Preventing Machine Learning Poisoning Attacks Using Authentication and Provenance

Jack W. Stokes
Microsoft Research
Redmond, WA USA
jstokes@microsoft.com

Paul England
Microsoft Research
Redmond, WA USA
pengland@microsoft.com

Kevin Kane
Microsoft Research
Redmond, WA USA
kkane@microsoft.com

***Abstract*—Recent research has successfully demonstrated new types of data poisoning attacks. To address this problem, some researchers have proposed data poisoning *detection* defenses which employ machine learning algorithms to identify such attacks. In this work, we take a different approach to preventing data poisoning attacks which relies on cryptographically-based authentication and provenance to ensure the integrity of the data used to train a machine learning model. The same approach is also used to prevent software poisoning and model poisoning attacks. A software poisoning attack maliciously alters one or more software components used to train a model. Once the model has been trained it can also be protected against model poisoning attacks which seek to alter a model's predictions by modifying its underlying parameters or structure. Finally, an evaluation set or test set can also be protected to provide evidence if they have been modified by a second data poisoning attack during inference. To achieve these goals, we propose VAMP which extends the previously proposed AMP system, that was designed to protect media objects such as images, video files or audio clips, to the machine learning setting. We first provide requirements for authentication and provenance for a secure machine learning system. Next, we demonstrate how VAMP's manifest meets these requirements to protect a machine learning system's datasets, software components, and models.**

***Index Terms*—data poisoning, provenance, cryptographic hash**

I. Introduction

As machine learning models become increasingly ubiquitous within industrial and governmental settings, more effort is needed to maintain, manage, and protect the data and software components used to train these models as well as the trained models themselves. Researchers have recently proposed data poisoning attacks [1]–[5] where data is specifically altered in order to train a model which produces incorrect outputs. In a related attack, which we call a software poisoning attack, the software or the training framework is altered to intentionally introduce a vulnerability or bug. Finally in a model poisoning attack, the model's parameters and its structure may be altered to again produce a malicious output.

Attackers seek to exploit data, software, and model poisoning attacks against machine learning systems during four phases of development and production. First, the training and validation datasets that are used for training a model may be altered in a data poisoning attack either by insider threats or from man-the-middle attacks which modify the data during transmission over a network such as the Internet. For example, a face recognition model which is trained using poisoned data may provide an attacker with a backdoor into computer systems allowing them to pose as a valid user [6]. Second in a software poisoning attack, the machine learning software, packages, or containers (e.g., Docker) used to train the model may be maliciously altered to introduce a vulnerability. Examples of software poisoning attacks include the introduction of a difficult-to-discover vulnerability, malware or a backdoor into the machine learning system infrastructure. Third, if the model was trained with pristine (i.e., clean) data and software, its parameters or structure may be modified in a model poisoning attack to produce incorrect results during inference. Finally, it may be possible to conduct an additional data poisoning attack against the unknown data used for inference or the test dataset used to evaluate the model's performance. As a result, we seek to prevent all types of attacks against the data, software, or the trained models.

One method for preventing data poisoning attacks is data poisoning detection which falls into two main approaches: offline [7] and online [8]. In an offline detection system, algorithms analyze the trained model to determine if it was trained with poisoned data. In online detection, algorithms seek to monitor the data being used to train the model during the training process and identify poisoned data. Since data poisoning detection relies on statistical algorithms to detect these types of attacks, they produce both false positives and false negatives.

Provenance has been previously proposed to protect machine learning systems [9], [10]. In this work, we take a different approach to the problem of preventing data poisoning attacks. We propose the use of *cryptographic hashing*, in addition to provenance, in order to protect the original datasets that are used for training and validation. In addition, cryptographic hashing also ensures the integrity of the software used to train or evaluate the model. The trained model itself is protected with cryptographic hashing. When a model has been trained, we again use cryptographic hashing in order to ensure that it is not been tampered with by attackers. This allows us to prevent the incorrect use of the model during inference. If the cryptographic hashes ensure that the datasets, software, and model have been authenticated using cryptographic hashes, the system's integrity is assured.

Using provenance and cryptographic hashing to combat fake

media, including photoshopped images and both cheapfake and deepfake videos, has been recently proposed [11]–[13]. A number of examples of this approach include Project Origin [12] and the Content Authenticity Initiative (CAI) [13]. CAI focuses on images captured in a camera, using a secure hardware enclave, through the content creation process using photo editing tools [13]. Project Origin, which is an alliance between the BBC, CBC, Microsoft, and the New York Times, instead protects the integrity of images and videos from the point of initial publication to the display on a webpage or in a mobile app [11], [12].

