Quantifying Security: Methods, Challenges and Applications

by

Armin Sarabi

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical Engineering: Systems)
in the University of Michigan
2018

Doctoral Committee:

Professor Mingyan Liu, Chair
Assistant Professor Tudor Dumitraș, University of Maryland
Professor J. Alex Halderman
Associate Professor Vijay Subramanian

Armin Sarabi

arsarabi@umich.edu

ORCID iD: 0000-0002-1431-7434

ACKNOWLEDGMENTS

I am very fortunate to have had the chance to complete this dissertation under the advisement of Professor Mingyan Liu. I am grateful for your continued support and guidance; you have been a true mentor to me, both professionally and personally. I would like to express my gratitude to my colleagues Yang Liu, Parinaz Naghizadeh, Chaowei Xiao for their efforts and shared insights. It has been a privilege to meet and work with all of you. I am especially thankful to Manish Karir, for his advice and helpful discussions. I also thank my dissertation committe, Professor Tudor Dumitraș, Professor J. Alex Halderman, and Professor Vijay Subramanian for their time and valuable feedback.

This disseration would not have been possible without the various data sets that provide the backbone of our experiments. Therefore, I am grateful to the VERIS community, and the Censys team, for maintaining and making available the comprehensive databases that have been extensively used in our studies. I also thank Professor Tudor Dumitraș for sharing the WINE data set with us, and Ziyun Zhu for his efforts in data curation and cleaning.

I am lucky to have so many wonderful friends. We have shared numerous joyful experiences, and will continue to do for years to come. Last but not least, I wish to thank my parents and sister, who have provided constant encouragement and inspiration throughout my life. Even though we are apart, I feel right at home whenever I hear your voice.

The work in this dissertation was supported by the National Science Foundation (NSF), and the Department of Homeland Security (DHS).

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Data and cyber security, whether defined from the point of view of corporations, individuals, or Internet hosts/networks, have been studied from a variety of perspectives, ranging from theoretical models, to measurement studies, and data-driven approaches that combine statistical analysis and learning techniques with real-world measurements, to assess security, find flaws in current systems, and regulate more secure designs. In this dissertation, we explore the applicability of machine learning, and statistical modeling, in building algorithms that are able to make generalized statements regarding the security of real-world entities: (1) We assess the security of organizations, quantified as the likelihood of sustaining data incidents, by combining previous breach disclosures, with geographic, industry, size, and Internet traffic information, and evaluate techniques for estimating the distribution of risk among various incident categorizations, in order to guide resource allocation, and improve security policies; (2) we leverage field measurements of patch deployment on user machines, to quantify updating behaviors (resulting in a simple, single-parameter model), inspect the dynamics between software vendors and consumers, and its impact on the security posture of user machines; and (3) we develop a framework for scalable analysis of Internet-facing hosts, by distilling an abundance of knowledge obtained from global scans of the public Internet, into compact numerical fingerprints, and examine their utility for detecting (potentially) malicious hosts, inferring unobserved attributes of web servers, quantifying similarities, and characterizing networks of hosts.

Our presented techniques can be utilized by system administrators, analysts, or individuals, to make informed decisions by self-monitoring, or assessing other parties. We select/develop our tools in accordance with the requirements of the examined supervised and unsupervised tasks, and attempt to address deficiencies in the utilized data sets, to advance the efficacy of our proposed algorithms for real-world applications. In order to develop interpretable designs that can produce actionable forecasts and recommendations, we provide case studies, perform statistical tests, and inspect trained models, to further support our claims, and draw high-level conclusions based on our findings. We develop and evaluate our frameworks on (often public) data sets that are available to us, though

they can also be applied on top of other similar databases. While most of the focus of this research is on cyber and data security, we also explore applications of data-driven analysis for monitoring of Internet hosts and networks for non-security related objectives.

Thesis statement Fine-grained attributes of organizations, end-users, and hosts/networks across many aspects, can produce macroscopic perspectives for evaluating host-level and organizational risks, understanding user behaviors, and to draw high-level insights that can provide guidelines for the enhancement of security and policies.

CHAPTER 1

Introduction

1.1 Motivation and background

There have been numerous high-profile data breaches and large-scale cyber attacks in the past few years, including Equifax (made public in 09/2017) affecting 143 million consumers [78], Anthem (made public in 02/2015) affecting 80 million medical records [77], Target (made public in 12/2013) affecting 40 million payment card accounts [79], as well as the WannaCry [49] and NotPetya [48] outbreaks. Security incidents can cause service disruptions, exert significant damage to a business's assets and reputation, and compromise users' sensitive data (e.g. identity information, health records, credit card numbers, and so forth). Data incidents, and the risk they pose to end-users' privacy, underline the need for, and have motivated a wide variety of research studies. These include, e.g., understanding trends and risk forecasting (e.g. predicting data breaches, critical software vulnerabilities, and exploits) [6, 19, 37, 86, 103, 134, 141, 145], the financial impact of data incidents [2, 22, 24, 55, 64, 68], inspecting data breach litigation and disclosure laws [120, 121], economics of cyber crime and the underground market [10, 44], models for software vulnerabilities and their security implications [5, 9, 12, 14, 23, 137], empirical analysis of software vulnerabilities and their disclosure [13, 27, 45, 83, 108, 116, 129, 153], studies on exploits and zero-day attacks [18, 56, 76, 102], the patching process and human factors in security [26, 30, 50, 88, 99, 107, 142], intrusion detection and prevention [118, 122], uncovering malicious behavior (e.g. systems for detecting phishing and drive-by-download attacks) [1, 3, 85, 91, 100, 132, 152], examination of past large-scale incidents and adoption of secure technologies [11, 33, 34, 93, 115, 149, 151], studies on the emerging cyber-insurance market [69, 70, 87, 97, 109, 119, 150], security interdependence and investments [41, 42, 81, 82], and privacy-preservation of big data systems [36, 101].

Additionally, security research can also be categorized into purely theoretical work, measurement studies, and those that employ statistical analysis and machine learning, to

inspect systems and entities through real-world measurements. A prominent example of theoretical work for security includes studies on the dynamics between strategic agents, through game theoretical models. On the other hand, various projects have been dedicated to collecting data on the status and activities of real-world entities, facilitating research that focus on quantitative and predictive analysis, e.g. databases containing detailed reports of publicly disclosed data breaches [110, 113, 140, 143], software vulnerabilities and exploits [40, 105, 136], field measurements on end-user machines [31], and measurements conducted on the public Internet [8, 32]. Data-driven studies examine real-world observations in order to collect statistics and make heuristic deductions based on their findings. Moreover, another class of security research combine statistical models, with real-world measurements, in order to build detection algorithms and predictive models for the betterment of security. By fitting standard machine learning models to historical data, researchers can make generalized statements about previously unobserved phenomena, and forecast future events and trends.

Alternatively, the subject of analysis provides another aspect by which one can group security related work. These include microscopic studies on software products, the vulnerabilities and exploits that affect them, and software patching mechanisms, as well as research regarding Internet hosts/servers, and the security implications of individual (e.g. end-user) behavior. On the other hand, macroscopic studies focus on organizations/networks, often regarded as collections of individuals or hosts, in order to examine security using a more broad view point. Note, however, that these boundaries can be ambiguous, and should not be regarded as a means to decouple the presented categories as, e.g., analysis of hosts/servers can also be indicative of the behavior of individuals (i.e. administrators) that control them, and measurements on lower-level entities (i.e. hosts and individuals) are aggregated to describe and analyze organizations/networks.

1.2 Where we stand

Table 1.1 includes a tentative categorization of security related references of the current thesis, based on the aforementioned factors. Note that for most theoretical work, we have not distinguished between the subject of cited studies, as proposed models can often be applied to various types of entities. Motivated by recent advances in statistical learning and its applications in security and network monitoring, in the set of work presented here, we utilize available (and often public) data sets, coupled with machine learning methods and statistical models, in order to develop tools that can characterize the underlying entities, such as organization, individuals, or Internet hosts; these models can then be used for supervised

Subject \ Method	Measurements & data-driven analysis	Statistical models & machine learning	Theoretical studies
Organizations & networks	[64, 119, 145]	[2, 22, 24, 37, 42, 55, 68, 86, 118, 120, 121, 141], Chapters 2, 4.5.3	[36, 41, 69, 70, 81, 82, 87, 97, 109, 150]
Individuals & end-users	[88, 99, 101, 142]	[50, 141], Chapter 3	
Internet hosts/servers	[11, 33, 34, 56, 93, 100, 115, 149, 151]	[1, 3, 85, 91, 132, 134, 137, 152, 153], Chapter 4	
Software, vulnerabilities, & exploits	[12, 18, 26, 27, 30, 45, 76, 99, 102]	[5, 6, 9, 13, 19, 50, 83, 103, 107, 108, 116, 122, 129], Chapter 3	[14, 23]

Table 1.1: Categorization of security research, and the presented studies in this dissertation.

learning tasks, including risk assessment, and inference of unobserved attributes, as well as to draw high-level conclusions that can guide the betterment of security, e.g. to monitor one's assets, and prioritize resource allocation. Hence, our main contributions (highlighted in blue) belong in the third column in Table 1.1. Based on the complexity of the task at hand, and the volume of our data sets, we employ a variety of tools and techniques, ranging from simple statistical tests, to training and evaluation of deep neural networks incorporating millions of parameters. The resulting models can then offer efficient and automated monitoring solutions that can be leveraged by security analysts, network administrators, or even individuals with limited security background, in order to make more informed decisions regarding their security, or assess the posture of other parties.

More specifically, in Chapter 2 we combine publicly available information about businesses, with labels obtained by manually inspecting previously reported data breaches, to train and evaluate a set of classifiers for risk assessment, i.e. forecasting the risk of experiencing a data breach. In this context, the use of granular attributes of businesses (such as industry, size, and web traffic information), distinguishes our work from existing literature by providing more accurate risk forecasts, as well as the breakdown of risk in multiple categorizations (e.g. type of the data breach, and the actor and assets involved in the incident) for producing actionable security recommendations.

In Chapter 3, we utilize field measurements of patch deployment, in order to measure and model end-user behavior in applying software patches, examine the dynamics between users and software vendors, and study its implications on the security posture of user machines. In contrast to most existing work in this domain, we aim to study software vulnerabilities from the perspective of individual users, rather than individual vulnerabilities, in order to examine the security posture of end-hosts over long observational periods spanning many public disclosures of software susceptibilities..

In Chapter 4, we develop a framework for obtaining numerical representations, or fingerprints, of Internet hosts, based on information gathered from global scans of the public Internet, and use the proposed framework for quantifying hosts similarities, detecting malicious IP addresses, inferring masked attributes of web servers, and characterizing networks (collections) of hosts. Common techniques for characterizing web servers (e.g. for phishing detection [1, 3, 134, 152]), or networks of IP addresses (e.g. for risk forecasting [86]) rely on features that are extracted heuristically, though well-informed by domain expertise. Furthermore, existing approaches to host fingerprinting, e.g. for detecting the installed operating system [130, 131, 146], rely only on a single or few select probes due to their intrusive nature. Our proposed techniques in this area can be applied on top of existing probes across many ports/protocols for semi-automated feature extraction, in order to build comprehensive representations of Internet hosts, that can be shared and applied to any of the aforementioned applications using state-of-the-art machine learning techniques.

A major distinction between common machine learning research, and its utility for security analysis, is the need for actionable and interpretable predictions. For instance, in tasks such as handwriting or image recognition, the objective is solely for the learning algorithm to be able to correctly mimic what a human can naturally achieve. However, in the context of security, simply building black-box models is not sufficient, as one cannot readily infer why a model is making a certain prediction, and how it can interpreted to create actionable recommendations for the subject of the analysis. Furthermore, a thorough understanding of the underlying data can prevent using a skewed/biased data set that inhibits the resulting model from generalizing to real-world examples. To this end, for each of the presented studies, we use a combination of case studies, statistical tests, and inspection of trained models, in order to reinforce our claims, and provide further insight into the utility of our techniques for real-world applications.

Moreover, the proposed tools in this dissertation are decoupled from the data sets they are evaluated on. As long as one has access to databases containing similar measurements to the ones employed in our studies, i.e. exhibiting the same level of fidelity, our techniques can be applied to obtain similar results. More importantly, the continuous attempts by malicious entities to discover and exploit security vulnerabilities, and the ensuing efforts of their targets in order to thwart said attempts, have evolved into a cat and mouse game between attackers and defenders. This property, along with the continuous advent of new technologies and software, have created ever-changing ecosystems, underlining the need for techniques that can keep up with shifts in the underlying data sets, e.g. prominent attack vectors, individual behavior, and configuration of Internet-facing machines. By keeping data sets up-to-date, the presented techniques can detect new and altered patterns,

and utilize them to continue producing relevant predictions. Therefore, a key takeaway from this dissertation, is the importance of transparency and data sharing in the field of security. The availability of data sets containing measurements and events (i.e. labels) for various levels of entities, enables the development of objective and unbiased models that can produce valuable deductions, benefiting all benevolent parties.

1.3 Main contributions

In this section, we will provide a brief summary of the various studies presented in this thesis, followed by an overview of our key contributions, and non-goals.

1.3.1 Fine-grained data breach prediction

Security incidents regarding loss or stolen data, as well as incidents leading to service interruptions, can cause considerable financial damage to victim organizations, and tarnish their reputation. Security incidents have been extensively studied from a variety of perspectives, aiming to prevent, or alleviate the detrimental effects of data breaches, and study their implications. Our contribution in this field of security research, is demonstrating how details about a business can be correlated with their risk of experiencing a data breach. Note that a number of projects have been dedicated to collecting and indexing publicly disclosed data breaches along with details such as cause of the incident and its monetary impact, as well as available information about the victim and the actor(s) responsible [110, 113, 140, 143], allowing us to procure labels for training supervised estimators of cyber-risk. Therefore, in Chapter 2 we aim to understand if, and to what extent, business details about an organization can help to assess a company's risk in experiencing a data breach incident, as well its distribution of risk over multiple incident types, in order to provide guidelines to effectively protect, detect, and recover from different forms of security incidents. In our previous work [86], we have examined the use of attributes defined over an organization's network, such as number of blacklisted IPs, or misconfigured servers, aggregated over organizational boundaries, to assess cyber-risk. We extend this definition to include non-hacking incidents such as internal error, insider misuse, and physical theft or loss, and leverage a broad set of publicly available business details to provide a more comprehensive analysis on incidents involving any form of data breach and data loss. We use reports collected in the VERIS Community Database (VCDB) [143], as well as data from Alexa Web Information Service (AWIS) [8], the Open Directory Project (ODP) [29], and Neustar Inc. [104], to train and test a sequence of classifiers/predictors. For prediction, we use features obtained

from AWIS, including traffic information and measurements collected on an organization's domain (traffic rank, website age, speed, number of pages linking to said domain, etc.), the domain's category (e.g. business, health, and so on) from ODP, and industry and size information (employee count and network size) extracted from VCDB and Neustar.

For training estimators that can distinguish between low and high risk organizations, we use random forests [128], a supervised classification technique. Assume $i \in \{1 \dots n\}$ to represent a set of observed samples (organizations), with $\boldsymbol{x}_i$ denoting a numerical representation of the underlying samples, i.e. numerical vectors containing the aforementioned feature set. Further assume y_i to provide labels for said samples, i.e. whether a given organization has encountered a data incident. Supervised models attempt to predict labels y_i, given the observations $\boldsymbol{x}_i$. In a Bayesian setting, trained models estimate the conditional density $p(y \mid \boldsymbol{x})$, and are evaluated over a held-out test set, in order to prevent over-fitting, and to examine whether the model can generalize to previously unobserved samples. Random forests are comprised of an ensemble of decision trees, where each individual estimator is trained over samples drawn with replacement from the training set, and individual nodes (splits) in each tree are chosen among a random subset of the features; by averaging the probabilistic prediction of many trees, random forests reduce over-fitting, and produce more accurate predictions.

Our results show that the proposed feature set can distinguish between victims of data breaches, and non-victims, with a 90% true positive rate, and 11% false positive rate, making it an effective tool for characterizing businesses, for the purpose evaluating their cyber-risk. Note, however, that simply assessing the risk of a breach, fails to provide any actionable recommendations on the weakest links that might lead to a data incident, and types of assets that are at most risk. Therefore, in addition to overall risk, we also evaluate the efficacy of public information in building risk profiles, i.e. distribution of risk over different types of data breaches (i.e. hacking and targeted attacks, internal error, insider misuse, and physical theft or loss), actors (internal, external, partners), and the assets involved in the incident (media, servers, devices, etc.). We provide evidence that using a broad feature set can lead to sparse risk profiles, i.e. a small subset of incident categories with high risk, or uniform profiles, when the organization's risk is distributed evenly among all categories, depending on the organization being examined. For businesses with non-uniform risk profiles and limited security budget, this method allows them to focus on a sparser set of preventative measures, thus achieving the same level of protection by spending less resources through more judicious prioritization.

Our evaluation of the presented techniques show how public information about a company can be used to characterize them and assess their cyber-risk. As opposed to studies

that aim to understand and predict trends for the overall population [37, 145], a fine-grained analysis of individual entities enables companies to adjust their policies in accordance with their own risk. Furthermore, risk profiles provide analysts with actionable information by pointing out the weakest link in an entity's security posture, allowing administrators to tighten security by re-allocating resources for more effective self-protection, or enforcing more strict security policies. Finally, the use of public information also allows third parties, e.g. insurance underwriters, to obtain a more accurate assessment of an organization's risk in experiencing a data breach, thus resulting in more optimal cyber-insurance contracts, and techniques for vendor risk management.

1.3.2 Modeling end-user patching behavior

In Chapter 3, we study the dynamics between vendors and end-users when it comes to software vulnerabilities. Unpatched and susceptible software represent a valuable resource for attackers, exploits for these vulnerabilities can allow miscreants to control the vulnerable hosts remotely, e.g. to bootstrap a botnet or to launch distributed denial of service attacks, or to steal sensitive information such as passwords, private keys, credit card numbers, medical records, and so on. To counter these threats, software vendors create and disseminate patches for security vulnerabilities, often increasing the automation of software updating mechanisms in an attempt to accelerate the patching process and sidestep possible tardiness on the part of the end-users. Nevertheless, constant discovery of software vulnerabilities, vendors' failure in deploying patches in time, and users' delay in installing said patches, create windows of opportunity for attackers to exploit unpatched vulnerabilities. Figure 1.1 displays the life-cycle of software vulnerabilities. Vendors' failure in deploying a patch before a vulnerability is publicly disclosed, imposes a windows of susceptibility on end-hosts, which may be further extended by the amount of time it takes for a user to install the patch on their respective machine.

To understand how end-users install software patches, and to examine the implication of individual behavior on the end-host vulnerability state, we propose techniques for quantifying the user updating behavior from field measurements of patch deployment provided by the Worldwide Intelligence Network Environment (WINE) [31], from the perspective of individual users, and examine the implications of this behavior on the vulnerability state of the users' machines. We utilize auxiliary data sets including the National Vulnerability Database (NVD) [105] (to find publicly disclosed vulnerabilities of specific product versions), databases of software flaws with known exploits [40, 136], as well as software release notes from vendors or third parties, in order to find the exact release date of each

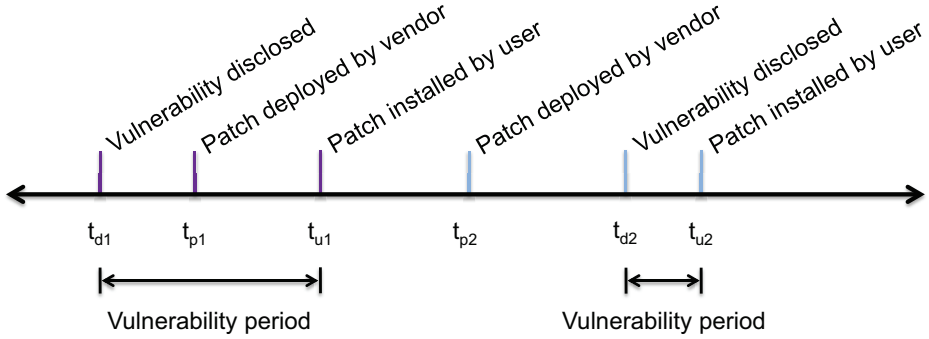


Figure 1.1: The vulnerability life-cycle

product version in our study. We measure an end-host's state (i.e. the set of installed application on a machine) at any point during the observation period from WINE, and find the exact time of installation for each software patch, which in turn results in recovering the users' delay in applying a patch from its release time by the vendor. Furthermore, we develop schemes for dealing with inconsistencies, such as irregular (non-chronological) release dates, when a vendor is developing multiple product lines in parallel, and the resulting challenges when a user installs multiple lines of the same product on their machine, allowing us to minimize the error in our measurements. We conduct these measurements for four client-side applications: Google Chrome, Mozilla Firefox and Thunderbird, and Adobe Flash Player; the resulting data set consists of 11,017,973 events over 426,031 unique hosts between 2010 and 2012.

Examining the obtained patch installation delays indicates that user behavior is fairly simple-minded, and can be summarized using a single-parameter distribution, more specifically the geometric distribution; this implies that the users' willingness to patch is memoryless, i.e. it is independent of any past decisions, and is largely consistent across consumers residing in different countries. Our findings result in a simple model for quantifying the relationship between user behavior and end-host security, and examining the vendors' role in the overall security posture of user machines; the single-parameter distribution of user behavior, suggests that we can rely on the average delay in applying software updates, for sorting users, and examining the impact of their habits on the vulnerability state of their respective machines.

Additionally, we quantify the security posture of end-hosts, by measuring the amount of time they remain vulnerable to disclosed (or exploited) software flaws. Comparing across different applications, we observe that silent updates do lead to shorter windows of vul-

nerability for end-hosts; however, even with silent updates, the majority of hosts have long windows of susceptibility, the large number of security flaws in popular client-side applications, coupled with the simple-minded behavior of end-users, limits the benefits of silent updates. Our observations imply that the rate at which vendors unintentionally introduce vulnerable code into their software, and not the speed at which they are removed, is mostly driving the vulnerability state of machines that utilize these products, revealing a drawback of a rapid release cycle.

Our main contribution in understanding software vulnerability cycles is studying them from the perspective of individual users, rather than individual vulnerabilities. Observing users over long periods spanning multiple vulnerability disclosures enables us to uncover the simplicity of user behavior in applying software updates, and allows us to measure the correlation between users' promptness in applying patches, and the security posture of their respective machines, i.e. the length of vulnerability windows. Additionally, we observe the vendor's role in making hosts susceptible to exploitation through (1) how often vulnerable code is added to the software (affected by the product's release cycle), and (2) how long vulnerable code stays on an end-host (affected by patch delivery times, the updating mechanism, and user behavior).

1.3.3 Numerical fingerprinting of Internet hosts

At any given moment, the IPv4 address space alone contains hundreds of millions of publicly accessible devices, such as web servers, personal computers, webcams, routers, and so forth. Furthermore, each individual device can be configured in a unique manner; these configurations may include open ports, the type of software used on each port for connecting to clients and serving content, and technologies used for encrypting traffic. The collection of configurations and traits associated with a device can be used to build a specific, and sometimes unique, fingerprint for an Internet host. Furthermore, with the advent of network scanners such as Nmap and ZMap [35,106], researchers can now perform global scans over the IPv4 address space to obtain hundreds of millions of such measurements, in order to compose rich data sets for studying the Internet ecosystem across a diverse set of traits. However, previous work mostly focus on specific attributes of IP addresses, for instance studying the susceptibility of hosts to critical software vulnerabilities [34], studying technology adoption rates [33], fingerprinting operating systems [130, 131, 146], or using hand-crafted features to estimate cyber-risk [86].

In Chapter 4, we explore a more generalized approach to fingerprinting Internet hosts, by generating machine learning compatible fingerprints that incorporate a wide range of

available measurements on the public Internet. More specifically, we tap into Censys [32], a large database of global scan measurements, as well as geolocation and routing information, over the entire IPv4 address space. These measurements are contained within a structured data format (JSON) for each IP address, and incorporate information gathered from any discoverable host on the Internet through periodic scans. As an example, for a host that responds to HTTP requests, headers and content of the HTTP GET response are collected; for HTTPS scans, the parsed SSL certificate and encryption configurations are also included. By combining attributes of an IP addresses over many ports/protocols, these measurements can characterize and provide a wealth of knowledge on a specific host, and can be utilized for various learning tasks. The comprehensiveness (breadth), granularity (depth), and volume (number of probed hosts) of information that is available over the public IPv4 address space, can result in models that can generalize using a diverse set of features, and can compose a high level picture of an Internet host, by understanding the underlying (and possibly abstract) factors that drive the observed attributes.

Note that documents in the Censys database correspond to a tree-like structure, i.e. parsed responses from each port are included within distinct fields, which may themselves encapsulate other fields, e.g. various headers from the HTTP response. Hence, extracting information from these documents in a way that can be fed into machine learning algorithms is a challenging task. In order to encode these structured records into numerical vectors, we develop a customizable recursive algorithm that recognizes the shared schema between all documents, and uses appropriate transformations for encoding each individual field, producing high-dimensional, yet sparse, binary representations. Our proposed algorithm results in feature vectors in the form of various tags that have been associated with an IP address, incorporating both the embedded structure of a document, i.e. the set of available measurements (e.g. open ports), as well as the corresponding values for said fields. Furthermore, our numerical representations also incorporate the context for the extracted features, e.g. the keyword *Microsoft* can have different meanings when it is observed within the description of an autonomous system (AS), or the HTTP server header; the ability to distinguish between these two cases is a useful attribute of our proposed technique. We then evaluate fingerprints comprised of roughly 10 000 features for a variety of supervised learning tasks and case studies.

Although supervised learning has the advantage of using ground-truth labels to train and evaluate the proposed models, issues arise when labels are scarce. Small training data sets limit one to using simple models that can not take into account higher order interactions between variables (features). However, with abundant unlabeled samples, one can leverage unsupervised learning in order to directly model the data set under examination.

Such techniques often attempt to learn the distribution of data points, i.e. finding the density $p(\boldsymbol{x})$ given the vector of inputs $\boldsymbol{x}$. By understanding how different features of an entity interact with each other, and learning the data generating distribution, unsupervised models can achieve tasks like creating synthetic data (i.e. generative models), dimensionality reduction, and dealing with missing data. From a topological point of view, vectors $\boldsymbol{x}$ often lie on a low-dimensional manifold in a higher dimensional space; by learning this manifold structure, unsupervised models may learn how to generate valid examples, or complete partial observations by projecting onto the space of real data vectors. Unsupervised algorithms can also be used as a pre-training stage to improve supervised techniques, by learning well-behaved features that can explain the original data points. Examples of unsupervised models include principal component analysis (PCA) [111], factor analysis (FA) [66], and neural networks such as restricted Boltzmann machines [59, 133], variational autoencoders [73, 117], and generative adversarial networks (GAN) [52].

Therefore, we explore the utility of generative unsupervised models, in order to derive a low-dimensional numerical representation, or embedding, of our high-dimensional fingerprints. More specifically, we use (1) variational autoencoders (VAE), a recent method combining ideas from deep learning, graphical models, and stochastic variational inference, that can learn low-dimensional representation of data vectors, and (2) restricted Boltzmann machines (RBM), a probabilistic graphical model for learning binary latent embeddings. Both techniques attempt to estimate the data generating distribution $p(\boldsymbol{x})$, where $\boldsymbol{x}$ denotes the set of observed variables, through finding a set of auxiliary, or latent, variables $\boldsymbol{z}$, and learning the joint probability distribution $p(\boldsymbol{x}, \boldsymbol{z})$. Additionally, by limiting the dimensionality of $\boldsymbol{z}$, both techniques can perform dimensionality reduction, by mapping the observed variables to lower dimensional embeddings. This is achieved by learning efficient encoders (representing the conditional density $p(\boldsymbol{z} \mid \boldsymbol{x})$), and decoders (approximating the density $p(\boldsymbol{x} \mid \boldsymbol{z})$) that can convert between the original and latent representations. We evaluate both models on binary fingerprints extracted from Censys records, and conclude that the use of continuous latent variables and deep neural networks, allows VAEs to outperform RBMs in constructing high precision and well-behaved numerical fingerprints. We train and evaluate a series of VAEs trained over binary representations comprised of $\sim$10 000 binary features, to find 2-50 dimensional embeddings of these records with varying degrees of accuracy. This method can also be interpreted as a non-linear dimensionality reduction algorithm, as opposed to linear models such as PCA, and can be used to speed up the resulting machine learning algorithms for analyzing IP addresses.

Note that our main goal for applying dimensionality reduction, is to reduce memory and computational requirements for processing and performing predictions on large populations

of hosts. To this end, we compare the performance of multiple techniques in terms of the loss for the encoded representations. From the numerous existing dimensionality reduction algorithms, we compare two candidates that are applicable to binary observations (i.e. the vector $\boldsymbol{x}$), and utilize real-valued and binary latent variable models, namely VAEs and RBMs, respectively. Additionally, we compare VAEs using linear transformations and deep neural network structures as encoders and decoders, and observe the efficacy of deep networks for approximating complex data generating distributions, thus resulting in higher precision embeddings compared to linear models.

Our proposed framework enables scalable analysis of Internet hosts in a machine learning setting, and the wide range of information embedded within the resulting representations can be utilized for many learning tasks. We examine a number of these applications for our numerical fingerprints, summarized as follows.

Visualization Our latent VAE representations of hosts result in well-behaved variables, following an isotropic Gaussian prior. Moreover, the distance between two samples in the latent space is a measure of their similarity, e.g. if they belong to the same geographical region, or offer the same set of services. Therefore, even though it is not easy to determine what factor each dimension in the latent space is capturing, it is possible to use the corresponding coordinates for an IP address for visualization. This can be achieved by displaying 2-dimensional embeddings, or a projecting higher dimensional representations onto two dimensions. We show that even in the case of coarse 2-D embeddings, a VAE clusters specific classes of hosts, such as hosts from different countries, close to each other in the latent space; this allows collections of examples to be readily categorized, when there is a clear clustering in a visualized data set. Consequently, this methodology offers a fast algorithm to visualize arbitrary collections of hosts, and can be used to facilitate analyzing large quantities of IP addresses, for instance by projecting hosts for easy categorization and filtering, or spotting conspicuous clusters of similar hosts within a collection, e.g. IP addresses belonging to a certain organization.

Supervised classification We further evaluate our high-dimensional binary fingerprints, along with their latent embeddings, for two supervised learning tasks, namely for classification of blacklisted malicious hosts, and inferring installed web server products. We compare various state-of-the-art supervised models, and observe that decision tree based models, specifically gradient-boosted trees [25], consistently outperform other models. The sparsity of our binary representations, allows them to be stored efficiently using sparse matrices, resulting in fast and accurate models; however, the low-dimensionality of our latent

fingerprints produce even faster models. Our results demonstrate that our framework can be used to quantify maliciousness, and to infer attributes of a host that might be masked from external observers. The latter suggests a possible security threat, as potential attackers can selectively target hosts susceptible to zero-day exploits in an automated fashion, even when the use of vulnerable software is not directly disclosed in a server's headers and banners.

Host similarity One drawback of common fingerprinting techniques, is the lack of the ability to detect similar hosts. As previously mentioned, the euclidean distance between two samples in the latent space can be used to quantify their similarity. This property, along with the low-dimensionality of VAE embeddings, enables us to use the nearest neighbors algorithm to query for similar hosts within a large corpus of IP addresses. Hence, we also compare k-NN models to gradient-boosted trees for classification, and show evidence that this technique produces similar results to state-of-the-art learning methods, establishing the utility of our latent fingerprints for fuzzy matching of hosts.

Network signatures Finally, we aggregate fingerprints of IP addresses over arbitrarily defined boundaries, i.e. autonomous systems, to obtain signatures for collections of hosts. Our previous work [86] relies on features that are extracted heuristically, though well-informed by domain expertise, to characterize collections of IPs. However, by aggregating granular fingerprints of IP addresses, e.g. through averaging, we can define numerical fingerprints of networks that are much more representative of the types of hosts they contain. This approach to characterizing networks can lead to machine learning models that are able to generalize over fine-grained attributes of hosts that compose the network. We evaluate this technique for classifying types of ASes, i.e. consumer, business, education, government, and information (hosting and content delivery networks). We show that capturing the joint distribution of hosts in a network, by averaging our binary fingerprints over AS boundaries, can produce significantly higher accuracies compared to classification at the host level. For instance, observing only a few servers in a network that utilize a `.edu` domain, can be a strong indication of an educational network. The scalability of the presented techniques, enables this form of analysis to be performed efficiently over the entire Internet, providing tools for analysis of large networks containing millions of discoverable hosts. Note that the proposed signatures can potentially be computed over partial measurements on a few target hosts, as long as observations are representative of the entire network, enabling one to perform less intrusive probes of networks, or to define signatures on IPv6 networks where global scans are not feasible.

1.3.4 Overview and non-goals

The key contributions of this dissertation, as well as the motivations behind our approach for each examined problem, can be summarized as follows.

- We present microscopic and macroscopic studies of entities' (i.e. organizations, individuals, or Internet hosts/networks) security, using appropriate learning techniques, and statistical models, for realizing the maximal value of utilized data sets.
- We choose our techniques in accordance with the following:
  - Complexity of the problem at hand, and quality/quantity of available information, e.g. fitting single-parameter distributions for inspecting user behaviors, or training deep neural networks with the availability of large databases over thousands of features.
  - Compatibility with available measurements, e.g. using tree-based supervised classifiers for binary and mixed (binary and continuous) features, and latent variable models that are applicable to binary observations.
  - We use cross-validation, by evaluating trained models on a held-out test set, when selecting the best candidate out of multiple algorithms.
- We further reinforce our findings by providing case studies, performing statistical tests, and inspecting trained models, in order to produce actionable and interpretable recommendations/predictions. This enables individuals with limited security, or machine learning background, to effectively use our proposed tools and techniques.
- When applicable, we draw high-level conclusions regarding the underlying ecosystems, e.g. the random nature of specific types of data incidents, the dynamics between vendors and end-users, and privacy implications of large data sets of global Internet scans.
- We further explore the utility of our proposed framework in Chapter 4 for non-security related applications, including visualizing collections of hosts, and quantifying hosts similarities, extending and generalizing common fingerprinting techniques.

Non-goals While our classification results in Chapters 2 and 4 uncover correlation between the observed attributes of entities, and the targeted labels (data breaches, malicious behavior, server software products), they do not necessarily denote a causal relationship. In

other words, we do not intend to reveal vulnerabilities, or detect manifestation of an intrusion, or choice of software, but rather to detect patterns and interactions between various characteristics of an entity, which can be leveraged for detection/prediction. Furthermore, we use already available (and public, in the case of Chapters 2 and 4) data sets; conducting our own measurements is another non-goal of this study. Nevertheless, we attempt to process these data sets to produce the most relevant measurements for the examined problems, and minimize noise/errors. Note that the separation between measurements and employed methodology suggests that the utilized techniques can potentially be applied to similar data sets, e.g. more comprehensive reports of data breaches, patch deployment measurement from other software products, and probes on private networks, or the IPv6 address space, extending the applicability of our studies to real-world scenarios.

1.4 Organization

The remainder of this thesis is organized as follows. In Chapter 2 (based on our published work [125, 126]), we will discuss our methodology for training a set of classifiers for quantifying the overall risk of data breach, as well as its distribution over multiple categorizations of incidents, using publicly available information about businesses, and disclosures regarding past data incidents. In Chapter 3, we will elaborate on our work [127] for distilling user patching behavior from a large corpus of patch deployment measurements, and its implications on the end-host security posture over long observation windows spanning many vulnerability cycles. In Chapter 4, we will present our framework for numerical fingerprinting of Internet hosts, providing tools for generating machine learning compatible representations of Internet-facing machines, and its applications for data set visualization, quantifying host similarities, detection of malicious hosts, inferring masked attributes of web servers, and numerical characterization of networks. We conclude and discuss possible future directions in Chapter 5.

CHAPTER 2

Fine-grained Data Breach Prediction Using Business Profiles

2.1 Introduction

Data is an important asset in every business; the valuable data of an organization may include private information such as medial records, credit card numbers, private customer data stored on the cloud, or even trade secrets, as well as public information such as the website of an online commerce company. Any incident involving such data, whether intentional (targeted attacks) or unintentional (internal errors), can disrupt a business and inflict damage on its assets and reputation. Therefore, a portion of an organization's resources should be dedicated to protecting itself from security incidents; preventive measures include maintaining regular backups, keeping software up-to-date, and employee education in order to reduce miscellaneous errors.

However, determining how to allocate resources in protecting one's assets, as well as choosing an optimal level of investment in each preventive measure, is not a trivial task, as there is a wide variety of ever-changing attack methods. To help identify common forms of data incidents, a number of projects have been created to collect information about incidents that involve some sort of data loss. Some of these projects, such as [110, 140], focus exclusively on hacking attacks, while some, e.g. [113, 143], cover a broader range of incidents, including human errors, and physical loss of data due to theft. Utilizing identifiable patterns in these reports, organizations can recognize prevalent incident vectors, and invest in self-protection in a more optimal way. However, a point that should not be overlooked is that not all businesses should be treated the same, as each business is prone to different forms of incidents. For instance, a cloud hosting company might be more likely to suffer from hacking or denial of service attacks, while a medical institution with a large number of personnel runs a relatively higher risk of data loss through human error.

In this chapter, we aim to better understand how information about a business is correlated with its risk of falling victim to different forms of data incidents. Determining the overall risk of experiencing any form of data incident will help organizations decide on an optimal level of security investment. Moreover, estimating the distribution of risk among multiple incident types (e.g. targeted attacks, miscellaneous errors, and insider misuse) will allow us to narrow down the recommendation on the most effective preventive measures, depending on the types of incidents the organization is most likely to face.

To this end, we use an incident data set collected by the VERIS community [143] reporting a broad class of data incidents; these reports consist of detailed information about the incident itself (e.g. type of data breach, and assets involved in the incident), as well as the victim organization (e.g. business sector, number of employees). Furthermore, we select a set of non-victim organizations by randomly selecting network domains from the Open Directory Project (ODP) [29]. We combine these with statistics obtained from Alexa Web Information Service (AWIS) [8] about the websites of victim and non-victim organizations, as well as information about network assets of an organization obtained from Neustar Inc. [104]. These features together constitute the *business details* of the organization. We then utilize this information to assess its overall risk of experiencing a data breach. We are able to identify, with 90% accuracy, victim organizations with the same attributes as companies that have previously experienced a breach, while maintaining a false positive rate of 11%. For victim organizations, we further estimate the conditional distribution of risk for specific incident types by considering three different categorizations for the incidents: (1) by type of data incident (e.g. error, hacking, misuse, etc.), (2) based on the source of the incident (external, internal, or partner) and the motive behind it, and (3) by considering the assets that were involved in the incident (e.g. media, servers, user devices, and so forth). Our results show that there is a clear correlation between each incident category and the victim's business details; this information can be used to provide guidelines on how an organization with limited budget for security should prioritize its security investment in allocating resources to different forms of self-protection.

In our earlier work [86] we examined the use of a different type of data, namely Internet measurement data on organizations' security posture (including malicious activities observed from hosts, and misconfiguration of Internet-facing machines), to predict future cyber-security incidents. In the present study, we broaden our scope to include not only network/cyber incidents, but also non-cyber data incidents such as miscellaneous errors, insider misuse, and physical theft and loss. This distinction warrants the use of non-network related measurements for our analysis, as network measurements are likely to present less value in forecasting non-cyber data incidents, while details regarding businesses, including

location, industry, and size, can help charactarize how business are targeted by malicious entities, and model other forms of data loss, e.g. due to human error.

We note that while correlation studies to identify prevalent attack vectors have been done before, most notably see Verizon's annual Data Breach Investigations Report (DBIR) [145] using business sector information, our goal is to use additional business information to enable a more fine-grained study, whereby the incident type distribution is quantified not just for an entire business sector, but for specific individual businesses based on other features such as employee size, region of operation, etc. This allows us to generate sharper (more highly concentrated) incident type distributions; that is, with more fine-grained definition of subsets within a sector, we are able to see incidents concentrated over a smaller number of types. An immediate consequence of this is that security investment and resource allocation decisions informed by such analysis are much more targeted and effective. We show that on average an organization can protect against 90% of all incidents by focusing on 70% of incident types; in some cases the latter can be significantly lower.

Our results are derived and presented in two parts. First, an unconditional prediction of an organization falling victim to a data incident, and second, prediction of the conditional distribution of risks over different incident types given that an incident occurs; the latter complements our estimation of the probability of an incident happening in the former. In practice, the absolute risk of experiencing an incident provides the organization with insight on the total amount of resources that should be allocated to self-protection, while the conditional risk can be used to decide the allotment of these resources to different forms of preventive measures. By combining these two results, one can also determine the absolute risk of a given incident type. In addition, the current study can guide better breach detection efforts. From this perspective, our study is aligned with the growing "assume breach" mentality in the security community [58]: everyone is a target hence all organizations should take measures to prevent, detect, and respond to incidents, in the most effective way. Last but not least, these findings can be used as guidelines in the emerging cyber-insurance market. A study of the distribution of risk among different forms of data incidents can help insurance providers better assess the potential amount of loss which in turn helps determine the contract terms, including premiums and coverage levels.

Non-goals Note that our main goal in this study is to reveal attributes of a business that are correlated with experiencing a data breach incident, rather than to detect the manifestation of a breach. Examples of such attributes include, for instance, hacking attacks target entities in the information sector more often than other industries, and an organization with a large number of employees is inherently more prone to data breach through human er-

ror. Detecting or forecasting a hacking incident by finding security flaws would require probing an organization's internal network and devices, another non-goal of this study. Furthermore, predicting incidents such as internal error, or employee misuse through the vectors that cause them are even more challenging, due to the presence of human elements. Therefore, when using the terms *risk prediction*, or *risk forecasting*, we are referring to *risk assessment* by comparing an arbitrary organization's attributes to those of victims and non-victims in our training samples, and not uncovering vulnerabilities that directly cause a breach. However, this does not imply that it is not possible to forecast cyber-security incidents without observing security flaws in how an organization is operating. A correlation study on the impact of business features such as sector and size on data breach incidents, can project how likely it is for an organization to be successfully targeted by an attacker, or suffer a data breach through human error, by determining how often similar entities have experienced data breaches in the past. Furthermore, even if a security flaw is detected in a system, the chance of it turning into a data breach by an attacker targeting said vulnerability, is partly determined by how the data breach can be monetized by the attacker, which is in turn influenced by the business features utilized in this study.

The rest of the chapter is organized as follows. In Section 2.3 we summarize existing work relevant to this study. In Section 2.2 we describe the data sets used in this chapter. In Section 2.4 we explain in detail how we build our risk assessment model, and we discuss and analyze the results in Sections 2.5 and 2.6. Section 2.7 concludes our study.

2.2 Data sets

In this section, we describe the data sets used in our study, namely the VERIS Community Database (VCDB) [143], the Open Directory Project (ODP) [29], the Alexa Web Information Service (AWIS) [8], and the IP Intelligence service from Neustar, Inc. [104].

2.2.1 VERIS Community Database (VCDB)

VCDB is a public database of reported data incidents, currently including roughly 7000 entries, more than 95% of which correspond to incidents after 2010. For this chapter, we focus only on 2013 and 2014 incidents, consisting of (at the time of this study) 1850 and 794 entries, respectively. The reports cover a wide variety of events, some examples of which are given in Table 2.1.

Each entry in the VCDB is reported using the Vocabulary for Event Recording and Incident Sharing (VERIS) [144]. The VERIS framework, as well as the VCDB, are initiatives

Incident summary
Hackers breach website of Hong Kong police force and publish non-public data, deface webpage.
A Lima, Ohio clinical psychologist is in the process of notifying clients that their office was robbed.
Pharmacy accidentally dumped hundreds of private medical records at a recycling depot.
Janitor is blackmailed into gathering documents from a court.
Parents of children at Hopkins Road Elementary Schools say their kids came home with sensitive data of other students.
Multiple Brazilian government sites defaced by Anonymous in protest to upcoming FIFA World Cup.
Hacking group DERP launches DDoS against Xbox Live networks.
Someone hacked into an electronic traffic sign on Van Ness Avenue in San Francisco.
Anonymous takes down 1,000 Israeli government and business websites for #OpSaveGaza.

Table 2.1: Incident examples from the VERIS Community Database.

by the Verizon RISK Team facilitating a unified approach to documenting and collecting security incidents. The VERIS fields for an incident are populated to answer "who did what to what (or whom) with what result?" [145]; details include the type of incident and the means by which it took place, the actor and motive, the victim organization, the assets which were compromised, timeline of the incident, and links to news reports or blogs documenting the incident. However, each entry might be only partially populated, since victim organizations tend to not disclose all the details regarding the incident.

We now explain the fields extracted from VCDB which are of interest in training and testing our classifiers. The first set is information regarding the type of a data breach, based on which an incident can classified as one of seven categories: `environmental`, `error`, `hacking`, `malware`, `misuse`, `physical`, and `social`. Each type may include additional fields that can help further differentiate incidents of the type. For instance, a physical incident might be further categorized as theft or loss, while a hacking incident might be identified as a SQL injection or a brute force attack. The second set identifies the actor responsible for the incident, falling in one of three types: `external`, `internal`, and `partner`. The data set may further include fields identifying the motive for each of these actor categories. The third set identifies the assets that were compromised during the incident. There are six possible asset types: `kiosk/terminal`, `media`, `network`, `people`, `server`, and `user-device`.

We also extract three features about the victim organization from the existing VCDB fields as input for our classifiers: industry code, number of employees, and the region of operation of the victim organization. The industry code provided is the North American Industry Classification System (NAICS) code [98] of the victim, which specifies the organization's primary economic activity. Although NAICS codes can extend to up to six digits, each further detailing the sector, we only extract the first two digits of the code for our incidents; this classifies the company as one of 25 different sectors. The employee count

captures information about the size of the organization; this entry may be a numeric range (1-10, 11-100, 101-1000, 1001-10 000, 10 001-25 000, 25 001-50 000, 50 001-100 000, and over 100 000), or simply `small` or `large` (for approximately below or over 1000 employees, respectively) when an exact number is not available. Finally, we use the region of the organization as a feature by extracting the continent of operation for the victim. Note that any said features can be missing for a specific VCDB entry; in such cases, we generally add an additional `unknown` category.

2.2.2 Alexa Web Information Service (AWIS)

AWIS is a service offered by Amazon Web Services (AWS) [15], that provides information and statistics about arbitrary websites (often second-level domains); these include traffic volume and (regional) rank, number of visitors, speed, number of pages linking to the website, and information about the organization that maintains the website, such as address, contact information, and stock ticker symbol.

We gather the following data from AWIS for all organizations in our data set. We include the global and regional traffic rank, and the number of pages linking in to the target website, as indicators of the popularity or familiarity of an organization. The regional rank of a website is extracted by finding the country which has the most contribution of page views to the website's traffic, and adding the rank in that country, as well as the country code, to our feature set. We also include the 30-day average and standard deviation of the website's global rank for a one month period before the incident, to identify recent trends in popularity. Other selected features include speed of the website (as a percentile compared to other websites), the age and locale of the website, the categories associated with it, and whether the underlying company is publicly traded in the stock market. We convert the number of pages linking in, and global, regional, and average historical rank to logarithmic scale, due to their large range of quantities. We further break each category, if possible, by separating the portion describing the region of the website. For instance, for `Regional/Caribbean/Barbados/Government`, the general category is `Government`, while `Caribbean/Barbados` is the regional category. For missing fields, we choose a reasonable default value, e.g. `unknown` for text fields, and ∞ for traffic ranks.[1] The aforementioned attributes of an organization can provide further insight into its sector, region, familiarity, and size. By combining these with features obtained from our other data sets, we are able to build a detailed description of a business, which can in turn help assess an accurate representation of its risk.

[1] Here, we use a fixed number larger than the maximum of all observed values as a proxy for ∞.

Other than age and historical traffic rank, AWIS only provides the most recent state of a webpage. Therefore, there is a relatively large time gap between our incidents (which happened in 2013 and 2014), and features obtained from AWIS (September 2015). Features such as main contributing country, locale, and category are related to the organization's region and sector of operation and are not expected to change over time. However global and regional rank, number of pages linking in, and whether the company is publicly traded can exhibit more dynamic behavior. For samples where both a global and historical rank was available,[2] the average mean absolute percentage error between the two was 6.5%. We therefore concluded that the order of a website's rank remains fairly static. Unfortunately, we could not procure similar measurements for other statistics of a webpage, since it involves caching results from AWIS and studying the changes over a long period. Nevertheless, since regional rank and number of links also capture the popularity of a page, we expect them to show similar behavior.

2.2.3 The Open Directory Project (ODP)

ODP (also known as DMOZ) was the largest publicly available directory of the web, until it was discontinued on March 17, 2017. Each entry includes a website URL, the title of the site and a short description, as well as the category of the website. By selecting random entries from this data set, we can effectively choose random non-victim organizations. For this study we use a snapshot of ODP obtained on September 19, 2015, consisting of 3 771 141 entries, of which a random selection of 16 780 entries that had not appeared in our victim data set, is used in this study as non-victim organizations. Note that our random selection may also capture victim organizations that were not reported in the VCDB. The portion of *tainted* samples in our non-victim set is upper bounded by the overall rate of data incidents.

To elaborate more on the process of selecting non-victim entities, we would first like to point out that an alternative way to select non-victim organizations would be to choose random entries from a global business directory. However, since we do no have access to such a directory, websites are used as a proxy to identify organizations. In our earlier work [86] we have used a random selection of networks to identify organizations which matched our use of network security posture measurement data. However, this selection method would limit us to companies that own network assets of their own, and those who rely on hosting providers and content delivery networks would be excluded. By contrast, almost all organizations own a website and would be included in our current approach.

[2] Alexa provides global and historical rank, for the top 30 million and 1 million websites, respectively. This is also the primary reason we have included both types in our feature set.

Furthermore, incidents covered in the VCDB include those concerning large companies, as well as data breach reports on smaller entities such as personal webpages. Using a web directory allows us to include smaller entities in our selection, resulting in a more representative non-victim group; this cannot be easily achieved by using a business or network directory.

2.2.4 Neustar IP Intelligence

IP Intelligence is a service offered by Neustar Inc. that includes geographic information, network characteristics, and ownership information over the IPv4 address space. More specifically, we use the ownership information in our study, consisting of the organization name that manages a given IP address, along with, where available, its corresponding NAICS code. This information allows us to identify the network responsible for maintaining a given website. Moreover, since VCDB only provides business sector information for victim organizations, the NAICS code included in the IP Intelligence data set allows us to include this information in our overall risk prediction for both victim and non-victim organizations. The snapshot used in this study was obtained on May 22, 2015. We include the name of the company listed as the owner of an IP address, its size (number of IP addresses owned by the same organization), and the NAICS code associated with it in our feature set. Doing so helps identify hosting providers with bad reputation, i.e. those with a higher than average presence in incident samples.

2.2.5 Pre-processing

To be able to combine these data sets for our study, we first have to match each incident report with the website of the victim organization. To obtain this information, we find the name of the victim organization through the `victim-id` field in VCDB, and extract the first Google search result for the organization name. We then manually verify the results to ensure that the websites match the victim organizations. For ambiguous victim IDs (e.g. "Indian government website"), we further read the incident report provided by a news report or blog entry (samples of which are often referred to in the VCDB description) to find the website of the entity that suffered the data breach. For the 2644 incidents that occurred in 2013 and 2014, we extracted the website for 2062 of them, and dropped the rest from our data set. Note that of the 582 incidents that we removed, 139 did not report the name of the victim organization, and the rest were not included in our study either because the victim name was too ambiguous (e.g. "Egyptian government", and "law firm in British Columbia"), or we could not find a website for the victim (e.g. "Ha Dinh primary

school", and "Purple Cow gas station"). The mapping between a victim organization and its respective website will allow us to combine entries in the VCDB with data collected from AWIS. Note that for a given year, we omit duplicate incidents for the same organization, i.e. corresponding to the same unique identifier (to be defined shortly). As an example, there are over 200 entries in the VCDB corresponding to error incidents in the United States Department of Veterans Affairs. We count all of these incident only two times, once in 2013 and once in 2014. If there are multiple entries corresponding to different forms of data incidents (e.g. hacking and misuse), we include them as separate entries when assessing risk for specific incident types in Section 2.4.4.

Note that statistics obtained from AWIS are often provided only for the second-level domain of a website. For instance domains such as `mail.google.com` and `maps.google.com` are redirected to the second-level domain `google.com`. Sub-domains are only regarded as separate entities when "they are identified as personal home pages or blogs" [7]. On the other hand, website details from ODP are generally more detailed, and can include any number of sub-domains and sub-pages. Therefore to avoid inconsistencies, we replace URLs associated with victim organizations and our random selection of URLs from ODP with their respective domains from AWIS. Hence, we will use the AWIS domain (generally the second-level domain) as a *unique identifier* for merging duplicate entries. This step reduces incident/victim and non-incident/non-victim samples to 1606 and 16 254 unique domains, respectively. We will further separate these samples, using an even split, into two independent data sets for training and testing our models, respectively.

The next step is to include features from Neustar Inc. We resolve the domain to obtain an IP address, and then look up the owner of that address. We augment our set of features with the name of the owner, its size (number of IP addresses listed under the same name), and the NAICS code associated with it. Out of the 17 860 domains from the previous step, we were able to map 17 772 of them to 5805 unique owners. Note that for 88 of the domains we were either not able to look up their IP address, or there was not any entry in the IP Intelligence data set for that address. For these samples we list `unknown` under owner and NAICS code, and a size of zero.

Finally we convert text fields to a set of binary features by tokenizing each distinct value. For categories and NAICS codes from IP Intelligence, we break each entry into multiple values with different levels of detail, and tokenize each separately. For example `Business/E-Commerce/Consulting` is a sub-category of `Business`, as well as the more detailed category `Business/E-Commerce`; and a NAICS code of 51 720 (Wired Telecommunications Carriers) is a sub-sector of 51 (Information), and 517 (Telecommunications). To limit the total number of features, we ignore tokens that have been repeated less than

10 times in our samples; this technique is often used when training predictive models over text documents, to avoid the inclusion of highly sparse features that would not present any value in a machine learning setting. The resulting reduction in features can also prevent the learning algorithm from over-fitting.

2.3 Related work

The main contribution of this chapter, compared to existing literature, is an in-depth and quantitative analysis of the risk distribution over security incident types for a given organization, which can help the latter more strategically allocate resources for prediction, prevention and detection of data incidents. The have been an increasing number of studies, leveraging data analysis and empirical evidence, in contrast to model-based approaches, for examining data breaches and their impact.

Data-driven analysis A relevant study to this study is Verizon's annual Data Breach Investigations Report (DBIR) [145]. The 2014 report, which corresponds to the timeline of this chapter, contains detailed analysis on more that 63 000 security incidents from multiple sources including VCDB. The report contains a detailed analysis on statistics of the data including action types and vectors, actor types and motives, as well as victim demographics and industry. Moreover, starting from DBIR 2014, the authors identify nine patterns describing 92% of the incidents in their report. By categorizing incidents into separate patterns, it is possible to analyze the distribution of incident varieties within each pattern and provide entities with a more specific recommendations on how to invest in their security. The report also provides the spread of attack patterns within each industry, to further narrow down the risk. For instance, it is pointed out that the main threat to organizations providing accommodation services is through point-of-sale intrusions (POS), which describes 75% of the incident reports within this industry.

As mentioned earlier, compared to the DBIR, we aim to provide a more fine-grained framework to give more specific guidance to organizations not only based on their industry, but utilizing a host of other features available to us. This includes demographic information, details about the size of the business and its popularity, and business sector information. Moreover, we couple our conditional risk distribution with the overall probability of breach in order to arrive at a more realistic sense of risk. For instance, even though a typical business in the accommodation sector is more prone to POS intrusions, their risk within that category might still be less than businesses in other sectors, given that their unconditional probability of breach is low.

In other data-driven studies, Thonnard et al. [141] perform an analysis on spear phishing targeted attacks, identifying risk factors at the organization level (industry sector and number of employees), and individual level (job level and type, location, and number of LinkedIn connections), that are positively or negatively correlated with the risk of experiencing targeted attacks. Aziz [16] presents an approach for analyzing access control policies, and leverages VCDB for evaluating the proposed framework. Farhang et al. [42] utilize VCDB to analyze the timing of security incidents and responses, and propose a two-player game for time-based security. Other works related to this chapter include studies on the trends and costs associated with data breaches. In [37], Edwards et al. use generalized linear models to uncover trends in data breaches, and conclude that the frequency and size of data incidents have not increased over the past decade. Furthermore, The 2015 Cost of Data Breach Study by Ponemon Institute and IBM [64], finds the average cost of a data breach to be $3.8 million, with $154 incurred for each lost or stolen record. [2,22,24,55,68] conduct event-study analyses on the impact of data breach disclosures on market value, and conclude that there exists a negative and statistically significant correlation between the two. Moreover, in [120] Romanosky et. al. provide an empirical analysis of data breach litigation, and in [121] discuss the impact of breach disclosure laws on identity theft.

Prediction of cyber incidents The notion of predicting cyber incidents (rather than detection) has also enjoyed popularity recently. In [134], Soska el al. apply machine learning tools to predict the chance of a website turning malicious in the future, and show that their method can achieve 67% true positive and 17% false positive. In our previous study [86], we examine to what degree cyber-security incidents may be predicted by using a range of security posture data. Compared to the above studies, our goal in the present study is to consider a broader range of data incidents, including targeted and untargeted physical and cyber-attacks from both internal and external sources, and incidents due to error, while at the same time recognizing the difference between specific incident types by emphasizing the relative risk each incident type poses to a particular organization. Note that of the 2644 reports in the VCDB for 2013 and 2014, 981 are hacking and malware incidents (cyber incident), and the rest are non-network related incidents (non-cyber incident).

2.4 Methodology

In this section, we will discuss the rationale behind the features selected for our model, followed by a detailed description of how to build a risk assessment model using the features and incident reports described in Section 2.2.

2.4.1 Feature set

In Section 2.2 we listed the features extracted from VCDB, AWIS, and Neustar IP Intelligence to be used in training our classifiers. We will now discuss our motivations for selecting these features, and why we expect them to be indicative of a company's risk of data breach. Note that while we provide simple examples for why a certain feature can be correlated with cyber-risk, our model can recognize more complex, i.e. higher order, relationships within out feature set that can help the classifier make more accurate assessments.

The first and foremost features are those that specify a company's sector of operation, namely the industry code extracted from VCDB, and the website category from AWIS. We expect an organization's industry to be strongly correlated with its risk of falling victim to different types of data breaches. A company's industry can provide insight into the types of records that can potentially be compromised (e.g. credit card information for retailers, or physical and digital records for health care), or motivations for targeted attacks (e.g. hacktivism for public administration entities). In addition, a business's sector can determine the value of data records to an attacker, which in turn influences the attacker's decision to launch an attack on said entity; this type of correlation also applies to other features used in our model, such as a business's size and region of operation. As we discussed in Section 2.3, DBIR also uses industry information to give security recommendations to businesses within a sector.

The next set of features are those that specify the size of a company: employee count from VCDB, and whether a company is publicly traded, which is provided by AWIS. We expect the size of a company to be correlated with how often it is targeted by cyber-attacks, since compromising a large company tends to be more profitable for an attacker. Furthermore, as we will see in Section 2.6.2, a larger employee count can increase the chances of breach through human error and employee misuse. Features like traffic rank (global, regional, or historical) and the number of links to a website are indicative of a website's popularity, and therefore correlated with the chances of a company being targeted.

We also expect a company's region of operation to be connected to its cyber-risk. The region is obtained from VCDB, as well as the website's top contributing country, locale, and regional category. Other features such as the age and speed of an organization's webpage, can provide more insight into its security posture. Older companies tend to be more experienced in protecting themselves against data breaches, and may have better policies in place to prevent them; a website's speed is an indication of how well it is being operated, which in turn can be associated with security posture.

Finally, our measurements from Neustar IP Intelligence will provide more details about an organizational network, which can be closely coupled with risk of network related

breach incidents. Note that an organization's website can be either hosted on the company's internal network, or by a hosting provider. The industry code from IP Intelligence can provide the classifier with the necessary information to distinguish between the two cases, since the NAICS code 518 (Data Processing, Hosting, and Related Services) can be associated with hosting providers. When an organization's website is hosted by a third party, the name of the hosting company and its size (in terms of number of IP addresses owned by the provider), can determine its reputation and how protected customers are. For self-hosted websites, the size of the organizational network will indicate the attack surface, and therefore the risk of breach through network incidents.

Table 2.2 includes the list of all features used in the training/testing of estimators in this study. We will further discuss this table in the remainder of this section.

2.4.2 Construction of the classifiers

Our ultimate goal is to provide risk assessment for an arbitrary organization given its features, i.e. a distribution of risk over all incident types. This risk can be represented in two parts as follows:

$$\Pr(\text{Incident type} \mid \mathbf{x}) = \Pr(\text{Incident} \mid \mathbf{x}) \times \Pr(\text{Incident type} \mid \text{Incident}, \mathbf{x}), \tag{2.1}$$

where $\mathbf{x}$ represents the organization in question, identified through its set of features. Moreover, the *incident type* can be any of the available data incident types, e.g. physical theft. The first term in the RHS of Equation 2.1 will be referred to as the *overall risk*, and the second term quantifies the *conditional risk*. These two probabilities are estimated separately by constructing different classifiers.

Toward this end, we use random forest classifiers [128], an ensemble learning method that constructs multiple decision trees over the training data, and outputs the average of all individual trees' outputs, hence producing more accurate prediction by averaging over many strong estimators. Random forest classifiers improve upon single decision trees by reducing over-fitting over the training set; this is achieved by training each estimator over bootstrapped samples (i.e. samples drawn with replacement), where each split in a tree is chosen among a random subset of features.[3] For overall risk estimation, we use our set of victim organizations coupled with a randomly selected set of non-victim organizations to build a binary classifier; in this case all victim organizations no matter the type of incident are given a label of one.

[3]For this study, the number of inspected features is set to be equal to the square root of the number of available features.

	Industry	Employee Count	Region	Rank	Local Rank	Rank History	Links In	Website Age	Speed	Locale	Traded	Category	Network Size	Network Name	Network Industry
Overall															
Overall	x	x	x	7.6	11.9	12.6	6.3	3	5.4	4.5	0.2	36.2	3.3	3.6	5.3
Action															
Error	21.4	25.2	18.8	x	x	x	x	9	x	x	5.7	x	19.9	x	x
Hacking	27.8	9	29.2	8.7	x	x	10.5	8.1	x	x	x	x	6.8	x	x
Comp. Cred.	0	25	8.3	x	x	x	x	16.7	25	x	x	8.3	16.7	x	x
Other	0	17.4	33.3	x	10.7	11.7	x	4.6	10.6	x	x	x	11.6	x	x
Malware	20.5	8.2	4.2	x	13	33	x	x	7.7	1.8	x	x	11.5	x	x
Misuse	17.4	9.7	6.9	24.2	x	x	19.5	9.3	x	11.4	1.6	x	x	x	x
Physical	11.3	3	7.6	x	x	33.1	6.1	x	5.6	x	0.4	33	x	x	x
Theft	26.4	0.5	2	x	x	38.7	x	6.4	6.9	x	1.9	x	17.2	x	x
Other	24.9	9.6	4.1	x	x	x	16.1	24.9	x	x	x	x	20.4	x	x
Social	14.2	21.4	18.9	x	18.8	x	x	x	26.8	x	x	x	x	x	x
Actor															
External	28.9	7.1	11.7	15.4	x	x	x	6.1	x	17.4	1.8	x	11.6	x	x
Financial	12.7	13.3	27.9	2.1	30.9	9.8	3.2	x	x	x	x	x	x	x	x
Ideology	18.7	38.5	25.8	6.8	x	x	4.1	x	6	x	x	x	x	x	x
Other	13.6	4.1	40.6	x	33.5	x	x	x	8.2	x	x	x	x	x	x
Internal	28.3	16.6	40.8	x	x	x	x	12.2	x	x	2	x	x	x	x
Financial	17.4	0	0	x	x	x	x	12.5	18	x	x	37.8	14.3	x	x
Other	8.3	0	0	x	x	37.4	18.3	x	x	x	x	20.4	15.6	x	x
Partner	19.3	11.8	12.2	x	x	x	x	16.5	x	5.3	x	22.3	12.6	x	x
Asset															
Kiosk/Terminal	13.3	11.7	5.1	x	x	9.9	1.9	x	2.9	x	0.9	54.4	x	x	x
Media	10.4	8.3	10.6	x	7.8	x	3.9	x	3.1	x	0.6	55.2	x	x	x
People	19.7	15.4	24.7	x	x	x	x	x	28.9	10.5	0.7	x	x	x	x
Server	15.7	3.1	17.6	x	13	11.9	x	3.7	x	2.2	1.6	27.2	3.9	x	x
User Device	8.3	5.5	7.2	14.3	17.3	38.9	x	6	x	2.3	0.2	x	x	x	x

Table 2.2: Features and feature importances for all classifiers.
Crosses indicate features that have not been used in training the corresponding classifier.

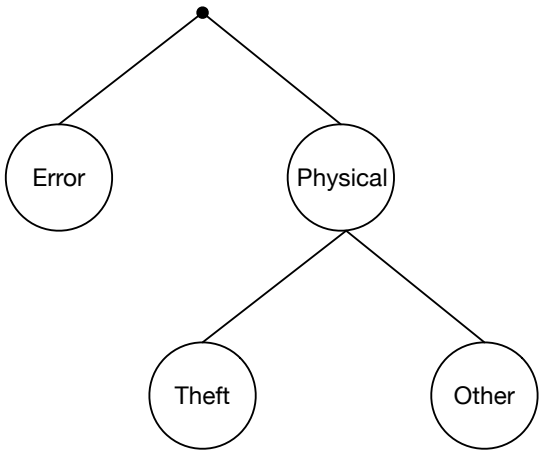


Figure 2.1: A sample risk assessment tree. The risk at each node is quantified by multiplying its conditional risk, by the risk of its parent node.

To assess the conditional risk, a naive way would be to take the incident signature (i.e. action, actor, and asset) of an entry as a class label, and the victim's features as input data for the classifier. However, given the large number of possible incident signatures, there are only a small number of samples per signature vector. Furthermore, as we have mentioned before, a significant number of incident entries provide only partial information about their corresponding incident. Ignoring such entries will leave us with even fewer samples.

Our solution to the above problem is to build multiple classifiers, each of them estimating a portion of the incident signature. This continues our previous use of the chain rule in probability. Assume that we want to estimate the risk factor for an organization with the feature vector $\mathbf{x}$ for experiencing a physical theft incident. We can break the conditional risk into multiple parts as follows:

$$\Pr(\text{Theft} \mid \text{Incident}, \mathbf{x}) = \Pr(\text{Physical} \mid \text{Incident}, \mathbf{x}) \Pr(\text{Theft} \mid \text{Physical}, \mathbf{x}). \tag{2.2}$$

As a result, entries that cite a physical incident without specifying additional details will still be included for building and testing the first classifier (first term in the RHS of Equation 2.2), but will be ignored when building the second classifier (i.e. theft). This method can be visualized as a tree as shown in Figure 2.1, where each node represents a data breach type. The risk score at a node is the result of multiplying the risk at its parent node, by the output of the classifier corresponding to said (child) node.

Note that the output of Figure 2.1 is a conditional probability, conditioned on the event that an incident has occurred. To derive the absolute risk for a given breach type, we need to multiply the result by the overall breach probability (first term in the RHS of Equation

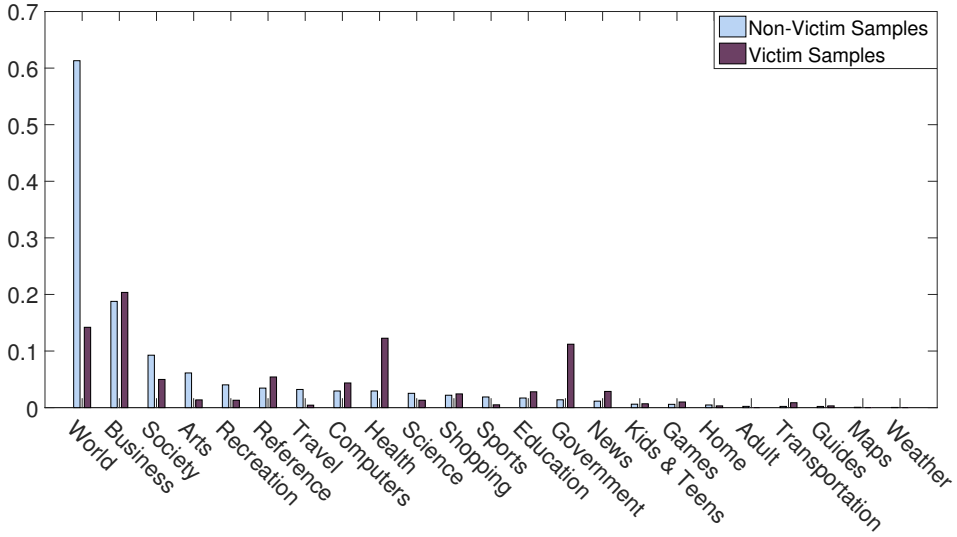


Figure 2.2: Distribution of all victim and non-victim samples (both training and testing) over Alexa's top categories. Note that a website can belong to multiple or no categories.

2.1). In the remainder of this chapter, we discuss our results on overall risk estimation and conditional risk for specific breach types separately. The rationale behind this separation is that the former serves as a forecast on security incidents. On the other hand, the point of the latter is not to make a single prediction on the type of incident that is going to happen, but to estimate the distribution of risk among multiple incident types; as we shall see, predictions for single categories are significantly less accurate than overall risk estimations due to its density estimation nature. We further elaborate on this point in Section 2.5.2.

2.4.3 Forecasting overall risk

To forecast the overall risk of breach, we assign labels zero and one to our non-victim and victim samples, and train a random forest consisting of 50 trees over victim samples from 2013, and a random selection of 11 585 (out of 16 254) entries from the non-victim data set. We use features from AWIS and IP Intelligence for prediction, and omit features from VCDB since they have only been provided for victim organizations. We use incident samples from 2014 and the remainder of our non-victim data set for testing.

The first row in Table 2.2 summarizes the importance of each feature in the final classifier. As is evident from this table, The most used feature for this type of risk assessment is the general category of the website. For further elaboration on this point, we have shown the distribution of victim and non-victim samples over the top-level categories from AWIS in Figure 2.2. While our incident samples have presence in most of the top level cate-

gories (excluding `adult`, `maps`, and `weather`), it is possible to identify categories that exhibit higher- or lower-than-average risk. For instance, the portion of incidents that belong to `health` and `government` are significantly larger than the global population, while the `world` and `arts` categories can be associated with low risk. Note that the `world` category describes webpages that are in languages other than English. The discrepancy in this case can also be due to under-reporting, since VCDB tends to focus more on incidents that happened in the US.

Note that inherent biases in our victim data set may affect the output of our trained model. The most prominent examples are biases toward incidents in the US, and also certain industries due to disclosure laws. For instance, businesses operating in retail are more likely to disclose data breaches due to concerns that customer information may have been compromised, while other industries might be under-represented in publicly disclosed data breaches. Consequently, we may underestimate or overestimate a business's cyber-risk based on its region or sector. In other words, our model is estimating the risk of a publicly disclosed breach, which is not necessarily the same as risk for undisclosed data incidents. This issue may be alleviated by training models over specific groups of victims and non-victims, e.g. training a classifier on the subset of samples that belong to a certain country, or industry, ensuring that samples are compared to organizations of the same type, and therefore with the same incident reporting rate. In this chapter, we do not train separate models for overall risk prediction since our goal is to provide a single assessment that can be used to compare organizational risk, regardless of region or sector. However, we will further explore this technique in Section 2.6 for assessing risk of different incident types.

2.4.4 Forecasting conditional risk

Given the training and test samples (incidents belonging to 2013 and 2014, respectively), we first train a binary classifier for each incident type, using a random forest model consisting of 20 trees. To prevent over-fitting, we set the minimum number of samples at each leaf of the decision trees to 25. However, we may still experience some over-fitting due to the large number of features available to our classifier. To help alleviate this problem, we limit the number of features used for each random forest as follows: we always use the three features extracted from the VCDB, namely industry, employee count, and region. Out of the remaining 10 features, we select the most significant through cross validation, i.e. training multiple classifiers using different combinations of features, and selecting the one with the best performance. The list of features used for each classifier, as well as their importance in the resulting random forest classifiers, are included in Table 2.2.

2.4.4.1 Incident categorizations

Using the classification method described above, we apply our risk assessment scheme separately to three parts of the incident signatures: *action*, *actor*, and *asset*. Each of these classifiers focuses on a separate aspect of an incident. If a single entry matches multiple incident categories, e.g. a hacking incident through misuse of privileges, we break it into multiple incidents that each belong to a single category.

Action type The action type falls into one of the seven general categories discussed in Section 2.2.5: `environmental`, `error`, `hacking`, `malware`, `misuse`, `physical`, and `social`. We omit environmental incidents, of which there are only four samples between 2013 and 2014. We further categorize hacking events into two sub-categories: (1) hacking incidents that involve data breach through compromised credentials, including stolen credential, brute force, and backdoor attacks, and (2) all other forms of hacking, 75% of which are SQL injection and Denial of Service (DoS) attacks. We also divide physical incidents into two sub-categories of (1) theft and (2) everything else, 88% of which are due to tampering.

Knowing the action type can provide significant information on the types of preventive measures that can be used to reduce loss. For instance, the first group of hacking incidents can be prevented by setting strong passwords and changing them on a regular basis, as well as not storing unencrypted credentials at insecure locations. Incidents due to human error and misuse of privileges can be reduced by employee education, setting and enforcing internal regulations, and avoiding unnecessary access privileges for employees and/or business partners.

Actor type and motive In addition to action types, we train our classifier based on the actor responsible for the incident, i.e. `external`, `internal`, and `partner`. Internal actors are divided, based on the underlying motive, into two sub-categories of (1) financial motives, and (2) other motives, including convenience, espionage, grudge, ideology, and fun. External actors are similarly sub-categorized into (1) financial, (2) espionage, (3) ideology, and (4) fear, fun and grudge. Incidents due to business partners are not further sub-categorized due to insufficient samples.

Assessing the risk associated with various actor types can prompt organizations to determine policies for employee education and access to data (for internal incidents), guard their network periphery from external attackers, and perform due diligence when selecting business partners.

Incident type	Crimeware	Cyber Esp.	DDos	Stolen Cred.	Error	Skimmers	PoS	Misuse	Web app	Else
# of samples	67	16	106	326	333	66	19	272	399	356

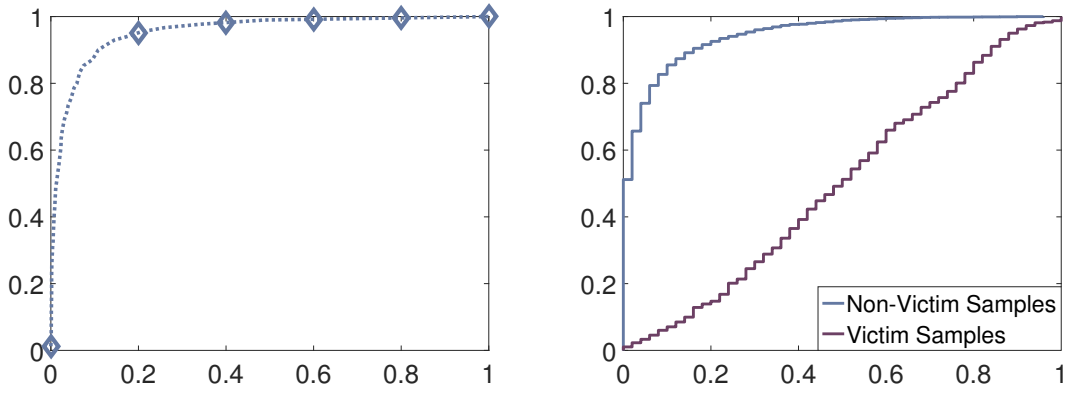
Table 2.3: VCDB data categorized using DBIR 2014 patterns. Only 82% of the data can be described by the nine patterns.

Asset type Finally, we look at the types of assets that were compromised during the incident. As detailed in Section 2.2.5, asset types include `kiosk/terminal`, `media`, `network`, `people`, `server`, and `user-device`. We have omitted network related assets due to insufficient number of samples. Knowing what asset types are more likely to be affected can significantly improve our ability to estimate the amount of potential loss following security incidents. This can guide insurance underwriters in designing more appropriate policies catered to specific client organizations. These predictions can also be used to adopt more strict access policies for assets that are at most risk, and advice network administrators to keep regular backups when assets such as media and servers are involved.

Comparison with DBIRs' categorizations Our choice of categorizations is consistent with the one adopted by Verizon in the 2008-13 DBIRs, but differs from the categorizations proposed in their latest reports. DBIR 2014 uses hierarchical clustering to identify nine incident classification patterns (combinations of actions, assets, and actors) that can be used to describe 92% of all incidents. Examples of these patterns include cyber-espionage, point of sale intrusions, and insider misuse. Despite the effectiveness of this clustering method in accurately describing incidents in the data set used by Verizon, an application to the subset available through VCDB would fail to provide a similar precision, see Table 2.3: due to lack of sufficient details, 18% of the VCDB data will not fit the nine proposed patterns (as opposed to only 6% in Verizon's larger data set). This is one of our main motivations for selecting three different categorizations based on VERIS primitives only, i.e., actions, actors, and assets.

2.5 Results

In this section we will evaluate the proposed empirical techniques for quantifying the overall risk of data breach, as well as the conditional risk distribution for various categorizations of incidents, as described in the previous section. All evaluation is conducted on a held-out test data set containing half of all available samples, which is not used for training the evaluated models.



(a) ROC (b) CDF

Figure 2.3: Receiver operating characteristic (ROC) curve for overall risk estimation (left), and cumulative distribution of risk (right)

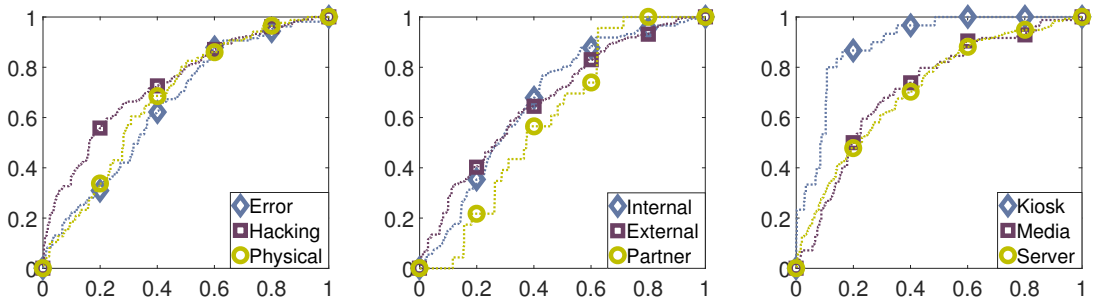
Category	Victims	Non-Victims	AUC	TPR	FPR
World	56	2204	0.928	91.1%	18.2%
Business	73	817	0.968	89.0%	4.0%
Society	20	325	0.922	90.0%	20.0%
Reference	21	134	0.841	85.7%	43.3%
Computers	25	119	0.939	88.0%	16.8%
Health	36	117	0.954	88.9%	21.4%
Government	58	42	0.876	89.7%	35.7%
Overall	482	4669	0.953	89.6%	11.3%

Table 2.4: Accuracy of overall risk estimation over Alexa's top categories.

2.5.1 Overall risk

Figure 2.3a displays the receiver operating characteristic (ROC) curve of our overall risk estimators, evaluated over the test samples. By identifying organizations with similar attributes to those that have previously experienced a data breach, we can achieve a 90% true positive rate in flagging organizations in our victim set as high-risk, while keeping false positive rate at 11%. These numbers are comparable with our previous results in [86], where we were able to forecast cyber incidents with 90% true positive and 10% false positive rate. Figure 2.3b shows the distribution of the classifier output scores for victim and non-victim test samples. There is a clear distinction between the two distributions, with victim samples having more bias toward higher scores, signifying more risk.

Moreover, Table 2.4 summarizes the accuracy of our model over Alexa's top categories



(a) Action (b) Actor (c) Asset

Figure 2.4: ROC curves for action (left), actor (middle), and asset (right) classifiers.

in Figure 2.2, as well as the overall accuracy on all samples. Each row in Table 2.4 displays our model's performance over the test samples in 2014 that belong to the corresponding category. We have removed categories where we have less than 20 victim samples. We have included the number of victims, and non-victims in each category, as well as the true positive rate that is closest to 90%, along with its corresponding false positive rate. The area under the curve (AUC) metric displays the area under the ROC curve. Note that the AUC score is independent of the fraction of the test population in each class, making it a useful metric for evaluating performance on unbalanced data sets, i.e. the efficay of the trained estimator for rank-ordering samples. The best accuracy belongs to the `business` category, and `reference` and `government` perform the worst.

2.5.2 Risk distributions

Figure 2.4 shows our results on prediction of specific incident types. We have drawn ROC curves for three types each in the action, actor, and asset categorizations. Comparing to Figure 2.3a, the accuracy of these classifiers is significantly lower, typically achieving a 80% true positive at 50-60% false positive rate, except for the kiosk asset type that achieves the same accuracy at 11% false positive rate; note that this asset type is only owned by a select few industries, which most likely contributes to the high accuracy observed here.

To explain the difference between Figure 2.3a and 2.4, we will consider a model with n different incident types, and a sample entity with probability of breach of p. We will then analyze this example for different scenarios. In the first case, the absolute (unconditional) probability of breach for one incident type is equal to p, while other types have zero probability, and we will be able to predict with certainty the type of the data breach. In the second case, assume that all breach types are equally probable, and the conditional risk is a discrete uniform random variable. In this scenario, if our predictor outputs a label of one

Industry/Organization	Error	Hacking		Malware	Misuse	Physical		Social
		Comp. Cred.	Other			Theft	Other	
Manufacturing	0.08	0.09	0.33	0.13	0.22	0.13	0.00	0.02
Retail Trade	0.15	0.26	0.11	0.19	0.09	0.09	0.11	0.02
Information	0.09	0.28	0.41	0.07	0.04	0.03	0.01	0.07
Russian Radio	0.14	0.16	0.40	0.02	0.10	0.10	0.03	0.03
Verizon	0.28	0.17	0.22	0.08	0.19	0.06	0.05	0.05
Finance & Insurance	0.25	0.09	0.11	0.05	0.12	0.10	0.19	0.07
Pro., Sci. & Tech. Svcs	0.16	0.09	0.56	0.04	0.13	0.09	0.00	0.02
Educational Svcs	0.30	0.13	0.21	0.06	0.11	0.14	0.00	0.05
Health Care & Social Asst	0.25	0.08	0.03	0.02	0.23	0.38	0.02	0.01
Accommodation & Food Svcs	0.08	0.37	0.00	0.18	0.16	0.11	0.11	0.00
Public Administration	0.27	0.09	0.29	0.03	0.17	0.10	0.01	0.03
Internal Revenue Service	0.21	0.08	0.15	0.06	0.17	0.09	0.02	0.03
Macon-Bibb County	0.20	0.13	0.23	0.07	0.14	0.23	0.04	0.04
Overall	0.22	0.12	0.21	0.06	0.15	0.14	0.04	0.04

Table 2.5: Conditional risk distribution by business sector, and for sample organizations (highlighted rows).

with probability q for all types, we will on average see q true positives, and $q(n-1)$ false positives, and the average true positive and false positive rates, averaged over all classifiers, will be equal to q. If the risk is equally distributed between k incident types, then for every true positive the predictor will be penalized by $k-1$ false positives, and the overall true positive and false positive rates will be q and $q(k-1)/n-1$, respectively. Note that regardless of the type of an organization, its risk will never be zero for breach types such as *error* and *misuse*, and as long as it owns any form of network assets, it will be vulnerable to hacking incidents (e.g through zero-day vulnerabilities). As a result, the main value of this risk distribution estimate is not as a forecast for a particular incident type, but rather as a prediction of how the overall risk is distributed over all incident types by combining the outputs of all classifiers. We will discuss how to interpret this conditional distribution in Section 2.6, and show that it can lead to a sparse or diverse range of risks.

To gain insights on how details about a business can affect their risk of experiencing various types of data breach, we start by deriving the distribution of risk over incident action types for each industry sector. The results for nine business sectors, as well as the overall distribution are included in Table 2.5; these results use only sector information in

training the corresponding classifiers. Note that this is equivalent to simply measuring the distribution of incidents in each sector, since the random forest classifier is using only a single feature. There are a few observations on the risk distribution of different sectors. For instance, information companies are more prone to both types of hacking, and less likely to sustain damage due to physical incidents. In contrast, the health care industry has low risk in hacking but high risk in physical attacks, especially theft. These observations are intuitively to be expected, since information companies' most valuable assets are generally stored in non-physical formats (e.g. on the cloud), while the health care industry may still use physical forms of archiving sensitive data such as patient information.

To highlight the additional gain we get by using more features than just industry sector information, we also show in Table 2.5 a number of examples. In these cases our classifiers can generate much more specific risk predictions. For instance, we can see that compared to a typical information company, Russian Radio has less risk in malware, social, and hacking through compromised credentials, but higher risk in error, misuse, and physical. Verizon and Macon-Bibb County exhibit a more uniform risk across the board. The higher risk for Verizon in error and misuse (also the lower risk of Macon-Bibb County in the same categories) can be attributed to their respective sizes. As the number of employees grows larger, so does the risk of data incidents due to human error and malevolent employees. These much more refined and targeted predictions would not be possible without using additional features. As we shall show later in Section 2.6.2, with proper thresholding the actual incidents in these organizations were also correctly identified.

2.6 Dealing with rare events

Looking at Table 2.5, there is an imbalance in the overall frequency at which different incident types appear in our data set. Social incidents occur rarely as compared to error and hacking incidents. It is indeed possible that social incidents are rare events, and therefore should not be a priority when determining security policies. However, an important challenge in building a risk assessment model is under-reporting of security incidents by victims. Data breach reports are largely undisclosed, as organizations tend not to expose their security posture information unless necessary. