such as HDF5.

References

1. CVE-2016-4330. Available from MITRE, CVE-ID CVE-2016-4330. (2016), https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-4330
2. CVE-2016-4331. Available from MITRE, CVE-ID CVE-2016-4331. (2016), https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-4331
3. CVE-2016-4332. Available from MITRE, CVE-ID CVE-2016-4332. (2016), https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-4332
4. CVE-2016-4333. Available from MITRE, CVE-ID CVE-2016-4333. (2016), https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-4333
5. CVE-2022-25942. Available from MITRE, CVE-ID CVE-2022-25942. (2022), https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-25942
6. CVE-2022-25972. Available from MITRE, CVE-ID CVE-2022-25972 (2022), https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-25972
7. CVE-2022-26061. Available from MITRE, CVE-ID CVE-2022-26061. (2022), https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-26061
8. Researchers weaponize machine learning models with ransomware (Dec 2022), https://www.technewsworld.com/story/researchers-weaponize-machine-learning-models-with-ransomware-177489.html
9. Hdf5 cves. Available from MITRE (2023), https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=HDF5
10. Hdf5 for python (2023), https://docs.h5py.org/en/stable/, available at https://docs.h5py.org/en/stable/
11. Using tensorflow securely (2023), https://github.com/tensorflow/tensorflow/blob/master/SECURITY.md, available at https://github.com/tensorflow/tensorflow/blob/master/SECURITY.md
12. Al-Garadi, M.A., Mohamed, A., Al-Ali, A.K., Du, X., Ali, I., Guizani, M.: A survey of machine and deep learning methods for internet of things (iot) security. IEEE Communications Surveys & Tutorials **22**(3), 1646–1685 (2020). https://doi.org/10.1109/COMST.2020.2988293
13. Apruzzese, G., Colajanni, M., Ferretti, L., Guido, A., Marchetti, M.: On the effectiveness of machine and deep learning for cyber security. In: 2018 10th International Conference on Cyber Conflict (CyCon). pp. 371–390 (2018). https://doi.org/10.23919/CYCON.2018.8405026
14. Arp, D., Quiring, E., Pendlebury, F., Warnecke, A., Pierazzi, F., Wressnegger, C., Cavallaro, L., Rieck, K.: Dos and don'ts of machine learning in computer security. In: 31st USENIX Security Symposium (USENIX Security 22). pp. 3971–3988. USENIX Association, Boston, MA (Aug 2022), https://www.usenix.org/conference/usenixsecurity22/presentation/arp
15. Berman, D.S., Buczak, A.L., Chavis, J.S., Corbett, C.L.: A survey of deep learning methods for cyber security. Information **10**(4) (2019). https://doi.org/10.3390/info10040122, https://www.mdpi.com/2078-2489/10/4/122
16. Cui, L., Yang, S., Chen, F., Ming, Z., Lu, N., Qin, J.: A survey on application of machine learning for Internet of Things. International Journal of Machine Learning and Cybernetics **9**(8), 1399–1417 (Aug 2018). https://doi.org/10.1007/s13042-018-0834-5, https://doi.org/10.1007/s13042-018-0834-5
17. Ferrag, M.A., Maglaras, L., Moschoyiannis, S., Janicke, H.: Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study. Journal of Information Security and Applications **50**,

102419 (2020). https://doi.org/https://doi.org/10.1016/j.jisa.2019.102419, https://www.sciencedirect.com/science/article/pii/S2214212619305046