Project Origin uses the AMP (*A*uthentication of *M*edia via *P*rovenance) system [11] as the underlying authenticity and provenance technology. AMP is a proof-of-concept Azure web service which includes several components that combine to convey provenance to the end user when consuming a piece of media. A publisher first creates metadata related to the image, video, or audio clip which embeds this metadata in a data structure we call a manifest, cryptographically binds the manifest to a media object via object hashes, and then signs the manifest. The publisher then uploads the manifest to the manifest database in the web service or alternatively embeds the manifest into the media file itself. Next, the manifest, or its cryptographic hash, is inserted into an immutable, provenance ledger using the Confidential Consortium Framework (CCF) [14]. This provenance ledger provides publicly available evidence that the manifest has not been modified by attackers. CCF returns a receipt to the database which can be used for fast verification that the manifest has been stored in the ledger without the need for the client to query the ledger itself.

In this paper, we extend AMP to create a new authenticity and provenance system called VAMP which aims to prevent data, software and model poisoning attacks directed towards all aspects of the machine learning system. VAMP stands for the *V*erifiablilty and *A*uthentication of *M*achine Learning and Media Objects via *P*rovenance. We first define the requirements of a cryptographically-protected machine learning provenance system. Next, we show how VAMP fulfills these requirements with only minor modifications to the underlying AMP system. The main contributions of this paper include:

- We propose a cryptographic-based authentication and provenance solution for machine learning systems for the prevention of data, software, and model poisoning attacks against the training, validation, test, and evaluation datasets, the machine learning software and components, and the trained model.
- We list the requirements for different phases of developing and deploying a machine learning system that need to be addressed by a machine learning authentication and provenance system.
- We extend the AMP system to the machine learning setting and show how its relational concepts that are important for protecting media are also important in the machine learning setting.
- We use a highly performant provenance ledger using the confidential consortium framework to offer public assurance of the integrity of the objects in the machine learning environment.
- We implement this system as an Azure web service.

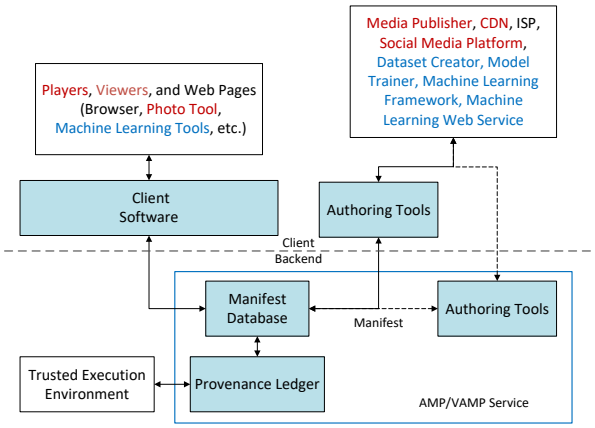

Fig. 1. High-level overview of the VAMP system components (shaded boxes).

The paper is organized as follows. The AMP system is reviewed in Section II. In Section III, we define the concepts of authentication and provenance, and then provide an overview of the system. The VAMP requirements for the protection of machine learning systems are given for datasets (Section IV), software (Section V), and models (Section VI). The key data structure for meeting these requirements is the Manifest which is described in Section VII. Finally, the trust model is given in Section VIII.

II. AMP

VAMP can be considered as an extension of the AMP system [11] which provides cryptographically-authenticated provenance for media. In this section, we briefly review the AMP system. Figure 1 depicts the original AMP system along with the extensions to VAMP. Its key components include a manifest which provides the metadata for the media objects, the *Manifest Database* which stores previously uploaded manifests, and a *Provenance Ledger* (i.e., blockchain) which provides publicly available evidence that the media has not been modified. In addition, the manifest includes fields which allow the media consumer to track its provenance backwards through the media capture, editing, publishing, and distribution graph to its original source.

AMP manifests can be stored using two separate methods: embedded or detached. AMP proposes a new way of embedding (i.e., inserting) manifests into MP4 files. In the detached manifest scenario, the manifest is instead stored separately in the *Manifest Database*. A client application such as a browser extension or a webpage then can authenticate media using the file itself if the manifest is embedded or using the *AMP Service* if the manifest is detached.

AMP serializes the manifest in two ways using both CBOR and JSON. CBOR is more efficient since it stores the data in a binary format whereas a manifest stored using JSON in a plaintext format is human readable.

AMP uses a X.509 PKI (public key infrastructure) trust model. The serialized manifests are signed with the publisher's private key allowing client applications to verify the publisher's identity.

In following sections, we discuss how the media-focused AMP system can also be extended to help protect machine learning systems.

III. Authentication and Provenance of Machine Learning Systems

Both authentication and provenance play critical functions in a *secure* machine learning system. Authentication of a machine learning system's components, such as datasets, software, and models, ensures the trustworthiness of the final prediction results. Similarly, provenance allows the model trainer, and to some extent the user, to trace and verify all of the components that were used to train the system backwards through the provenance graph to the original sources.

