Characterizing the Internet Host Population Using Deep Learning: A Universal and Lightweight Numerical Embedding

Armin Sarabi
University of Michigan
Ann Arbor, MI, USA
arsarabi@umich.edu

Mingyan Liu
University of Michigan
Ann Arbor, MI, USA
mingyan@umich.edu

ABSTRACT

In this paper, we present a framework to characterize Internet hosts using deep learning, using Internet scan data to produce numerical and lightweight (low-dimensional) representations of hosts. To do so we first develop a novel method for extracting binary tags from structured texts, the format of the scan data. We then use a variational autoencoder, an unsupervised neural network model, to construct low-dimensional *embeddings* of our high-dimensional binary representations. We show that these lightweight embeddings retain most of the information in our binary representations, while drastically reducing memory and computational requirements for large-scale analysis. These embeddings are also *universal*, in that the process used to generate them is unsupervised and does not rely on specific applications. This universality makes the embeddings broadly applicable to a variety of learning tasks whereby they can be used as input features. We present two such examples, (1) detecting and predicting malicious hosts, and (2) unmasking hidden host attributes, and compare the trained models in their performance, speed, robustness, and interpretability. We show that our embeddings can achieve high accuracy (>95%) for these learning tasks, while being fast enough to enable host-level analysis at scale.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; • **Networks** → **Network measurement**; *Network security*;

KEYWORDS

Host Embedding, Machine Learning, Network Measurement

ACM Reference Format:
Armin Sarabi and Mingyan Liu. 2018. Characterizing the Internet Host Population Using Deep Learning: A Universal and Lightweight Numerical Embedding. In *2018 Internet Measurement Conference (IMC '18), October 31-November 2, 2018, Boston, MA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3278532.3278545

1 INTRODUCTION

In this paper, we develop a framework to characterize the Internet host population by leveraging advances in both network scanning and machine learning. Specifically, we train a deep neural network model using global Internet scan data; this model then generates *embeddings* of Internet hosts, i.e., features defined over the input data, whose values, when computed over a given host's information, capture the main characteristics of said host. Each embedding is a vector representation of its respective host; in this sense this framework may be viewed as a host-to-vector transformation (referred to as "Host2vec" in Figure 1), similar in concept to the Word2vec [39] algorithm. There are three distinct properties of these embeddings: (1) they are *numerical*, as opposed to the structured text response of scanned protocols; this means they can be used out-of-the-box to perform quantitative studies, (2) they are *lightweight*, much lower in size compared to the raw data, and (3) they are *universal*, meaning that they are generated without a specific target application, and are adaptable for a diverse set of tasks.

These properties lead to a scalable and versatile analysis framework. We will show that even though the embeddings are low-dimensional, they retain most of the information contained in raw measurements; this also suggests that Internet scan data is *compressible* or *sparse* in nature. Due to the embeddings' universality, they capture attributes across many aspects and protocols, resulting in highly adaptive representations that can be applied to a variety of applications, including the detection of (actively or potentially) malicious hosts, inferring unobserved attributes (e.g., the installed web server product), and quantifying host similarities.

Our framework can be used on any network scan data set to add machine learning capabilities; in this paper, to develop and evaluate our framework, we tap into Censys [8], a large database of parsed information obtained by regular scans on 20 different ports, in addition to geolocation and ownership information. One of the challenges we encounter in applying machine learning algorithms to host data is the need to first extract a numerical representation of the available information. Hence, we will first develop a novel semi-automated tool for extracting sparse binary feature vectors from JSON documents, the format used for storing Censys records. The corresponding software has been made publicly available.[1]

We then explore the use of deep generative models, namely variational autoencoders [16, 27], for training probabilistic graphical models on unlabeled data, to encode our binary representation into meaningful low-dimensional embeddings, or numerical latent variables/features. We train multiple models for varying number of dimensions in the latent space, and show that with deep structures we can learn meaningful representations with as little as two dimensions, and rich representations for up to 50 dimensions.

These latent representations, or embeddings, can be thought of as succinct summaries of the observations such as location, type of

[1] https://github.com/arsarabi/jsonvectorizer, https://github.com/arsarabi/vae.

IMC '18, October 31-November 2, 2018, Boston, MA, USA

ACM ISBN 978-1-4503-5619-0/18/10…$15.00
https://doi.org/10.1145/3278532.3278545

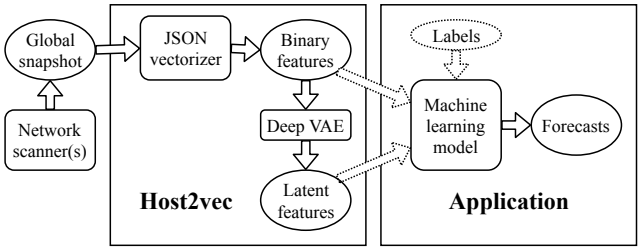


Figure 1: The analysis framework. Dashed arrows/blocks indicate that a supervised or unsupervised model can use binary, latent, or both types of representations.

the host (e.g., web/mail server, router, and so on), or misconfigurations such as open ports or expired certificates. A machine learning algorithm can then take these summaries to perform specific classification tasks. We further discuss how the observations themselves are factored into the classification output in Section 7.

Our framework is illustrated in Figure 1, where our key technical contributions lie in the "Host2vec" block where we process Internet scan data and apply deep learning techniques to obtain numerical representations, or feature vectors, from Internet hosts. Furthermore, the "Application" block contains studies highlighting the utility of this framework.

While many studies on analyzing Internet hosts tend to be driven by specific applications and focus on a small set of properties, the techniques presented in this paper can be used to make much more generalized statements about the Internet host population based on features acquired from various, often orthogonal, types of measurements. This allows us to gain a broad, macroscopic view of the Internet across many features. Moreover, once the numerical representations of hosts have been obtained offline, it is possible to use fast models to perform classification/inference in an online fashion. This makes it feasible to perform scalable analysis of large data sets of hosts, without the need for special tools such as distributed computing platforms.

Our main contribution in this paper is developing a highly scalable framework for host-level analysis of the Internet, depicted in Figure 1, and summarized below.

- We develop a semi-automated tool for extracting binary vector representations of host records, stored in tree-like (JSON) documents, and combine that with deep learning techniques, to capture information from global Internet scan data and to produce lightweight numerical embeddings. We show that these embeddings are able to retain most of the information contained in the original data, in a way that is extractable by machine learning algorithms.
- The universality and adaptability of these numerical representations allows them to be applicable to a wide range of studies, without the need to re-train features, or to design and collect new network probes. We demonstrate this by evaluating their utility for two supervised learning tasks: (1) detection/prediction of malicious hosts, and (2) inferring missing attributes of a host, more specifically for uncovering installed web server products.
- We also show how to leverage latent embeddings to obtain a similarity metric for hosts. We use this metric to build nearest-neighbor models, resulting in a novel technique to search for similar hosts within large collections.

Note that in this study, we use *features* or *vectors* interchangeably to refer to both our binary and latent numerical *representations* of hosts, while *embeddings* exclusively refers to latent representations.

The remainder of the paper is organized as follows. In Section 2 we go over the data sets used in our study. In Section 3 we explain in detail how we convert Censys records to binary feature vectors to be able to feed them into a machine learning algorithm, and briefly introduce the mathematical model of a variational autoencoder. In Section 4, we evaluate and visualize the obtained low-dimensional embeddings of Internet hosts. We examine and discuss applications of the proposed framework in Sections 5-7, review related work in Section 8, and conclude in Section 9.

2 DATA SETS

Our data sets consist of IP intelligence collected by Censys [8], obtained from regular global scans of the public Internet over 20 different ports, in addition to geolocation information (such as country, and approximate latitude/longitude) provided by the MaxMind GeoLite2 database [22], and Autonomous System (AS) information (including AS name, description and owner organization) provided by Merit Network [24] and Team Cymru [37]. For any given Internet-facing IP address, Censys contains a JSON record reporting all available information, including location/AS information, headers/banners, parsed certificate chains, and so on.

For this study, we use four global snapshots, collected on 7/1, 7/16, 8/1, and 8/16 of 2017; each scan includes roughly 150 million records. We concatenate and randomly sample from these snapshots to generate the different data sets used in our experiments for training/evaluation of machine learning algorithms. Unless otherwise stated, all data sets used throughout the study have been independently and randomly sampled from all four snapshots.

We also collect IP addresses that have been identified for conducting malicious actives during July and August of 2017, and discuss the effectiveness of our methodology for profiling them in Section 5. These data sets include the following.

Reputation blacklists. We collect IP addresses reported by hpHosts [13] and PhishTank [26], using the resolved IP address of URLs listed on every day during July and August of 2017; these are records on 123 823 and 15 026 unique hosts for hpHosts and PhishTank, respectively. Note that the latter is a blacklist focusing on phishing, while the former also targets ad/tracking and malware websites.

Malware passive DNS (MPDNS). This data set contains passive DNS data produced by the Georgia Tech Information Security Center's malware analysis system [10], by executing suspect Windows executables, and recording each sample's use of DNS, i.e., the domain name of each DNS query, and the resolved IP address. We further query VirusTotal [41] for every unique domain name in this data set, and extract the respective IP addresses of domains with at least one positive report as a set of malicious hosts, resulting in a curated data set containing 38 773 unique hosts.

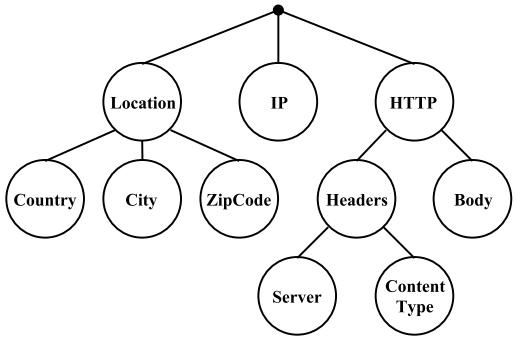


Figure 2: A sample tree-like document.

3 VECTOR REPRESENTATION OF HOSTS

To extract features from Censys, we develop a semi-automated algorithm to parse JSON documents, learn their structure, and generate binary features vectors. We then use variational autoencoders, a probabilistic graphical model, to encode compact, yet low-loss, embeddings of our binary representations. In this section, we will describe our feature extraction algorithm for tree-like documents, and briefly introduce the underlying Bayesian model in a VAE.

3.1 Learning binary representations of tree-like documents

Tree-like document models such as JSON, or XML and HTML, are used to express complex data structures containing nested fields and (possibly) polymorphic data types. In these data models, each document can be summarized as a graph, more specifically a tree; Figure 2 illustrates an example for describing hosts. Each leaf (or field) in the tree contains an attribute of the host, e.g., its origin country, while non-leaf nodes (intermediate fields) are sub-documents that encapsulate other nodes/leaves. Such data models can be used to serialize documents into human-readable text, or to facilitate programmatic data manipulation. However, learning algorithms require numerical representations of data samples, and simply applying a bag-of-words model to serialized documents fails to recognize their embedded structure.

If one already knows the schema (map) of the documents being analyzed, then a numerical representation can be constructed by transforming each field accordingly, and concatenating the resulting features to obtain a numerical array. For instance, a bag-of-words model can be applied to text fields, or categorical variables can be converted using one-hot encoding. However, constructing a schema manually can be tedious, especially for deeply nested documents; specifically, the document tree of Censys records contains more than 2400 leaves, roughly 1800 of which correspond to parsed SSL/TLS certificate chains of various protocols, including HTTPS, IMAP(S), POP3(S), and SMTP(S). Moreover, the schema has to be maintained as the data model changes as, e.g., additional ports are scanned, or parsers are modified to include more detailed attributes.

To address these challenges, we develop a novel algorithm that can automatically recognize the structure of sample documents and generate high-dimensional binary feature vectors based on a learned schema. We then feed these representations to a deep neural network model to reduce their dimensionality. Note that we maintain control over the generated features using various hyper-parameters to avoid sparse features, ignore fields that may not be useful in a machine learning setting, etc. While we develop this method for semi-automated feature extraction from JSON documents, the same concept can also be applied to other document models with a tree-like structure.

Algorithm 1 Extending a JSON schema

```
1:  function EXTEND(schema, document)
2:      schema.type ← schema.type ∪ TYPE(document)
3:      schema.counts[TYPE(document)] += 1
4:      if TYPE(document) = dict then
5:          for all (key → value) in document do
6:              EXTEND(schema.properties[key], value)
7:          end for
8:      else if TYPE(document) = array then
9:          for all item in document do
10:             EXTEND(schema.items, item)
11:         end for
12:     else if TYPE(document) ∈ {number, string} then
13:         schema.values ← schema.values ∪ document
14:     end if
15: end function
```

A typical JSON document can take a number of forms: *null* to represent a missing value, *boolean* for binary values, as well as *number*, and *string*. In addition, a JSON object can also be an *array* or a *dictionary* of (key → value) mappings, where array items, and dictionary values are themselves valid JSON objects. This definition allows for arbitrarily nested fields, such as `HTTPS.Cert.Subject.CN`, containing the common name of the subject of a certificate obtained from the TLS handshake of the HTTPS protocol on port 443. We use a simplified version of the JSON schema [14] to define and learn a map by inspecting a series of sample documents. A schema is itself a nested document that may contain any of the following (intermediate) fields.

- *type*: Defines the data type(s) that the document can take. For this paper we use the following data types: null, boolean, number, string, dictionary, and array.
- *properties*: For dictionaries, this field contains the names and schema of sub-documents, e.g., headers of the HTTP protocol.
- *items*: For arrays, the schema for array items. We assume that all items conform to the same schema.
- *counts*: We also keep track of the number of encountered sub-documents to, e.g, identify optional/sparse fields.
- *values*: For numbers and strings, we also append the list of encountered values for each field to the schema. We will later use the collected values for generating features.

To learn a schema from sample documents, we apply the recursive procedure defined by Algorithm 1 to process samples in sequence, and gradually build a schema of all encountered fields,

Section	General information			Protocol information										
	Location	Ownership (AS)	Metadata	HTTP(S) (80, 443)	IMAP(S) (143, 993)	POP3(S) (110, 995)	SMTP(S) (25, 465)	SSH (22)	CWMP (7547)	FTP (21)	Telnet (23)	DNS (53)	UPnP (1900)	SMB (445)
# of features	1471	302	74	3449	1847	1769	1105	599	199	143	118	46	33	1

Table 1: Break-down of features extracted from different sections of Censys documents. The numbers below protocol names correspond to their respective port numbers.

along with sample values. Once we have iterated over all documents, we traverse the learned schema, and define binary features for every node `X` based on its data type.

- *dict*: We define the feature "`X has property Y`" for each *optional* property in a dictionary, i.e., fields that are not present in at least one of the inspected documents.
- *boolean*: We define the feature "`X = True`".
- *number*: We bin collected values and use one-hot encoding to define the features "`X ∈ [A, B)`".[2]
- *string*: For strings, we tokenize recorded values, and define features "`X has token Y`". For categorical values, we simply use one-hot encoding for each unique value, i.e., "`X = Y`".
- *polymorphic fields*: When an object can take multiple types, we use one-hot encoding to reflect the object type as a set of binary features, e.g, "`X is dict`" or "`X is string`".[3]

We recursively apply the above rules to the learned schema over sample documents and all its sub-schema, and concatenate extracted features to construct a binary feature vector.[4] Table 2 includes descriptions for a number of the extracted features.

3.2 Fine-tuning

While the algorithm presented above can work out-of-the-box and with minimal supervision, one can further tune it, motivated by domain expertise, to extract more meaningful features. For this paper, we extract features from a data set comprised of one million Censys documents, and tune our algorithm to avoid highly sparse features, i.e, those that are observed for less than 0.05% (500) of all inspected hosts, resulting in 11 156 features. We have included the break-down of the number of features extracted from different sections of Censys documents in Table 1. Note that due to the small number of publicly discoverable SMB servers on the Internet, our algorithm is only extracting a single feature from this protocol, i.e., whether a host responds to requests on port 445.

First, we prune the learned schema by dropping fields that are present for less than 0.05% (500) of samples. We bin numerical fields (such as the latitude in Table 2) into $\min(100, n/500)$ bins, where n is the number of recorded values for said field. We also convert timestamps (e.g., certificate validity periods) to unix times, and treat the resulting value as a numerical field.

[2] Alternatively, numbers can be mapped as is to a single feature. However, this produces mixed feature types (binary and real-valued), which can be problematic when used in a learning method without proper normalization.

[3] We only encounter four polymorphic fields in Censys documents. As an example the Certificate Policies extension (https://tools.ietf.org/html/rfc5280#section-4.2.1.4) for certificates can be represented as a string containing the policy identifier, or a dictionary that combines the identifier with one or more policy qualifiers.

[4] Note that for array data types, these rules result in n separate feature vectors, where n is the length of the array for a given document. For this study, we take the logical *or* of all vectors to describe an array.

Field name	Data type	Attribute
Root	Dict	has property "FTP"
CWMP.Headers	Dict	has property "Server"
DNS.OpenResolver	Boolean	=True
Location.Latitude	Number	∈ [33.82, 34.02)
HTTPS.Cert.Validity.End	Number	∈ [2019-12-12, 2020-01-01)
Location.Country	String	= "United States"
AS.Organization	String	has token "university"
FTP.Banner	String	has token "vsftpd"

Table 2: Examples of features extracted from (intermediate) fields in Censys documents. We extract features representing a wide range of characteristics, resulting in universal representations.

For a text field, we treat it as categorical if there are less than $\min(250, n/4000)$ unique values; the second term is to avoid treating sparse fields as categorical due to the small number of collected samples, while still detecting categorical variables with many valid options, e.g., countries. For extracting tokens from non-categorical text (e.g., banners) we use two distinct case-insensitive patterns: (1) strings of length two and more consisting of alphabets and underscores, and (2) string of length two and more consisting of numbers and dots. We use the former pattern to extract words, and the latter to extract numbers and versions. For instance, the string "Boa/0.94.14rc21" taken from a sample's HTTP Server header is broken into the following tokens: "boa", "0.94.14", "rc", and "21". We further limit the number of extracted tokens from each individual field to the top 100 tokens ordered by term frequency, to avoid features being dominated by a few fields.

We do not extract features from hashes, and other fields that can act as unique identifiers, since they behave like random noise and cannot produces useful features for a machine learning algorithm. Furthermore, we ignore the IP address of the host; while one can capture the IP prefix that a host belong to by binning the IPv4 address space, it would result in tens of thousands of features even at a crude (e.g., /16) level. Nevertheless, similar information about the local network of a host is already being extracted from geolocation and ownership (AS) information. We also ignore HTTP and CWMP bodies; extracting numerical representations of webpages is out of scope for this paper and will be left for future work.[5] Finally, we exclude timestamps added by the scanner to make our representations time-agnostic, in order to prevent bias when applying trained models to future scans for prediction, as we will do in Section 5.

[5] While the HTML format corresponds to a tree-like structure, our feature extraction algorithm assumes that all documents conform to the same schema. However, a webpage can have its own unique structure, which would prevent our algorithm from extracting meaningful features.

Our algorithm results in a set of tags that can be assigned to a probed host. We have included some examples in Table 2, demonstrating how we can extract a wide range of features from various sections of host records, including open ports, reported headers, misconfigurations (e.g., allowing a DNS server to be used in amplification attacks), a host's approximate physical location and the type of organization it belongs to, the installed server products, and even granular attributes such as the validity period of certificates. The median and average number of tags associated with a host are 46.0 and 99.2 bits. For our implementation, it takes an average of 0.67 millisecond (per record) to transform documents to their binary representations on a single thread of an Intel Core i7-7770K processor, using batches of 1000 records.

The sparsity of our binary representations is due to the nature of extracted features (i.e. one-hot encoded values, and tokenized strings), and that a typical server offers only a small subset of all protocols probed by a scanner. This sparsity or compressibility directly leads to the possibility of dimension reduction which we explore next. Interestingly, the obtained binary representations were also unique identifiers for ∼17% of hosts in each snapshot.

3.3 Variational autoencoders

To extract latent features from the binary representations of hosts, we use variational autoencoders [16, 27]. VAEs provide a framework for training graphical models using deep neural networks, and can learn low-dimensional embeddings of high-dimensional data.

Artificial neural networks are built by combining multiple layers of units called neurons, or perceptrons. A single layer in a fully-connected multilayer perceptron (MLP) corresponds to the transformation $\mathbf{y} = f(\mathbf{x}\mathbf{W} + \mathbf{b})$, where $\mathbf{x}$ and $\mathbf{y}$ are the input and output vectors, $\mathbf{W}$ and $\mathbf{b}$ are the weight matrix and the vector of biases, and $f(\cdot)$ is the non-linearity, e.g., the logistic function $1/(1+e^{-x})$ for sigmoid neurons, or the rectifier function $\max(\mathbf{0}, \mathbf{x})$ for the rectified linear unit (ReLU). A MLP with multiple layers can approximate a wide range of non-linear functions, and is parametrized by $\boldsymbol{\theta}$, where $\boldsymbol{\theta}$ is the weights and biases from all layers of the MLP. The parameters $\boldsymbol{\theta}$ are then trained in order to minimize a loss function $\mathcal{L}(\cdot)$ using stochastic gradient descent.

Autoencoders are a type of MLP that can learn efficient representations (encodings) of data vectors, along with reconstructions (decodings) of these representations. Hence, an autoencoder is trained to minimize the objective function $\mathcal{L}(\mathbf{x}, \hat{\mathbf{x}})$, where $\mathbf{x}$ is the original data vector, $\hat{\mathbf{x}}$ is the reconstructed input, and $\mathcal{L}$ is a loss function for computing the reconstruction error, e.g., cross-entropy for binary data. Variational autoencoders combine ideas from deep learning and statistical inference to train graphical models that can find latent embeddings of input vectors.

Specifically, let $\mathbf{x}$ and $\mathbf{z}$ denote a vector of visible and hidden (or latent) variables, respectively. Let $p_{\boldsymbol{\theta}}(\mathbf{z})$ be the prior for $\mathbf{z}$, often assumed to follow an isotropic Gaussian distribution; $\mathbf{z}$ then drives the generation of $\mathbf{x}$ through the conditional distribution $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$. For binary $\mathbf{x}$, $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ is assumed to follow a Bernoulli distribution. Inference is performed by drawing from the posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$, given by Bayes' rule:

$$p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) = \frac{p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})p_{\boldsymbol{\theta}}(\mathbf{z})}{p_{\boldsymbol{\theta}}(\mathbf{x})} = \frac{p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})p_{\boldsymbol{\theta}}(\mathbf{z})}{\int_{\mathbf{z}} p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})p_{\boldsymbol{\theta}}(\mathbf{z})d\mathbf{z}}. \quad (1)$$

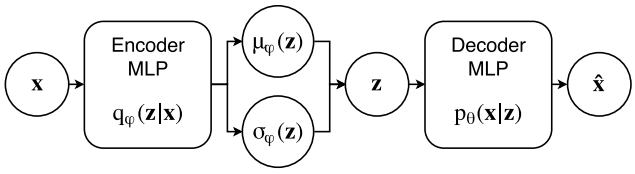


Figure 3: The overall structure of a variational autoencoder. x, z, and $\hat{\mathbf{x}}$ denote the input variables, latent embeddings, and reconstructions, respectively.

In general, the integral on the RHS of Equation (1) is intractable, and variational Bayesian methods [4] use an approximation of the true posterior given by $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$. In the context of autoencoders, $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ and $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ denote the *recognition* model (encoder), and the *generative* model (decoder), where $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$ specify the weights and biases of the encoder and decoder MLPs, respectively. A typical choice for the approximate posterior is a Gaussian distribution with diagonal covariance: $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \sim \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{x}), \boldsymbol{\sigma}^2_{\boldsymbol{\phi}}(\mathbf{x}))$, where $\boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{x})$, and $\boldsymbol{\sigma}^2_{\boldsymbol{\phi}}(\mathbf{x})$ are outputs from the encoder MLP.

Figure 3 depicts the overall structure of a VAE. The use of MLPs allows the training algorithm to learn complex non-linear transformations for the encoder (i.e., $\boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{x})$ and $\boldsymbol{\sigma}^2_{\boldsymbol{\phi}}(\mathbf{x})$) and decoder (i.e., the generated probabilities for the Bernoulli distribution, as a function of $\mathbf{z}$), resulting in a highly versatile tool for unsupervised modeling. For learning $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$, we aim to maximize the variational lower bound given by:

$$\mathcal{L}(\mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})\right] - D_{KL}\left[q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})||p_{\boldsymbol{\theta}}(\mathbf{z})\right], \quad (2)$$

which tries to minimize the error given by the cross-entropy between $\mathbf{x}$ and its reconstruction (the first term on the RHS), and penalizes deviation from the assumed prior for $\mathbf{z}$ (isotropic Gaussian) as measured by the Kullback-Leibler (KL) divergence [18] of the approximate posterior $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ from the prior $p_{\boldsymbol{\theta}}(\mathbf{z})$ (the second term on the RHS). The derivation of Equation (2) may be found in [16]. This penalty forces the recognition network to learn meaningful features by keeping representations of similar points close together in the latent space. Without this strong regularizer, the encoder might learn to *cheat* by mapping each sample to a different region in the latent space.

For the remainder of this paper, when referring to latent embeddings, we are simply taking $\boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{x})$, the mean of the Gaussian distribution produced by the encoder; we do not observe any performance benefit by including the estimated standard deviations.

4 EVALUATING HOST EMBEDDINGS

In this section we evaluate information retention properties of two types of VAE models with varying numbers of latent dimensions, and visualize two-dimensional host embeddings generated by trained models.

# of latent features	ELBO (nats)	Mean error (bits)	Percentage error				
			Mean	Percentiles			
				50	75	90	95
2	-228.0	65.5	77.2%	86.3%	94.4%	96.0%	96.7%
5	-162.7	44.5	55.9%	60.0%	79.6%	89.5%	92.3%
10	-118.5	30.2	37.1%	35.2%	54.5%	70.0%	77.3%
20	-87.1	18.2	21.6%	16.9%	32.3%	46.2%	55.0%
50	-64.6	8.1	9.9%	7.1%	13.9%	23.2%	29.6%

(a) Linear recognition/generative networks

# of latent features	ELBO (nats)	Mean error (bits)	Percentage error				
			Mean	Percentiles			
				50	75	90	95
2	-71.1	19.2	20.0%	13.1%	31.2%	50.0%	59.8%
5	-46.9	10.4	9.5%	4.5%	12.9%	26.9%	37.5%
10	-37.7	6.4	5.5%	2.4%	7.4%	15.3%	22.9%
20	-33.1	4.0	3.7%	1.7%	5.3%	10.0%	14.5%
50	-31.8	3.0	3.1%	1.1%	4.3%	8.5%	12.4%

(b) Deep recognition/generative networks

Table 3: Evidence lower bound (ELBO) and reconstruction errors for VAEs with linear (left) and deep (right) structures for encoders/decoders. The percentage error is computed by dividing the number of errors in the reconstructed vector, by the number of ones in the original binary vector. Deep models achieve significantly lower errors, especially for small numbers of latent variables, resulting in higher-fidelity embeddings.

4.1 Information retention

To evaluate the latent variable model, we compare (1) VAEs with simple linear transformations (i.e., no hidden layers) for the recognition and inference networks, and (2) VAEs with two hidden layers consisting of 1000 units each for both networks. We then compare models with linear and deep structures, to inspect how they can benefit from deep structures. For models with hidden layers, we use the ReLU non-linearity, and for the output of the decoder we use sigmoid neurons that produce an output between zero and one for the Bernoulli distribution. We also add weight normalization [30] to all layers excluding the output layer, and minimize the loss given by Equation 2 with 10 draws from the approximate posterior for training the model, using adaptive moment estimation (Adam) [15] for the stochastic gradient descent algorithm.

For training, we use a set of 10 million samples selected from Censys; we then evaluate trained models using an independent data set containing one million records. Table 3 summarizes the performance of resulting models. The ELBO can be regarded as the overall accuracy of a model, with higher values indicating a better match between the learned probabilistic model, and real-world observations. We do not observe any gain by increasing dimensionality beyond 50.

Note that for deep VAEs, using two hidden layers achieves the best accuracy for the embedded representations; using more layers leads to a sub-optimal model, possibly due to vanishing gradients and the increased number of free parameters. Furthermore, increasing the number of hidden units generally results in higher accuracy at the cost of speed. We found that 1000 hidden units achieve a good trade-off between accuracy and speed; using 2000 units improves the ELBO of 2 (50) dimensional models by 3.0% (5.3%).

To further inspect the loss associated with encoded representations, we examine the reconstruction error over our testing data set. To obtain reconstructions, we first run these samples through the encoder to find latent representations, and then use those representations to draw from the decoder. Note that since decoding relies on drawing from a Bernoulli probability distribution, it is a non-deterministic process, and each draw could result in slightly different outputs. Hence, to reduce sampling noise, we average the output probabilities over 10 draws, and then binarize these probabilities to obtain reconstructed samples. We then compute the error (i.e., hamming distance), between the original and reconstructed binary vectors, and report the average bit error and percentage error (i.e., error divided by the number of ones in the original vector) over the held-out test set.

From Table 3, we can see a clear advantage for using deep networks, especially for small numbers of latent variables. However, even with 50-dimensional embeddings, we see that a deep structure results in ~63% less reconstruction error, i.e, mean error in bits. This underlines the benefit offered by deep neural networks for learning the complex data generating distribution, and to produce high-fidelity embeddings.

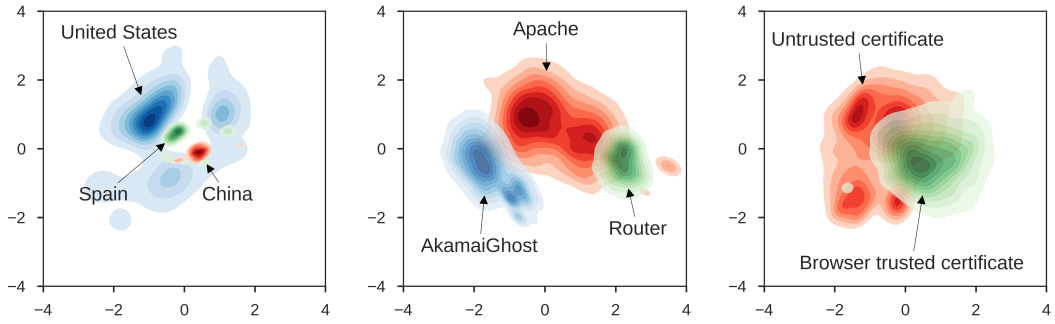
Note that principal component analysis (PCA) is an alternative technique for dimensionality reduction that utilizes linear transformations. However, PCA assumes real-valued input variables, and is thus not an appropriate model for our binary vectors. The models in Table 3a are more similar to a generalization of PCA for exponential families of probability distributions [7], which can be adapted to binary data, i.e., logistic PCA. Nonetheless, for a more direct comparison of deep and non-deep models we use VAEs for both cases, employing a variational framework with explicit assumptions on the prior for latent variables.

From Table 3b, we observe that with as low as 5-D embeddings, we can achieve an average percentage error below 10%, which further reduces to 3.0% for 50-D embeddings. This supports our claim that the underlying data is highly compressible, and that our embeddings retain most of the information of the original documents, providing a valuable resource for conducting quantitative studies.

For our implementation of a VAE in TensorFlow [38], using a Nvidia GeForce GTX-1080-Ti GPU, it takes 40 microseconds (per sample) to transform a record (find its latent embedding), and 190μs (220μs) to reconstruct a sample with a single draw (10 draws) from the decoder. These numbers are computed for the deep VAE model with 50 latent variables, and averaged over batches of 100 samples; we observe similar times for other numbers of latent variables.

4.2 Visualizing distance-based similarity

While Table 3 demonstrates the high information retention of VAE embeddings, it fails to evaluate another sought after (somewhat orthogonal) property of latent representations, i.e., their efficacy as a proxy for measuring similarity. As an example, when training



(a) Countries (b) HTTP servers (c) HTTPS servers

Figure 4: 2-D representations of hosts from different countries (left), different HTTP servers (center), and correctly configured and misconfigured HTTPS servers (right). Figure 4a uses our 2-D representations, and for Figures 4b and 4c we use PCA projections from 50-dimensional embeddings. The observed clustering of similar categories of hosts indicates that distances in the latent space can be used as a metric for similarity.

a model on hand-written digits, we expect to see samples of each digit mapped to the same region, since which digit the picture belongs to is the major deciding factor in what the picture should approximately look like, with other factors such as handwriting only contributing to minor differences. On the other hand, overfitting (especially when using higher dimensional embeddings) can lead to models that do not exhibit any particular clustering of similar samples in the latent space, while at the same time having high information retention, much like a reversible hash function. Thus in this section, we aim to examine the distance preservation of host embeddings, using heuristic criteria for similarity, i.e., proximity in physical location, or configuration.

Note that by definition, VAE embeddings resemble an isotropic Gaussian distribution for the entire host population. To inspect whether embeddings of similar hosts form more concentrated clusters, we use the originating country, different types of HTTP servers, and whether the host serves a browser trusted or untrusted (such as self-signed) certificate over the HTTPS protocol, by illustrating the distribution of hosts in latent spaces in Figure 4.

Figure 4a displays our results for three different countries, by illustrating the distribution of hosts in the latent space learned by the 2-D VAE. Note that not all samples fall within the drawn contour lines; however, the plots show where most samples are concentrated. As expected, we observe that hosts from the same country fall within close proximity of each other.

Figure 4a illustrates that when examining a subsection of the global population, points generally become concentrated into one or multiple clusters. If one needs to further examine points within these sub-populations, it makes sense to find a transformation that can re-normalize data points for creating interpretive plots. Since we are working with real-valued and well-behaved numerical vectors, fast linear transformations using PCA is sufficient for this purpose. Furthermore, since PCA can also be used for dimensionality reduction, we can train normalizers to project higher dimensional embeddings onto two dimensions by computing the top two principal components, thus leveraging the precision offered by high dimensionality. For web servers and certificates, we are essentially examining a smaller sub-population of hosts, i.e., those that are running HTTP and HTTPS services. For these sub-populations, we use PCA to project 50-D embeddings to two dimensions for visualization. The results are illustrated in Figures 4b and 4c; while the depicted clusters do slightly overlap, they show a clear difference in distributions of the encoded representations, reinforcing our claim that they provide meaningful embeddings where distances in the latent space can be used as a similarity measure.

Note that it is generally not possible to interpret the axes corresponding to VAE embeddings (or their projections) in Figure 4, since they represent a non-linear combination of binary features deemed by the training algorithm to retain the most amount of information. However, we show that the distance between hosts in this space can be used to quantify host similarities. This leads to each region in the latent space representing a certain category of hosts. We will use this property in the following sections to search for and classify similar hosts using nearest-neighbor models.

While we have only used two dimensions for visualization, machine learning algorithms can leverage higher-dimensional embeddings with much higher fidelity, as is evident from the reported errors in Table 3. In the next two sections we explore applications of these numerical representations of hosts in two concrete case studies, detecting (potentially) malicious servers and inferring installed web server products. Note that VAE models are trained independently of the examined applications, which further motivates their utility for constructing universal numerical embeddings.

5 DETECTING AND PREDICTING MALICIOUS HOSTS

In this section we use reported malicious hosts (as labels) to train a series of supervised models that assign reputation scores to arbitrary hosts, and evaluate their performance over a held-out test set from our ground-truth data. We use two types of labels that capture different types of malicious activities, the hpHosts and PhishTanks blacklists, and a set of malicious hosts curated from the malware passive DNS (MPDNS) data set, as described in Section 2.

For each of the Censys snapshots included in our study, we take hosts that have been listed after the date of the present snapshot but before the next one as the set of known malicious hosts for that period. For instance, the set of reported IP addresses during July 1-15 constitute labels for the snapshot collected on July 1. Note that by training on labels that are obtained in the future, we can build classifiers that can perform both *detection* and *prediction*, depending on whether a host is already serving malicious content when it is scanned, or if it turns malicious (or is detected for the first time) in the two-week period following a snapshot. For hpHosts, PhishTank, and MPDNS data, we obtain 401 264, 43 641, and 69 474 labeled samples, 91.8%, 94.4%, and 89.6% of which have corresponding records in Censys, respectively.

In addition to the selected malicious samples, we randomly pick 400 000 samples (100 000 from each snapshot) from Censys that have not been reported for malicious behavior, and add them to our label sets as benign samples. We then train supervised classifiers with the objective of discerning between (potentially) malicious and benign hosts. We use samples between July 1 and August 15 for training/testing on a 50/50 split. We also hold out samples from August 16-31 for evaluating the predictive performance of trained models, which we will discuss in Section 5.4.

We compare several supervised algorithms, including ensembles of decision trees, namely random forests [31] and XGBoost [6] (an implementation of gradient-boosted decision trees), and fully-connected MLPs. After tuning the hyper-parameters of these models, we found that XGBoost consistently outperformed other algorithms, and therefore our results are reported using gradient-boosted trees. We train classifiers using 100 trees, a learning rate of 0.1, and a maximum depth of 20 for each individual tree. We further regularize models by sub-selecting 50% of all training samples for each individual tree, and inspecting half of all features for each individual split. The aforementioned parameters are chosen by cross-validation. In Tables 4 and 5 we compare the performance of these different models on three curated data sets: gradient-boosted trees trained on the binary representations of hosts detailed in Section 3.1, their 50-D latent embeddings, and the concatenation of both binary and latent features; and k-nearest neighbors (k-NN) classifiers trained on 10-D latent embeddings. In the next few subsections we will examine these results in detail and highlight the advantages of the low-dimensional embeddings.

5.1 Integrating data from adjacent hosts

When using 50-D latent embeddings for tree-based methods, we also append the average of embeddings obtained from 10 *adjacent* hosts to our features, resulting in 100 features total. More specifically, we sort hosts by their respective IP address, and average latent embeddings of the five closest hosts (with records in Censys) before and after the inspected one. Note that we are using data from adjacent hosts since they typically reside in the same network, allowing us to further profile a host's owner for classification. It is worth mentioning that it would be more appropriate to average the embeddings of (a random subset of) hosts residing in the same organizational network. However, since organizations tend to own contiguous blocks of IP addresses, the utilized technique can capture the same information (with the exception of small blocks, and hosts close to network boundaries) without the need to query a separate database for network membership and boundaries. Empirically, we found that leveraging information from adjacent hosts results in a small (<0.5% in AUC), but statistically significant, performance boost. We will also show that the addition of this information allows us to build more robust models in Section 6.

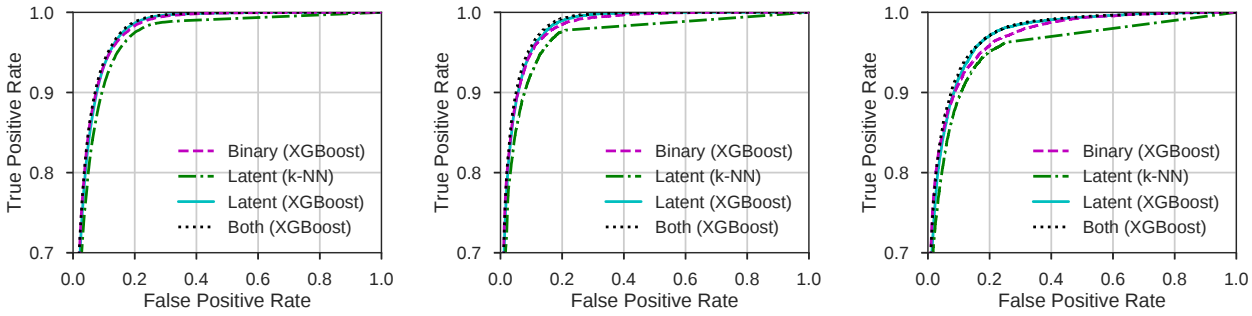
While the same approach can be applied to the high-dimensional binary vectors, averaging results in loss of sparsity, leading to representations that require more than 2.5 times memory space compared to the binary representation. We also did not observe the same performance boost to the classifiers trained on binary vectors (first columns of Tables 4 and 5) – the training algorithm cannot utilize the large number of added features without over-fitting. In contrast, for the latent embeddings this averaging provides a relatively low-cost technique to use additional data for learning tasks.

5.2 Fast and accurate classification

We compare the performance of classifiers trained on different types of representations in Figure 5 and Tables 4 and 5. The output of each model is a number between zero and one, quantifying the likelihood that a given host is malicious, or will turn malicious in the near future. We first binarize these outputs for the held-out test set using different thresholds, and plot the resulting receiver operating characteristic (ROC) curves by computing true positive rates (TPR) and false positive rates (FPR) for different operating points (corresponding to different thresholds) of each model in Figure 5. We also report a number of true positive rates (TPR) and false positive rates (FPR), the area under the ROC curve (AUC), and the speed (time to evaluate a single sample) of each model in Tables 4 and 5. The latter is computed using one thread on an Intel Core i7-7770K processor. For each statistic in Tables 4 and 5, we are reporting its mean and standard deviation over 5 different runs, since each run can result in a slightly different model due to inherent randomness in training procedures. However, ROC curves are drawn for a single trained classifier. Note that the AUC measures the overall classification accuracy by measuring how accurately a classifier can rank-order samples, i.e., the probability that a random malicious sample is scored higher than a random benign one.

We first compare the performance of gradient-boosted trees. Our results suggest that all examined tree-based models perform roughly the same in terms of overall and predictive performance. This underlines the fact that latent embeddings retain the information of their binary counterparts; note that this observation holds across all three data sets, demonstrating the robustness of these embeddings. The major distinction between these models is their speed; latent embeddings result in classifiers that are more than ten times faster than using binary features, allowing one to process an entire Censys snapshot containing ~150 million host records in less than hour on a single CPU thread. While additional processing is needed to obtain these embeddings, they are computed independent of (and can be shared across multiple) applications.

We also examine the sensitivity of our results to the number of extracted binary features in Section 3.1. Using the top 100 (1000) most frequent binary features for classification, XGBoost models achieve an overall AUC of 94.9% (97.2%), 95.0% (97.9%), and 92.6% (96.2%) for hpHosts, Phishtank, and MPDNS, respectively, indicating



(a) hpHosts (b) PhishTank (c) MPDNS

Figure 5: Receiver operating characteristic (ROC) curves of classifiers trained on different data sets.

		Binary	Latent		Both
	TPR	XGBoost	k-NN	XGBoost	XGBoost
FPR (%)	50%	0.8 ± 0.0	1.3 ± 0.3	0.8 ± 0.0	0.7 ± 0.0
	80%	3.7 ± 0.1	5.0 ± 0.1	4.1 ± 0.0	3.6 ± 0.0
	90%	7.5 ± 0.1	9.5 ± 0.1	7.8 ± 0.0	7.2 ± 0.1
	95%	12.0 ± 0.1	14.5 ± 0.1	11.9 ± 0.1	11.4 ± 0.0
AUC (%)	Overall	97.3 ± 0.0	96.2 ± 0.0	97.3 ± 0.0	97.5 ± 0.0
	Prediction	95.6 ± 0.0	93.8 ± 0.1	95.5 ± 0.0	95.8 ± 0.0
Time/sample		∼ 0.45ms	∼ 0.7ms	∼ 20μs	∼ 0.5ms

(a) hpHosts

		Binary	Latent		Both
	TPR	XGBoost	k-NN	XGBoost	XGBoost
FPR (%)	50%	0.3 ± 0.0	0.4 ± 0.0	0.3 ± 0.0	0.3 ± 0.0
	80%	2.1 ± 0.1	3.1 ± 0.1	2.4 ± 0.1	2.0 ± 0.0
	90%	5.5 ± 0.2	7.6 ± 0.2	5.7 ± 0.1	5.1 ± 0.1
	95%	9.8 ± 0.1	12.8 ± 0.3	9.9 ± 0.2	9.0 ± 0.3
AUC (%)	Overall	98.1 ± 0.0	96.8 ± 0.1	98.1 ± 0.0	98.3 ± 0.0
	Prediction	97.2 ± 0.1	95.8 ± 0.1	97.3 ± 0.1	97.4 ± 0.0
Time/sample		∼ 0.3ms	∼ 0.7ms	∼ 15μs	∼ 0.35ms

(b) PhishTank

Table 4: Performance/speed of classifiers trained on hpHosts and PhishTank data. When using latent embeddings, k-NN and XGBoost models use 10-D and 50-D vectors, respectively. AUC scores are reported over test samples from 07/01/2016 to 08/15/2017, and previously unblacklisted hosts from 08/16/2016 to 08/31/2017. Models on latent embeddings achieve similar performance to binary vectors, but at much higher speeds. k-NN models are less accurate, but provide more interpretable forecasts by associating samples with similar hosts (e.g., within the same AS) with recorded malicious activities in the past.

		Binary	Latent		Both
	TPR	XGBoost	k-NN	XGBoost	XGBoost
FPR (%)	50%	0.1 ± 0.0	0.2 ± 0.0	0.2 ± 0.0	0.1 ± 0.0
	80%	2.7 ± 0.2	4.8 ± 0.2	3.5 ± 0.2	2.9 ± 0.1
	90%	8.7 ± 0.4	11.1 ± 0.5	8.8 ± 0.4	7.7 ± 0.1
	95%	16.9 ± 0.6	20.1 ± 0.9	14.8 ± 0.6	13.8 ± 0.3
AUC (%)	Overall	97.0 ± 0.1	95.3 ± 0.2	97.1 ± 0.1	97.4 ± 0.0
	Prediction	94.8 ± 0.2	92.2 ± 0.3	94.5 ± 0.2	94.9 ± 0.1
Time/sample		∼ 0.2ms	∼ 0.7ms	∼ 20μs	∼ 0.3ms

Table 5: Performance/speed of classifiers trained on MPDNS data. Our results are consistent with Table 4.

a significant (marginal) drop in performance. While using reduced feature sets produces faster models, classifiers trained on latent embeddings are still more than three (seven) times faster compared to those using 100 (1000) binary features.

Note that while hpHosts and PhishTank focus on malicious websites, our third data set includes servers that communicate with malware; this indicates that our framework is suitable for profiling different types of maliciousness. It would also be interesting to examine whether the obtained accuracies hold for other types of malicious behavior, e.g., to detect botnet infections for identifying sources of spam or DDoS attacks.

5.3 Enabling distance-based analysis

One immediate advantage of transforming scans into vectors, either the binary representations or the latent embeddings, is that vectors lend themselves to distance based analysis, similar to what has been done in the natural language processing (NLP) domain following tools such as Word2Vec [39]. Furthermore, latent embeddings provide a better distance measure by decoupling variables; this is because metrics like the Hamming distance of binary representations could potentially recount redundant information, e.g., a host's origin country is repeated in different sections of the document.

This advantage allows us to train k-nearest neighbors (k-NN) classifiers; with each host located in a vector space, an unlabeled host can be compared to nearby labeled hosts. The search algorithm becomes prohibitive with high-dimensional vectors. Empirically we found 10-D embeddings to achieve a good trade-off between speed and performance. We thus use 10-D embeddings and train k-NN models that return the percentage of malicious hosts within the most 50 similar training samples to an inspected one as its predicted score, i.e., the likelihood of exhibiting malicious behavior.

The main observation here is that while k-NN models are not as accurate as tree-based ones, they are very close. This is indeed quite remarkable because k-NN uses a much simpler logic for classification: by inspecting similar hosts we can achieve overall accuracies of more than 95% across all data sets. Additionally, the simplicity of these models results in interpretable predictions; by associating samples with similar and labeled hosts, predictions can be accompanied with real-world examples that are known for malicious behavior, such as blacklisted websites, or malware that have attempted to communicate with similar hosts in the past. Interestingly, we observe that on average and across all three data sets, roughly 50% of the returned IPs reside in the same AS as the examined host, allowing one to compare similar hosts owned by the same entity.