18. Folk, M., Heber, G., Koziol, Q., Pourmal, E., Robinson, D.: An overview of the hdf5 technology suite and its applications. In: Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases. p. 36–47. AD '11, Association for Computing Machinery, New York, NY, USA (2011). https://doi.org/10.1145/1966895.1966900, https://doi.org/10.1145/1966895.1966900
19. Folk, M., Heber, G., Koziol, Q., Pourmal, E., Robinson, D.: An overview of the hdf5 technology suite and its applications. In: Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases. p. 36–47. AD '11, Association for Computing Machinery, New York, NY, USA (2011). https://doi.org/10.1145/1966895.1966900, https://doi.org/10.1145/1966895.1966900
20. Goldblum, M., Tsipras, D., Xie, C., Chen, X., Schwarzschild, A., Song, D., Madry, A., Li, B., Goldstein, T.: Dataset security for machine learning: Data poisoning, backdoor attacks, and defenses. IEEE Transactions on Pattern Analysis and Machine Intelligence pp. 1–1 (2022). https://doi.org/10.1109/TPAMI.2022.3162397
21. Hatcher, W.G., Yu, W.: A survey of deep learning: Platforms, applications and emerging research trends. IEEE Access **6**, 24411–24432 (2018). https://doi.org/10.1109/ACCESS.2018.2830661
22. Hatcher, W.G., Yu, W.: A survey of deep learning: Platforms, applications and emerging research trends. IEEE Access **6**, 24411–24432 (2018). https://doi.org/10.1109/ACCESS.2018.2830661
23. He, Y., Meng, G., Chen, K., Hu, X., He, J.: Towards security threats of deep learning systems: A survey. IEEE Transactions on Software Engineering **48**(5), 1743–1770 (2022). https://doi.org/10.1109/TSE.2020.3034721
24. Huang, S., Papernot, N., Goodfellow, I., Duan, Y., Abbeel, P.: Adversarial attacks on neural network policies (2017). https://doi.org/10.48550/ARXIV.1702.02284, https://arxiv.org/abs/1702.02284
25. Karatas, G., Demir, O., Koray Sahingoz, O.: Deep learning in intrusion detection systems. In: 2018 International Congress on Big Data, Deep Learning and Fighting Cyber Terrorism (IBIGDELFT). pp. 113–116 (2018). https://doi.org/10.1109/IBIGDELFT.2018.8625278
26. Katz, G., Huang, D.A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljić, A., Dill, D.L., Kochenderfer, M.J., Barrett, C.: The marabou framework for verification and analysis of deep neural networks. In: Dillig, I., Tasiran, S. (eds.) Computer Aided Verification. pp. 443–452. Springer International Publishing, Cham (2019)
27. Koziol, Q., Robinson, D., of Science, U.O.: Hdf5 (3 2018). https://doi.org/10.11578/dc.20180330.1, https://www.osti.gov//servlets/purl/1631295
28. Li, Y., Li, Y., Wu, B., Li, L., He, R., Lyu, S.: Invisible backdoor attack with sample-specific triggers (2020). https://doi.org/10.48550/ARXIV.2012.03816, https://arxiv.org/abs/2012.03816
29. Ling, X., Ji, S., Zou, J., Wang, J., Wu, C., Li, B., Wang, T.: Deepsec: A uniform platform for security analysis of deep learning model. In: 2019 IEEE Symposium on Security and Privacy (SP). pp. 673–690 (2019). https://doi.org/10.1109/SP.2019.00023
30. Liu, H., Lang, B.: Machine learning and deep learning methods for intrusion detection systems: A survey. Applied Sciences **9**(20) (2019). https://doi.org/10.3390/app9204396, https://www.mdpi.com/2076-3417/9/20/4396

31. Nassif, A.B., Talib, M.A., Nasir, Q., Albadani, H., Dakalbab, F.M.: Machine learning for cloud security: A systematic review. IEEE Access **9**, 20717–20735 (2021). https://doi.org/10.1109/ACCESS.2021.3054129
32. Nguyen, G., Dlugolinsky, S., Tran, V., Lopez Garcia, A.: Deep learning for proactive network monitoring and security protection. IEEE Access **8**, 19696–19716 (2020). https://doi.org/10.1109/ACCESS.2020.2968718
33. Papernot, N., McDaniel, P., Sinha, A., Wellman, M.P.: Sok: Security and privacy in machine learning. In: 2018 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 399–414 (2018). https://doi.org/10.1109/EuroSP.2018.00035
34. Poisel, R., Tjoa, S.: A comprehensive literature review of file carving. In: 2013 International Conference on Availability, Reliability and Security. pp. 475–484 (2013). https://doi.org/10.1109/ARES.2013.62
35. Rojas, E., Kahira, A.N., Meneses, E., Bautista-Gomez, L., Badia, R.M.: A study of checkpointing in large scale training of deep neural networks. CoRR **abs/2012.00825** (2020), https://arxiv.org/abs/2012.00825
36. Salem, A., Wen, R., Backes, M., Ma, S., Zhang, Y.: Dynamic backdoor attacks against machine learning models. In: 2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P). pp. 703–718 (2022). https://doi.org/10.1109/EuroSP53844.2022.00049
37. Verbraeken, J., Wolting, M., Katzy, J., Kloppenburg, J., Verbelen, T., Rellermeyer, J.S.: A survey on distributed machine learning. ACM Comput. Surv. **53**(2) (mar 2020). https://doi.org/10.1145/3377454, https://doi.org/10.1145/3377454
38. Wang, J., Hassan, G.M., Akhtar, N.: A survey of neural trojan attacks and defenses in deep learning (2022). https://doi.org/10.48550/ARXIV.2202.07183, https://arxiv.org/abs/2202.07183
39. Wei, Y., Zheng, N., Xu, M.: An automatic carving method for rar file based on content and structure. In: 2010 Second International Conference on Information Technology and Computer Science. pp. 68–72 (2010). https://doi.org/10.1109/ITCS.2010.23
40. Xiao, Q., Li, K., Zhang, D., Xu, W.: Security risks in deep learning implementations. In: 2018 IEEE Security and Privacy Workshops (SPW). pp. 123–128 (2018). https://doi.org/10.1109/SPW.2018.00027
41. Xue, M., Yuan, C., Wu, H., Zhang, Y., Liu, W.: Machine learning security: Threats, countermeasures, and evaluations. IEEE Access **8**, 74720–74742 (2020). https://doi.org/10.1109/ACCESS.2020.2987435