Authentication. In a secure machine learning system, authentication is the process of the consumer validating the veracity all of the machine learning objects. Before using a machine learning object in a secure system, the object consumer first verifies its signature. Next, they confirm that each object's one or more signed cryptographic hashes match those generated from its contents. Building a cryptographically protected provenance system for the prevention of data, software, and model poisoning attacks involves three main tasks including protecting the training and validation datasets, protecting the software used to train and evaluate the model, and later protecting the trained model so that it can be used for tamper-evident inference. In addition, the evaluation dataset can also be protected and verified before inference as an optional fourth task.

VAMP's pipeline in illustrated in Figure 2. First, the dataset creator generates the training and validation datasets. Next, they create the manifests (i.e., metadata and data bindings) and either upload (i.e., publish) them to the VAMP service or embed them directly in the individual datasets. Manifests for all of the software (e.g., source code, packages, containers) that is used for training the model, as well as any software that is used to evaluate the model, are then embedded in the objects or uploaded to the VAMP service. Next, the model creator validates that the content bindings in the manifests match the data and labels in the datasets and the model training software components. After the validation succeeds, the model creator trains the machine learning model. Afterwards, the model creator generates a manifest for the model and either uploads the manifest to the VAMP service or embeds the manifest into the model itself.

All of the software (e.g., source code, packages, containers) that is used for training the model, as well as any software that is used to evaluate the model, is then uploaded to the service.

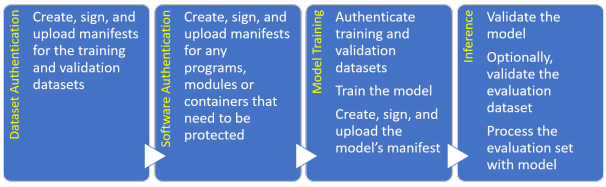

Fig. 2. Authentication steps for protecting datasets, software, and a trained machine learning model.

Next, the model creator validates that the content bindings in the manifests match the data and labels in the dataset and the model training software components. After the validation succeeds, the model creator trains the machine learning model. Afterwards, the model creator generates a manifest for the model and either uploads the manifest to the VAMP service or embeds the manifest into the model itself.

Once the model has been trained and its manifest embedded or uploaded to the service, the user can perform inference on the unknown evaluation set. Similar to the previous steps in this process, authentication and provenance are also an important aspects for inference. The user first validates that the model bindings match the contents of the model file. It is possible at this point that the manifest for the evaluation dataset has been previously generated and either uploaded to the VAMP service or embedded in the dataset. If so, the user also validates that the bindings in the manifest match the data, and labels if they exist, in the dataset. After validating the model, and possibly the evaluation dataset, the user produces the final scores for the evaluation dataset using the model.

Since VAMP is an extension of AMP, it also employs CCF to help ensure the integrity of the machine learning system by providing a public audit trail of the machine learning objects that were either used to train the system or to evaluate new data. When manifests for data, software, and models are uploaded to the service, they are stored in the ledger.

Provenance. In a machine learning system, provenance involves being able to understand how all of the different machine learning objects are processed to yield the final prediction score. Provenance also plays a key role in securing a machine learning system. Provenance can typically be represented as a directed acyclic graph (DAG), and therefore provenance can be traced in either direction from the start to the finish or backwards from the end to the beginning. Many complex machine learning systems consist of simpler machine learning components. Each of these components may be trained with different datasets which may or may not be standard (e.g., ImageNet). In order to verify that the final complex model has been trained using authenticated datasets, software, and underlying (i.e., upstream) models, it requires a provenance graph which indicates all of these system components. Provenance is also important for creating machine learning systems that generate reproducible results [15], [16].

In the next sections, we first describe the requirements im-

posed by many machine learning systems on datasets, software components, and the training process. We then describe how manifests and VAMP can be used to meet these machine learning requirements.

IV. Datasets

The first requirement for preventing data poisoning attacks in machine learning systems is creating datasets with manifests which cryptographically bind the metadata to the data and labels, if they exist. Since data poisoning attacks primarily target the model training process, it is most important to protect the training and validation datasets so the model training code (e.g., PyTorch, TensorFlow, ML.Net) can authenticate the data used during training.

Unlike formatted media objects which are addressed by AMP, machine learning datasets are often text files, although the raw data underlying vision datasets are often standard image formats such as JPEG. Text file-based machine learning datasets typically have a custom format which either has a prescribed format definition or includes a header which defines this format. It may not be possible to modify standard datasets since the metadata cannot be inserted into the dataset without breaking the existing training code. In this case, the dataset creator's manifest must be stored externally such as in a separate sidecar file or in a web service. For new datasets, the metadata format could be specified and inserted into the text file itself.

Another important requirement of a secure machine learning system is determining what is the important metadata needed for the datasets and what are important aspects (i.e., fields) of the metadata to cryptographically protect.

V. Machine Learning Software

In addition to protecting the datasets, it is also important to protect all of the software used to train the model to prevent software poisoning attacks. For machine learning systems, this software can either be text-based software such as Python, Java, or C# code or it may be binary packages needed to create features (e.g., OpenCV) or train the model itself. Machine learning models are often trained within containers (e.g., Docker, NuGet), and these containers need to be protected as well. Thus, it is important for a machine learning system to protect all aspects of the machine learning software training and inference environments.

VI. Model Training

After the manifests for the training and validation datasets and all software components have been successfully uploaded to the authentication service or embedded into the files, the datasets, software, and manifests can be used to train the machine learning model with cryptographic guarantees that the components have not been altered. The software must be validated before training the model. The datasets can either be verified at the start of training or continuously verified during the training process if the dataset is provided using a streaming service. After training has completed, it is important to protect the final model used to evaluate unknown datasets during inference.

Verifying the Datasets Before Training. In the majority of cases, the training and validation datasets can be verified once before training the model. This verification may be done by computing a single hash for each dataset or the verification may be done in chunks for more efficient, and perhaps parallel, processing.

Multiple Minibatch Sizes. When training standard machine learning models such as logistic regression, the parameters are updated based on the loss for each *individual* sample in the training set. In this batch setting, the order that the samples are processed is chosen at random for each subsequent epoch. On the other hand, deep learning models are typically trained using minibatches, and one important hyperparameter that is chosen during training is the minibatch size. Instead of randomly choosing the order for each sample as done in the standard machine learning setting, the order of contiguous minibatches is chosen randomly when training deep learning models. The system must be able to validate datasets in both the batch and mini-batch modes during training. In addition, the system must support multiple minibatch sizes for both of these modes.

Adaptive Minibatch Sizes. A recently proposed deep learning training approach seeks to learn the optimal minibatch size during training [17]. For adaptive minibatch size training, the objective function includes a term which allows the minibatch size to be changed during training using stochastic gradient descent (SGD). Thus, if adaptive minibatch size training is required, the secure machine learning system must be able to support dynamically changing the minibatch size from epoch to epoch.

VII. VAMP Manifests

Next, we demonstrate how VAMP fulfills the requirements of a secure machine learning provenance system. The manifest is the key data structure in VAMP. Its main two functions are to 1) define and cryptographically protect the critical metadata that is important for the datasets, software, and models, and 2) cryptographically bind this metadata to these machine learning objects. In addition, manifests can also define relationships between these objects which allow the model trainer, and maybe even the user in some cases, to trace the machine learning object provenance from the final prediction score or the inference object back to all of its original components.

Key Manifest Metadata Fields. All of the metadata structures related to media authentication and provenance are described in the AMP system design and are provided in Appendix C in [11]. We believe that the AMP metadata structures can also be used for machine learning metadata with minor modifications. Thus, we have not added or removed any of metadata fields for VAMP.

The key manifest fields are listed in Table I. These fields match those originally specified in AMP with one exception. AMP specified a MediaID, but this field has been changed to ObjectID in VAMP allowing it to generalize to the machine

learning scenario. The ObjectID is the identifier of the machine learning object that is being protected. In addition, a number of the AMP field descriptions reference media related concepts. In VAMP, these field descriptions are updated to be more general.

Manifest Data Binding. Manifests are cryptographically bound to the data and the labels, if they exist, located in the datasets by computing one or more cryptographic hashes of the data and labels. Similarly, the cryptographic hashes are also computed for any software components and the final trained model. VAMP uses SHA2 cryptographic hashes such as SHA2-256 or SHA2-512 for this purpose.

VAMP supports four different types of content binding: static, fixed-length chunk, box-based, and offset-length [18]. For a static content binding, a cryptographic hash is computed over the entire contents of the object (e.g., machine learning small dataset, source code, model) that need to be protected. For large datasets using a fixed-length chunk content binding, the file is divided into consecutive fixed-length chunks, and the cryptographic hash is computed for each chunk.

In AMP, the box-based binding is used to insert the metadata and content hashes into the media file according to its format specification (i.e., moov, moof, mdat), and this type of binding is also included in VAMP to support media objects. VAMP also uses a offset-length binding which embeds the offset and length of the key fields and structures when this information is embedded in the file. VAMP's layout of a machine learning object is provided in Table II for files where the header can be modified. This format supports both embedded and detached manifests. The beginning of the file contains the ManifestType which indicates if the manifest is embedded or detached. The next field, ManifestSerialization, specifies if the manifest is stored in canonicalized JSON or canonicalized CBOR. If the ManifestType indicates that the manifest is detached, the next field ManifestLocator provides the URI of the manifest. Otherwise, the signed manifest is embedded and located in the next section of the file. For text files, the manifest is Base64 encoded. Finally, the media object itself is stored in the remainder of the file.

The offset-length binding can also be used to support datasets with minibatches of different lengths for training deep learning models. Since the best minibatch size is typically learned during hyperparameter tuning, it may not be possible to set this parameter beforehand. Therefore, the challenge with supporting different minibatch sizes is that the minibatch hashes must be computed for each possible setting of the minibatch size hyperparameter.

An example of a text-based dataset with an embedded manifest is depicted in Table III. where the data is divided into minibatches 1 though N. These minibatches contain a fixed number of examples (e.g., 64). However, since each row in the dataset consists of a potentially different number of characters, each minibatch can have a different length and be located at a non-uniform offset from the beginning of the file. The offset-length binding is also important in the case where the user may want to repeatedly validate each minibatch for an epoch during training. In the future, AMP's box-based binding format can also be extended in VAMP to handle chunks of varying offsets and lengths to accommodate minibatches in a deep learning dataset.

Machine learning datasets can also be stored in a binary format instead of a text file. In this case, the minibatches are contained within a fixed number of bytes. However, these fixed-length minibatches may still be stored with a random offset due to the inclusion of the manifest and the column specification header, and the offset-length content bindings are still required. Once the hashes have been computed and inserted into the manifest, along with the other important metadata, the key parts of the manifest that need to be protected are then signed to provide evidence if an attacker has modified the dataset, software, or model.

Transformation Manifests. Transformation Manifests are used to indicate provenance in a machine learning system. All of the datasets, software components, and models can form a complicated graph. Understanding this complete graph is important for implementing reproducible machine learning systems. Transformation Manifests include one or more backpointers to the manifests of other machine learning objects. These backpointers support a derived object relationship and can be used in a number of ways. For example, one common practice in machine learning is to start with a pretrained model which has been trained with data and labels for another purpose. Using this pretrained model as a starting point, it is finetuned with a different set of data and labels to achieve another objective. In this case, the finetuned model is "transformed" from the original, pretrained model, and a backpointer from the finetuned model to the pretrained model indicates this relationship.

In another example, an uncompressed dataset can be "transformed" from a compressed dataset. Decoding an encoded dataset into a text or binary file can be represented using a Transformation Manifest. In addition, an encoded dataset may be transcoded from one lossless compression algorithm to another including compression algorithms such as Gzip, Huffman encoding, or Run-length encoding, and this transcoding operation can be represented using another Transformation Manifest.

A Transformation Manifest also supports the notion of derivation from multiple objects. In machine learning, we can create a Transformation Manifest which contains backpointers to the manifests of the training and validation datasets used to train a model, for example. In another case, the Transformation Manifest may provide pointers back to multiple submodels used during preprocessing steps to create features for the final model.

Facsimiles. Facsimiles are datasets, and possibly even models, that the data publisher or the model creator believe are similar or related in some way. Manifests can authenticate any number of facsimiles in machine learning scenarios where facsimiles may be used for 1) splitting one labeled dataset into separate training, validation, and test datasets, 2) training with different minibatch sizes, 3) adaptively selecting the minibatch

Field	Manifest Type	Description
ObjectID	Static/Streaming	Publisher-assigned identifier for the object.
MasterCopyLocator	Static/Streaming	URI of a stable, publisher provided location service or a generic URL redirector service.
EncodingInformation	Static/Streaming	String describing the object type (e.g., "JPEG", "MP4", "Gzip", "Huffman encoding").
OriginManifestID[]	Static/Streaming	One or more ManifestIDs that describe the source object used to create a derived work.
Copyright	Static/Streaming	Copyright string associated with the object.
ObjectHash[]	Static	Cryptographic hash of the associated simple object (or collection of related objects objects).
ChunkDigest	Streaming	An ordered array of chunk-hashes starting from the beginning of the work.

TABLE I
KEY MANIFEST FIELDS.

Field	Storage	Description
ManifestType	Embedded / Detached	Specifies if the manifest is Embedded or Detached.
ManifestSerialization	Embedded / Detached	Specifies if the manifest is serialized using canonicalized JSON or CBOR.
ManifestLocator	Detached	URI of the manifest.
Manifest	Embedded	The signed manifest of the machine, learning object.
Data/Code	Embedded / Detached	The data for a dataset, the code for a program, the binary data for a software component, or the architecture and parameters for a model.

TABLE II
MACHINE LEARNING OBJECT FILE STRUCTURE SUPPORTING BOTH DETACHED AND EMBEDDED MANIFESTS.

Field	Description
ManifestType	Specifies that the manifest is Embedded.
ManifestSerialization	Specifies if the manifest is serialized using canonicalized JSON or CBOR.
Manifest	The signed manifest of the dataset.
Data	Minibatch 1
Data	Minibatch 2
Data	...
Data	Minibatch N

TABLE III
TEXT-BASED, DATASET FILE EXAMPLE WITH AN EMBEDDED MANIFEST AND FIXED-LENGTH MINIBATCHES.

size during training, and 4) constructing datasets of different sizes, including subsampling and oversampling, from a single dataset.

In the first example, a single dataset can be split into separate training, validation, and test datasets. In this case, the single large dataset can be considered to be a facsimile of a collection of the individual training, validation, and test datasets. Another example of a pair of datasets which can be considered as facsimiles includes two datasets where the first is a single multiplexed dataset containing both the data and labels, and the second includes two datasets where the data and the labels are stored separately.

Manifest Storage Locations. In the original design, AMP allows for manifests to differ based on their storage location. The different storage types of AMP manifests include detached and embedded. Likewise, VAMP includes both detached and embedded manifests. Furthermore, since machine learning datasets are typically stored in text-based files, as opposed to well-known binary media format files (e.g., JPEG, MP4), VAMP further allows for *detached* manifests to be stored locally or in the cloud. Finally, VAMP also allows manifests to be stored both locally *and* in the cloud simultaneously.

Detached Manifests: Detached Manifests are manifests which are stored separately from the machine learning objects themselves. Datasets are not implemented using standard structured text or binary formats, and similarly, current machine learning models trained with PyTorch or TensorFlow have a prescribed model format. In both cases, the manifest can be created and uploaded to the *VAMP Service* after the dataset has been created or the model has been trained.

Detached Manifests can either be *Detached Local Manifests* or *Detached Cloud Manifests*. Detached Local Manifests are stored in the same directory structure (e.g., in the same directory) whereas Detached Cloud Manifests are stored in the cloud and uploaded or read using the VAMP web service.

Embedded Manifests: Embedded Manifests are inserted into the object files which allows them to be easily authenticated. Unlike media which relies on standard encoding formats (e.g., JPEG, MP4), machine learning datasets are not typically stored using a standard format. On the other hand, machine learning models are usually stored using a format defined by the machine learning framework (PyTorch, TensorFlow, scikit-learn). We anticipate that as cryptographic-based provenance solutions become more ubiquitous, it is possible that a current dataset or model structure can be extended, or new structures can be created, to allow the inclusion of a manifest.

Manifests Stored in Multiple Locations: This case includes storing the manifest in two locations, locally and in the cloud. For example, a machine learning object's manifest can be stored in a Local Detached Manifest file in addition to being stored as a Detached Cloud Manifest in the *Manifest Database*. Since media is typically stored as self-contained *binary* files, AMP only supported *Embedded Manifests* or *Detached Cloud Manifests*, where embedding manifests in the

file is the preferred method for media. Thus, the original AMP design did not consider storing the manifest in both locations.

Detached Local Manifests. As noted above, a machine learning system may consist of a number of different machine learning related files such as the source code for training or inference, separate training, validation, and test datasets and the trained model itself. Furthermore, each dataset may be separated into two different files for the data and the labels. As such, a directory structure may contain many different *Detached Local Manifest* files. If the original media object file can be modified, a reference to the manifest file can be included using the ManifestLocator in Table II. However, it may not be possible to add this additional metadata to the original media object file. In this case to allow the user to quickly associate the manifest file and the machine learning object file when storing the manifest as a *Detached Local Manifest*, VAMP requires the following naming convention. If a machine learning object is stored as *name.xxx*, then the manifest is stored in the same directory as *name.xxx.man*. For example, the training set might be stored as *data/training.csv*, and the manifest would then be stored as *data/training.csv.man*.

Serialization. As noted in Section II, AMP serializes manifests in two ways using both canonicalized JSON and canonicalized CBOR. Since media formats are binary, using the CBOR binary serialization format for manifests is a natural fit. However, for text-based datasets typically encountered in machine learning systems, CBOR serialization for Embedded Manifests would result in binary data being inserted into a file which otherwise consists of text. Thus, VAMP allows for the option of serializing the manifests using either JSON for text-based datasets or CBOR for datasets implemented in a binary format. Binary arrays are Base64 encoded before serialization.

Manifests are signed by the machine learning object creator to ensure that the machine learning object has not been modified from its original version. Canonicalized JSON manifests are signed using a JWS while canonicalized CBOR manifests are signed using COSE.

Verification. Before training or inference, the signed manifests of each required machine learning object must be first verified. For training, the manifests for the training and validation datasets and the training software components are verified using the signer's public key. If the media object is signed using COSE, the manifest is Base64 decoded to reveal its authenticated contents. The signer's public key may be different for each machine learning object. Similarly, the dataset, software components and model can also be verified during inference using the same process.

VIII. Trust Model

As mentioned previously, AMP uses X.509 to implement its trust model. Similarly, VAMP also employees X.509 and certificate authorities (CAs) to establish the authenticity of machine learning datasets, software, and models. A dataset creator or a model trainer's certificate is used to sign the machine learning object's manifest in VAMP. The certificates can be issued by any standard CA.

In general, determining the identity of the entity who signed a media object is much easier for VAMP users than for AMP users who are trying to confirm the identity of the person or organization who published a media object. The main reason is because machine learning objects are much less prevalent than all of the media objects that are available on the Internet.

We have not identified any changes needed for AMP's trust model to enable the machine learning scenario addressed by VAMP.

IX. Provenance Ledger

A ledger provides additional security guarantees over a simple signature. AMP uses a provenance ledger to further ensure the veracity of media, and this ledger is implemented using the Confidential Consortium Framework (CCF) [14]. Since VAMP is an extension of AMP, it also employs CCF to help ensure the integrity of the machine learning system by providing a public audit trail of the media objects that were either used to train the system or to evaluate new data. One particular advantage is that the ledger preserves the order of publication of different objects. In this case, digital signatures require a local clock (which may be adversarially set) or need to use a trusted timestamp authority (TSA).

When the manifest for the machine learning object is uploaded to the cloud service, the entire manifest or its cryptographic hash would also be written to the ledger. CCF provides a receipt which can be used to ensure the authenticity of the machine learning object without the need to query the ledger itself. Examples of the performance for storing the manifest in the ledger, as well as writing and reading long videos to the Manifest Database, are provided in [11] for a number of different Azure data center configurations.

X. Related Work

Provenance solutions have been previously proposed for data and machine learning systems. One early work which frames the need for data provenance is [19]. A blockchain solution was proposed for data provenance in the IoT setting [20]. Provenance systems [9], [10] have been previously proposed for machine learning systems against data poisoning attacks. However, unlike VAMP, neither system used cryptographic hashes to ensure the integrity of the data and software.

VAMP is also related to creating reproducible machine learning systems, debugging machine learning systems and incorporating explainability in machine learning systems. One example of a reproducible machine learning framework is dagger [15]. Provenance has been used for other reproducible machine learning environments including [21], [22]. Principles related to provenance, reproducibility, and FAIR data are given in [16]. Another framework for debugging machine learning system is given in [23]. An interactive and explainable framework for machine learning is proposed in [24].

Provenance systems have also been proposed for authenticating media. Since VAMP is an extension of AMP, it is most similar to that system [11]. In this work, we describe the changes that are needed to use AMP in the machine

learning provenance and authentication scenario. The Content Authenticity Initiative [13] is also similar to VAMP although, like AMP, CAI targets the authentication of media. Truepic is using provenance to provide a image provenance service for the insurance industry [25]. Amber has also built a system for using provenance for video [26]–[28]

The protection of the machine learning software is related to previous work in the protection of the software bill of materials (SBOM) and the software supply chain. SPDX [29] has been proposed as a way to protect the software bill of materials. In addition, in-toto [30], [31] provides a system for protecting the entire supply chain.

XI. Conclusion

Authentication and provenance play key roles in preventing poisoning attacks in secure machine learning systems. We find that with minor modification, the AMP system, which has previously been proposed for the authentication and provenance of distributed media objects, can be extended to VAMP and used to protect machine learning datasets, software, and models. VAMP's provenance features allow all of a machine learning's subcomponents to be discovered and verified. Provenance is particularly important for complex machine learning systems which require the discovery and verification of smaller machine learning subcomponents. As a result, a model creator can verify that the training and validation sets used during training have not been altered. Similarly, the user of a model can also authenticate both the trained model and the evaluation set, if a manifest has also been generated for these machine learning objects during inference. This work provides key requirements for standards such as the Coalition for Content Provenance and Authenticity (C2PA) [18] to be extended to protect machine learning systems.

References

[1] S. Alfeld, X. Zhu, and P. Barford, "Data poisoning attacks against autoregressive models," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, Feb. 2016. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/10237

[2] B. Li, Y. Wang, A. Singh, and Y. Vorobeychik, "Data poisoning attacks on factorization-based collaborative filtering," *arXiv preprint arXiv:1608.08182*, 2016.

[3] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, "Targeted backdoor attacks on deep learning systems using data poisoning," *arXiv preprint arXiv:1712.05526*, 2017.

[4] P. W. Koh, J. Steinhardt, and P. Liang, "Stronger data poisoning attacks break data sanitization defenses," *arXiv preprint arXiv:1811.00741*, 2018.

[5] Y. Wang and K. Chaudhuri, "Data poisoning attacks against online learning," *arXiv preprint arXiv:1808.08994*, 2018.

[6] Y. Dong, H. Su, B. Wu, Z. Li, W. Liu, T. Zhang, and J. Zhu, "Efficient decision-based black-box adversarial attacks on face recognition," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 7706–7714.

[7] K. Liu, B. Dolan-Gavitt, and S. Garg, "Fine-pruning: Defending against backdooring attacks on deep neural networks," in *Research in Attacks, Intrusions, and Defenses*, M. Bailey, T. Holz, M. Stamatogiannakis, and S. Ioannidis, Eds. Cham: Springer International Publishing, 2018, pp. 273–294.

[8] R. Wang, G. Zhang, S. Liu, P.-Y. Chen, J. Xiong, and M. Wang, "Practical detection of trojan neural networks: Data-limited and data-free cases," *arXiv preprint arXiv:2007.15802*, 2020.

[9] N. Baracaldo, B. Chen, H. Ludwig, and J. A. Safavi, "Mitigating poisoning attacks on machine learning models: A data provenance based approach," in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, ser. AISec '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 103–110. [Online]. Available: https://doi.org/10.1145/3128572.3140450

[10] N. Baracaldo, B. Chen, H. Ludwig, A. Safavi, and R. Zhang, "Detecting poisoning attacks on machine learning in iot environments," in *2018 IEEE International Congress on Internet of Things (ICIOT)*, 2018, pp. 57–64.

[11] P. England, H. S. Malvar, E. Horvitz, J. W. Stokes, C. Fournet, R. Burke-Aguero, A. Chamayou, S. Clebsch, M. Costa, J. Deutscher, S. Erfani, M. Gaylor, A. Jenks, K. Kane, E. Redmiles, A. Shamis, I. Sharma, J. C. Simmons, S. Wenker, and A. Zaman, "Amp: Authentication of media via provenance," *arXiv preprint arXiv:2001.07886*, 2020.

[12] P. Origin, "Protecting trusted media," https://www.originproject.info/, 2020.

[13] C. A. Initiative, "Creating the standard for digital content provenance," https://contentauthenticity.org/, 2020.

[14] Microsoft, "Ccf: A framework for building confidential verifiable replicated services," https://github.com/microsoft/CCF/blob/master/CCF-TECHNICAL-REPORT.pdf, 2019.

[15] M. Paganini and J. Z. Forde, "dagger: A python framework for reproducible machine learning experiment orchestration," *arXiv preprint arXiv:2006.07484*, 2020.

[16] S. Samuel, F. Löffler, and B. König-Ries, "Machine learning pipelines: Provenance, reproducibility and fair data principles," *arXiv preprint arXiv:2006.12117*, 2020.

[17] M. Alfarra, S. Hanzely, A. Albasyoni, B. Ghanem, and P. Richtarik, "Adaptive learning of the optimal mini-batch size of sgd," *arXiv preprint arXiv:2005.01097*, 2020.

[18] C. for Content Provenance and Authenticity, "C2pa technical specifications," 2021. [Online]. Available: https://c2pa.org/public-draft/

[19] P. Buneman, S. Khanna, and T. Wang-Chiew, "Why and where: A characterization of data provenance," in *Database Theory — ICDT 2001*, J. Van den Bussche and V. Vianu, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 316–330.

[20] H. Olufowobi, R. Engel, N. Baracaldo, L. A. D. Bathen, S. Tata, and H. Ludwig, "Data provenance model for internet of things (iot) systems," in *Service-Oriented Computing – ICSOC 2016 Workshops*, K. Drira, H. Wang, Q. Yu, Y. Wang, Y. Yan, F. Charoy, J. Mendling, M. Mohamed, Z. Wang, and S. Bhiri, Eds. Cham: Springer International Publishing, 2017, pp. 85–91.

[21] R. Souza, L. Azevedo, V. Lourenço, E. Soares, R. Thiago, R. Brandão, D. Civitarese, E. V. Brazil, M. Moreno, P. Valduriez, M. Mattoso, R. Cerqueira, and M. A. S. Netto, "Provenance data in the machine learning lifecycle in computational science and engineering," *arXiv preprint arXiv:1910.04223*, 2019.

[22] D. Xin, H. Miao, A. Parameswaran, and N. Polyzotis, "Production machine learning pipelines: Empirical analysis and optimization opportunities," *arXiv preprint arXiv:2103.16007*, 2021.

[23] R. Lourenço, J. Freire, and D. Shasha, "Debugging machine learning pipelines," *Proceedings of the 3rd International Workshop on Data Management for End-to-End Machine Learning - DEEM'19*, 2019. [Online]. Available: http://dx.doi.org/10.1145/3329486.3329489

[24] T. Spinner, U. Schlegel, H. Schäfer, and M. El-Assady, "explainer: A visual analytics framework for interactive and explainable machine learning," *IEEE Transactions on Visualization and Computer Graphics*, vol. 26, no. 1, pp. 1064–1074, 2020.

[25] Truepic, "Photo and video verification you can trust," https://truepic.com, 2019.

[26] Amber, "Instilling trust into video," https://app.ambervideo.co/, 2019.

[27] L. H. Newman, "A new tool protects videos from deepfakes and tampering," https://www.wired.com/story/amber-authenticate-video-validation-blockchain-tampering-deepfakes/, 2019.

[28] R. Hodgson, "Preserving video truth: an anti-deepfakes narrative," https://www.youtube.com/watch?v=zR-V1nf-dT0, 2020.

[29] SPDX, "The software package data exchange (spdx)," https://spdx.dev/.

[30] in toto, "A framework to secure the integrity of software supply chains," https://in-toto.io.

[31] S. Torres-Arias, H. Afzali, T. K. Kuppusamy, R. Curtmola, and J. Cappos, "in-toto: Providing farm-to-table guarantees for bits and bytes," in *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, Aug. 2019, pp. 1393–1410.