5.4 Comparison with content-based techniques

For assessing the predictive accuracy of models, we evaluate their performance over reported hosts from August 16-31, that have not been listed in any of the preceding dates. For hpHosts, PhishTank, and the MPDNS data set, this results in 6860, 893, and 2013 previously unobserved hosts, respectively. In addition to hosts that have not yet turned malicious, these subsets also capture hosts that have not yet been detected. Therefore, by examining the performance of a classifier over these samples, we can evaluate its ability to predict malicious behavior before it appears in these lists. As seen in Tables 4-5, we can generally achieve a *prediction* AUC of 95% or higher.

To put the above result in context, we note that Soska et al. [36] have shown that traffic and content information can be used to forecast malicious websites, achieving an AUC between 70% and 80% over a one-year horizon on PhishTank. While a direct comparison of our methods is not possible due to non-overlapping observation windows, the large difference in performance suggests that our feature sets, which are not extracted from website content, are potentially more suitable (compared to content-based techniques) for predicting this type of risks, by capturing the configuration of a host rather than inspecting content.

In the same context, it is also worth noting that in a detection setting, content-based techniques [2, 42] tend to achieve higher accuracies than those presented in Tables 4-5; however, they are often slower than the techniques presented here. For instance, [2] achieves a FPR of 0.5% at a TPR of 90.2% on PhishTank; however, it can take 7-17 seconds to process a single sample. This naturally suggests an efficient divide-and-concur method, whereby highly suspicious samples are first identified by using the approach presented here without using the content, and are subsequently examined by more intensive content processing.

6 UNMASKING HIDDEN ATTRIBUTES OF INTERNET HOSTS

In this section we utilize our representations to infer hidden host attributes. A web server can be configured to mask the software used for serving content, which otherwise may be revealed in the HTTP protocol's *Server* header. This technique hides the architecture of the server's backend system, which in turn impedes a potential attacker from using vendor specific vulnerabilities to compromise the server. Below we show that using our framework it is possible to infer the installed application with substantial accuracy.

Server header	Token
Apache/2.4.18 (Ubuntu)	apache
Boa/0.94.14rc21	boa
GoAhead-Webs/2.5.0	goahead
lighttpd/1.4.28-devel-4975	lighttpd
Microsoft-IIS/8.5	microsoft
mini_httpd/1.19 19dec2003	mini_httpd
nginx/1.4.6 (Ubuntu)	nginx
squid/3.5.23	squid

Table 6: Example HTTP Server headers, and extracted tokens for identifying server products.

6.1 Two approaches to data masking

We manually inspect tokens that were extracted from the Server header during our feature extraction process, and recognize 23 types of web servers. We have included some example Server headers, along with the extracted tokens that allow us to identify the installed server product in Table 6. The identified products then constitute labels for training and evaluating classifiers. For the observation window of this study, we found that 84.0% of all HTTP servers include a Server header in their responses, 88.9% of which belong to one of the 23 extracted categories; this result indicates that the majority of web servers (74.7%) are reporting the product deployed on their respective machines.

We experiment with two types of data masking. In the first exercise, we select 200 000 hosts with non-empty Server headers, label and then remove *only* the Server header from all of our samples, and compute binary and latent features for the modified documents. Note that Censys parsers also extract metadata such as manufacturer and product names from various headers/banners and append them to their records. Thus, before computing embeddings, we also remove all metadata extracted from the masked header.

This first masking approach is a rather simplistic representation of reality; in practice, security-conscious administrators may hide headers and banners of other ports and protocols on the same host, in order to make the task of inference more difficult. For instance, the use of Microsoft products on other protocols could suggest their deployment on the inspected one. For this reason, we also employ a second approach by masking all headers/banners (as well as metadata) from all the probed protocols, to obtain a series of examples that mimic a more meticulous masking strategy.

In each case, we train classifiers to infer the aforementioned categories of web servers, using an even split for training/testing. To make estimators robust to both types of data masking, we train them on both sets of samples, and report their performance on each set, separately. We continue to use data from adjacent hosts in our latent feature set. To minimize target leakage due to adjacent hosts that are controlled by the same administrator, we also apply the same masking technique as an examined host to its neighbors.

6.2 Robust inference

Table 7 displays our results. We report the overall accuracy of trained classifiers, as well as their true positive rate (TPR), false positive rate (FPR), and precision (PRC) for the four most frequent

		Binary	Latent		Both
		XGBoost	k-NN	XGBoost	XGBoost
Apache (30.1%)	TPR	97.9 ± 0.1	87.9 ± 0.3	95.8 ± 0.2	98.0 ± 0.1
	FPR	1.5 ± 0.0	9.2 ± 0.2	4.9 ± 0.1	1.7 ± 0.0
	PRC	96.5 ± 0.1	80.5 ± 0.3	89.5 ± 0.2	96.1 ± 0.1
Microsoft (14.6%)	TPR	96.9 ± 0.2	89.7 ± 0.3	93.7 ± 0.3	97.2 ± 0.2
	FPR	0.4 ± 0.0	2.2 ± 0.1	1.0 ± 0.0	0.4 ± 0.0
	PRC	97.9 ± 0.1	87.5 ± 0.3	93.9 ± 0.2	97.9 ± 0.2
NGINX (14.5%)	TPR	96.9 ± 0.1	66.9 ± 0.6	86.7 ± 0.2	96.8 ± 0.1
	FPR	0.6 ± 0.0	3.0 ± 0.1	1.3 ± 0.0	0.6 ± 0.0
	PRC	96.7 ± 0.1	79.1 ± 0.4	91.8 ± 0.2	96.6 ± 0.2
Lighttpd (2.3%)	TPR	88.1 ± 0.5	72.5 ± 0.8	80.4 ± 0.6	87.8 ± 0.7
	FPR	0.1 ± 0.0	0.4 ± 0.0	0.2 ± 0.0	0.1 ± 0.0
	PRC	95.7 ± 0.4	82.2 ± 0.2	92.3 ± 0.8	95.3 ± 0.2
Accuracy		97.9 ± 0.0	86.3 ± 0.1	93.9 ± 0.1	97.8 ± 0.0
Time/sample		~ 0.5*ms*	~ 0.35*ms*	~ 0.25*ms*	~ 0.6*ms*

(a) Masked HTTP Server header

		Binary	Latent		Both
		XGBoost	k-NN	XGBoost	XGBoost
Apache (30.1%)	TPR	86.3 ± 0.4	80.8 ± 0.3	87.5 ± 0.3	89.4 ± 0.2
	FPR	12.0 ± 0.4	12.3 ± 0.3	9.1 ± 0.1	7.9 ± 0.2
	PRC	75.7 ± 0.6	74.0 ± 0.5	80.6 ± 0.3	83.0 ± 0.4
Microsoft (14.6%)	TPR	77.6 ± 0.5	73.5 ± 0.9	83.9 ± 0.3	84.8 ± 0.5
	FPR	4.5 ± 0.1	4.4 ± 0.2	3.1 ± 0.1	2.8 ± 0.1
	PRC	74.6 ± 0.5	73.9 ± 0.6	82.5 ± 0.3	84.0 ± 0.3
NGINX (14.5%)	TPR	68.0 ± 1.0	59.5 ± 0.7	73.1 ± 0.5	77.5 ± 0.6
	FPR	2.6 ± 0.1	4.4 ± 0.1	3.1 ± 0.1	2.6 ± 0.1
	PRC	81.7 ± 0.6	69.5 ± 0.5	79.9 ± 0.3	83.7 ± 0.4
Lighttpd (2.3%)	TPR	60.8 ± 0.9	59.3 ± 1.0	65.4 ± 0.6	68.2 ± 0.6
	FPR	0.2 ± 0.0	0.7 ± 0.0	0.3 ± 0.0	0.2 ± 0.0
	PRC	89.8 ± 0.6	68.1 ± 1.5	84.8 ± 1.3	87.1 ± 0.8
Accuracy		80.9 ± 0.1	77.0 ± 0.1	83.6 ± 0.1	85.2 ± 0.1
Time/sample		~ 0.45*ms*	~ 0.3*ms*	~ 0.25*ms*	~ 0.6*ms*

(b) Masking all headers/banners

Table 7: Accuracy of models for detection of web (HTTP) server products, when only the *Server* header is masked (left), or with thorough masking of headers/banners from all protocols (right). For the former, binary vectors achieve the best accuracy, while appending latent embeddings improves performance for the latter.

server products in our data set (excluding *AkamaiGhost*[6]), namely *Apache*, *Microsoft*, *NGINX*, and *Lighttpd*. For each product, we also report its frequency in our labels. Similar to the previous section, we train models on binary and latent features, as well as the combination of both. The hyper-parameters of the inspected models are the same as those used in Section 5, though we found that using trees with a maximum depth of 10 was sufficient for this experiment.

When naive masking is employed, we can infer the hidden information with high precision and recall (true positive rate), even for sparse categories (e.g., Lighttpd), with a maximum overall accuracy of 97.9%. We observe a lower accuracy of 85.2% for more thoroughly masked records. While estimators using latent embedding continue to produce the fastest models, we do not observe as boost in speed as those shown in Section 5. However, models trained on latent embeddings are more robust to thorough masking. This is due to the use of information from adjacent hosts (we are also applying the same masking strategy to adjacent hosts), eliminating these features would result in accuracies of 94.4% and 79.8% in Tables 7a and 7b. Like in the previous section, we can improve performance by combining both feature sets, at the cost of more computation. However, compared to the previous section, we notice a bigger gap between the performance of k-NN and tree-based classifiers. Nevertheless, the accuracy of k-NN models (86.3% and 77.0%) suggests that they are also robust to this type of data manipulation.

We also examine the accuracies of classifiers trained on reduced feature sets. Using the top 100 (1000) most frequent binary features, XGBoost models achieve accuracies of 79.7% (97.1%) and 63.9% (79.5%) in Tables 7a and 7b, respectively. Similar to the previous section, we observe a significant drop in performance when using 100 features, and a marginal drop with 1000 features.

[6] AkamaiGhost (Akamai Global Host) are servers belonging to Akamai's content delivery network, and are the second most frequent (23.1%) web server in our data. They receive true positive rates and precisions higher than 99% for all models in Table 7.

While we have evaluated our methodology for one case study, the same concept can be applied to other protocols, or for OS fingerprinting. Furthermore, it would be interesting to acquire independent measurements of hidden attributes, in order to evaluate how the proposed technique can generalize to hosts that are currently masking these information in-the-wild. Note that while the majority of HTTP servers report the utilized product, this is not necessarily the case for other categorizations. For instance, only 5.4% of hosts in Censys report their operating system, highlighting the importance of an independent data set for validation.

7 DISCUSSION

In this section we discuss practical advantages of the proposed framework for large-scale analysis, and briefly talk about interpretability of predictions/models.

7.1 Practical advantages

It is worth noting that the proposed framework is decoupled from the data set it is trained on, as long as samples follow the tree-like document structure detailed in Section 3, and the applications for which it is ultimately utilized. In comparison to their binary counterparts, latent embeddings take up much less storage space, allowing one to load an entire snapshot to memory for analysis. Using 16-bit floating-point format, 10-D to 50-D embeddings can be stored in 20-100 bytes (~3-14GB for a snapshot, given that a typical Censys snapshot contains ~150 million records), while using a sparse matrix format, a typical binary representation takes up ~200 bytes (~28GB for a snapshot), assuming that only indices of non-zero features are stored using 16-bit integers. For comparison, the original JSON records take up roughly 2000 bytes (~280GB for a snapshot); this number is obtained after removing the ignored

fields detailed in Section 3.2. However, even in the absence of memory/storage constraints, unlike binary/latent representations, JSON records cannot be used out-of-the-box for machine learning.

Note that for our analysis we have access to entire Censys snapshots upfront. While both Algorithm 1 and training of VAEs can be performed incrementally with streaming data, feature extraction has to be executed in one-shot and fixed before training any subsequent models. Nevertheless, once the binary representations have been fixed, VAEs can be kept up-to-date in real-time using incremental training, and supervised models can be updated or retrained periodically with recent data in order to adapt to the ever-changing Internet ecosystem.

The universality of our framework leads to representations that can be used for a wide range of applications; however, we may fail to capture rarely encountered attributes that are of particular interest, e.g., to security analysts. In such cases, one can further fine-tune feature extraction (Section 3), motivated by domain knowledge, to capture sparse, yet crucial, characteristics for specific applications, while at the same time retaining the application-agnostic features studied herein. Furthermore, errors in the underlying data (e.g., inaccurate geolocation information or errors due to IP churn) can lead to inaccurate embeddings. While any present errors have already been factored in our reported results in Sections 5 and 6 since we are validating classifiers using ground-truth labels, higher-fidelity data can in turn produce more accurate classifiers.

The scalability of the proposed framework leads to the question as to whether it could potentially be used for data collection and analysis of the IPv6 address space, where global scanning is infeasible. While we do not directly evaluate our methods over this space, the proposed techniques can be applied to measurements collected on IPv6 hosts, e.g., through targeted scanning [25]. This can be achieved by reusing IPv4 models, or training new models on a representative subset of IPv6 hosts, to make generalized statements about visible hosts in this space. The scalability of our techniques are also of particular interest in this space, as the growing number of connected machines, especially IoT devices, underscore the need for methods capable of analysis at scale.

7.2 Interpretability

To further understand how a diverse feature set allows supervised models to distinguish between different types of hosts, we have included a number of our binary features that are correlated with malicious behavior in Table 8. For each feature we are reporting its odds ratio with respect to a given data set, defined as follows.

$$\text{OR} = \frac{\Pr\{y = 1 \mid x_i = 1\}}{\Pr\{y = 1\}} = \frac{\Pr\{y = 1, x_i = 1\}}{\Pr\{y = 1\}\Pr\{x_i = 1\}}, \qquad (3)$$

where x and y denote the binary feature array and label, and i specifies a single binary feature. We evaluate these probabilities by computing the percentage of samples in our data set that satisfy the aforementioned conditions. An odds ratio higher (lower) than one indicates a higher (lower) than typical likelihood that a host exhibiting a certain feature is malicious.

The examples in Table 8 reveal how a supervised model can leverage our diverse feature set to profile and distinguish between (potentially) malicious and benign hosts. While these characteristics are not direct causes for malicious behavior, they help identify prevalent behaviors using correlational analysis. For instance, hosts residing in different countries/networks can be more or less likely to exhibit malicious behavior (interestingly, we found that Chinanet hosts are less likely to host malicous websites, but more likely to communicate with malware). Additionally, we observe that while an open FTP port is indicative of higher than typical risk, the use of a FileZilla server tends to lower this risk. Even granular attributes such as the choice of server-to-client (S2C) ciphers for the SSH protocol, or the choice of a certificate authority can reveal habits of malicious/benign hosts. Note, however, that while we are only examining univariate correlations in Tables 8, classifiers such as gradient-boosted trees can also model higher-level interactions between multiple attributes, and measure their correlation with the target labels for making more accurate forecasts. Combined with the large pool of available features, this allows us to achieve the high accuracies reported in Sections 5 and 6.

Feature	Odds Ratio		
	hpHosts	PhishTank	MPDNS
AS.Description has token "ovh"	3.72	3.16	3.21
AS.Description has token "chinanet"	0.18	0.11	1.94
Location.Country = British Virgin Islands	17.6	21.2	38.0
Location.Country = Columbia	0.05	0.13	0.05
Root has property FTP	6.04	9.15	2.27
FTP.Banner has token "filezilla"	1.14	0.57	1.65
any(SSH.S2C-Ciphers) = arcfour128	3.42	3.94	1.87
any(SSH.S2C-Ciphers) = twofish128-cbc	0.21	0.17	0.10
HTTPS.Cert.Issuer.CN has token "comodo"	9.88	14.43	3.83
HTTPS.Cert.Issuer.CN has token "entrust"	0.28	0.22	0.26

Table 8: Odds ratios corresponding to different features for profiling malicious hosts. Note that a classifier can also leverage the interaction between multiple attributes and use our full feature set to make accurate forecasts.

One drawback of the latent embeddings developed in this study is the loss of interpretability, as with virtually all deep learning models. While latent variables contain a multitude of knowledge on any host, it is unclear what each dimension in the latent space is capturing. It is up to the one using these tools to interpret the results for specific tasks; we have shown an example of how to alleviate this issue by associating hosts with similar and labeled examples using nearest-neighbors searches.

Additionally, for models trained on binary representations, we can measure the contribution of each variable to the trained model.[7] We then aggregate feature importances to obtain the importance of specific fields in Censys documents (e.g., data observed on different ports) in the decision process. In Table 9 we show a break-down of the contribution from different aspects of host configuration for three of the developed models in this paper.

Table 9 suggests that geolocation and ownership (AS) properties have a major impact on performance across all applications, indicating that hosts belonging to different regions tend to behave

[7] The technique used in the XGBoost library for generating feature importances [12], first computes the contribution of features to an individual decision tree, by iterating over all (non-leaf) nodes, and computing the contribution of the feature that is split on (i.e., reduction in node purity from the split) to the tree. Feature importances for the entire ensemble are then computed by averaging over all estimators.

Field name	Contribution			
	hpHosts	PhishTank	MPDNS	Servers
Geolocation	16.3%	16.3%	19.2%	24.2%
Ownership (AS)	11.6%	11.7%	14.2%	17.8%
Port 21 (FTP)	2.9%	3.8%	3.0%	1.3%
Port 22 (SSH)	7.2%	6.4%	7.1%	4.8%
Port 25 (SMTP)	5.1%	5.2%	3.8%	2.6%
Port 80 (HTTP)	17.8%	17.3%	17.1%	20.3%
Headers	12.4%	11.8%	11.7%	13.8%
Port 443 (HTTPS)	27.0%	23.5%	25.1%	21.9%
Certificate	19.8%	16.7%	17.3%	12.4%
Other ports	12.1%	15.8%	10.5%	7.1%

Table 9: Break-down of contributions from different fields for the examined applications. The observed spread of contributions justifies the use of a diverse feature set.

differently when it comes to the inspected applications. We also observe a high contribution from attributes of HTTP and HTTPS protocols, a large portion of which is ascribed to the headers and certificate for these protocols, respectively. This is to be expected, as the majority of our labels in Section 5 capture malicious web activity, and we are examining HTTP server products in Section 6. Data from the remaining protocols, the most notable being SSH, are minor contributors; however, overall they constitute more than 15% of contributions across all applications. This further motivates and justifies the use of a diverse feature set, including cross-protocol information, allowing models to fall back to other attributes in the absence of certain relevant fields (e.g., headers in Table 7b).

8 RELATED WORK

There has been an increasing number of studies using machine learning for various security related problems [3, 11, 34], and to process Internet measurement as well as social media data [21, 28, 29]. A supervised approach was used in [21] on Internet host measurement data to perform data breach prediction. A semi-supervised learning method was used in [28] to process social media data to identify un-reported data breaches. Vulnerability and Twitter data are analyzed in [29] to train classifiers for detecting software exploits. In virtually all these cases, features were extracted heuristically, though well-informed by domain expertise. The methodology presented here uses a systematic feature extraction, yet still guided by domain knowledge, to produce high fidelity representations for a diverse set of applications. Most importantly, our techniques retain the scalability required for large-scale analysis.

The embeddings in this study bear some resemblance to the line of work on fingerprinting, e.g., [32, 33, 40] for OS fingerprinting, [9, 17, 19] for profiling physical devices, and [5] for automated fingerprint generation. The key difference between the two is that fingerprinting studies typically focus on specific and known attributes, e.g., the installed operating system, in order to produce unique identifiers (e.g., for physical devices) or customized probes. By contrast, the deep models developed in this study are unsupervised, and the resulting embeddings are thus generic, without any target application. Furthermore, the learning is done over a much more broad and diverse set of measurements; as a consequence, the embeddings lend themselves to different types of supervised learning tasks as we demonstrated in Sections 5 and 6.

There has been a number of studies on the specific application of detecting malicious websites and intelligent phishing classification. Zhang et al. [42] examine the content of web pages, using the TF-IDF algorithm to extract information from the content and detect phishing websites. Afroz et al. [2] use content similarities in order to spot malicious sites imitating the appearance of legitimate websites. Soska et al. [36] design a classifier leveraging both content and traffic data to predict whether a given benign website will become malicious in the future. Li et al. [20], and Mekky et al. [23], find malicious websites by examining HTTP redirections. Shibahara et al. [35] propose a system for detecting drive-by-download attacks by inspecting redirection graphs of malicious, benign, and compromised websites. Abdelhamid et al. [1] use associative classification to discover correlation between blacklisted pages, and produce simple rules for phishing detection. Of these, [36] is the only study that offers prediction capabilities while all the others focus on detection. Our method for identifying malicious servers can perform both detection and prediction by comparing the recorded embeddings of a host with previously reported servers, and as shown in Section 4 it outperforms the content-based prediction developed in [36]. In general, we believe this approach can significantly complement other approaches by offering features from an orthogonal source of measurements, which are useful for the detection and prediction of malicious hosts, among other applications.

9 CONCLUSIONS

In this paper, we demonstrated a deep learning based approach to distilling an extremely broad range of characteristics of Internet hosts into universal and lightweight numerical representations, while still maintaining the richness of the high-fidelity characteristics. One major advantage of the proposed methodology is that it allows one to reuse the extracted representations of hosts for a diverse set of applications, to readily conduct quantitative studies without the need to design or collect their own features/probes, as we have demonstrated in the two practical case studies: (1) quantifying the maliciousness of hosts, and (2) inferring installed web server products. To the best of our knowledge, this is the first study on producing universal numerical embeddings, which can then be adapted for specific applications using machine learning algorithms.

ACKNOWLEDGMENTS

We thank the anonymous reviewers and Parinaz Naghizadeh for their insightful comments. This material is based on research sponsored by the Air Force Research Laboratory and DHS Office of S&T under agreement number FA8750-18-2-0011. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Research Laboratory and DHS Office of S&T or the U.S. Government. This work is also partially supported by the NSF under grants CNS-1422211, CNS-1616575, and CNS-1739517.

REFERENCES

[1] Neda Abdelhamid, Aladdin Ayesh, and Fadi Thabtah. 2014. Phishing detection based associative classification data mining. *Expert Systems with Applications* 41, 13 (2014), 5948–5959.

[2] Sadia Afroz and Rachel Greenstadt. 2011. PhishZoo: Detecting phishing websites by looking at them. In *IEEE International Conference on Semantic Computing*. IEEE, 368–375.

[3] Diogo Barradas, Nuno Santos, and Luís Rodrigues. 2018. Effective Detection of Multimedia Protocol Tunneling using Machine Learning. In *USENIX Security Symposium*. 169–185.

[4] Matthew James Beal. 2003. *Variational Algorithms for Approximate Bayesian Inference*. University of London.

[5] Juan Caballero, Shobha Venkataraman, Pongsin Poosankam, Min G Kang, Dawn Song, and Avrim Blum. 2007. FiG: Automatic fingerprint generation. In *Network and Distributed System Security Symposium*.

[6] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A scalable tree boosting system. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 785–794.

[7] Michael Collins, Sanjoy Dasgupta, and Robert E Schapire. 2002. A generalization of principal components analysis to the exponential family. In *Advances in Neural Information Processing Systems*. 617–624.

[8] Zakir Durumeric, David Adrian, Ariana Mirian, Michael Bailey, and J Alex Halderman. 2015. A search engine backed by Internet-wide scanning. In *ACM Conference on Computer and Communications Security*. ACM, 542–553.

[9] Xuan Feng, Qiang Li, Qi Han, Hongsong Zhu, Yan Liu, Jie Cui, and Limin Sun. 2016. Active profiling of physical devices at Internet scale. In *International Conference on Computer Communication and Networks*. IEEE, 1–9.

[10] Georgia Tech Information Security Center. Malware Passive DNS Data. https://impactcybertrust.org/dataset_view?idDataset=520.

[11] Hamza Harkous, Kassem Fawaz, Rémi Lebret, Florian Schaub, Kang G Shin, and Karl Aberer. 2018. Polisis: Automated Analysis and Presentation of Privacy Policies Using Deep Learning. In *USENIX Security Symposium*. 531–548.

[12] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd Edition*. Springer.

[13] hpHosts. https://www.hosts-file.net.

[14] JSON schema. http://json-schema.org.

[15] Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[16] Diederik P Kingma and Max Welling. 2013. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114* (2013).

[17] Tadayoshi Kohno, Andre Broido, and Kimberly C Claffy. 2005. Remote physical device fingerprinting. *IEEE Transactions on Dependable and Secure Computing* 2, 2 (2005), 93–108.

[18] Solomon Kullback and Richard A Leibler. 1951. On information and sufficiency. *The Annals of Mathematical Statistics* 22, 1 (1951), 79–86.

[19] Qiang Li, Xuan Feng, Lian Zhao, and Limin Sun. 2017. A framework for searching Internet-wide devices. *IEEE Network* (2017).

[20] Zhou Li, Kehuan Zhang, Yinglian Xie, Fang Yu, and XiaoFeng Wang. 2012. Knowing your enemy: Understanding and detecting malicious web advertising. In *ACM Conference on Computer and Communications Security*. ACM, 674–686.

[21] Yang Liu, Armin Sarabi, Jing Zhang, Parinaz Naghizadeh, Manish Karir, Michael Bailey, and Mingyan Liu. 2015. Cloudy with a chance of breach: Forecasting cyber security incidents. In *USENIX Security Symposium*. 1009–1024.

[22] Maxmind. GeoLite2 database. https://www.maxmind.com/en/geolite2-developer-package.

[23] Hesham Mekky, Ruben Torres, Zhi-Li Zhang, Sabyasachi Saha, and Antonio Nucci. 2014. Detecting malicious HTTP redirections using trees of user browsing activity. In *IEEE International Conference on Computer Communications*. IEEE, 1159–1167.

[24] Merit Network. https://www.merit.edu.

[25] Austin Murdock, Frank Li, Paul Bramsen, Zakir Durumeric, and Vern Paxson. 2017. Target generation for Internet-wide IPv6 scanning. In *Internet Measurement Conference*. ACM, 242–253.

[26] PhishTank. https://www.phishtank.com.

[27] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. 2014. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082* (2014).

[28] Alan Ritter, Evan Wright, William Casey, and Tom Mitchell. 2015. Weakly supervised extraction of computer security events from Twitter. In *International Conference on World Wide Web*. ACM, 896–905.

[29] Carl Sabottke, Octavian Suciu, and Tudor Dumitras. 2015. Vulnerability disclosure in the age of social media: Exploiting Twitter for predicting real-world exploits. In *USENIX Security Symposium*. 1041–1056.

[30] Tim Salimans and Diederik P Kingma. 2016. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*. 901–901.

[31] Scikit-learn. Ensemble methods: Random forests. http://scikit-learn.org/stable/modules/ensemble.html#forest.

[32] Zain Shamsi, Daren BH Cline, and Dmitri Loguinov. 2017. Faulds: A non-parametric iterative classifier for Internet-wide OS fingerprinting. In *ACM Conference on Computer and Communications Security*. ACM.

[33] Zain Shamsi, Ankur Nandwani, Derek Leonard, and Dmitri Loguinov. 2014. Hershel: Single-packet OS fingerprinting. In *ACM International Conference on Measurement and Modeling of Computer Systems*. ACM, 195–206.

[34] Mehdi Sharifzadeh, Chirag Agarwal, Mohammed Aloraini, and Dan Schonfeld. 2017. Convolutional neural network steganalysis's application to steganography. In *Visual Communications and Image Processing*. IEEE, 1–4.

[35] Toshiki Shibahara, Yuta Takata, Mitsuaki Akiyama, Takeshi Yagi, and Takeshi Yada. 2017. Detecting malicious websites by integrating malicious, benign, and compromised redirection subgraph similarities. In *IEEE Computer Software and Applications Conference*, Vol. 1. IEEE, 655–664.

[36] Kyle Soska and Nicolas Christin. 2014. Automatically detecting vulnerable websites before they turn malicious.. In *USENIX Security Symposium*. 625–640.

[37] Team Cymru. http://www.team-cymru.org.

[38] TensorFlow. https://www.tensorflow.org.

[39] Tensorflow. Vector representations of words. https://www.tensorflow.org/tutorials/word2vec.

[40] Franck Veysset, Olivier Courtay, Olivier Heen, and IR Team. 2002. New tool and technique for remote operating system fingerprinting. *Intranode Software Technologies* 4 (2002).

[41] VirusTotal. https://www.virustotal.com.

[42] Yue Zhang, Jason I Hong, and Lorrie F Cranor. 2007. Cantina: A content-based approach to detecting phishing web sites. In *International Conference on World Wide Web*. ACM, 639–648.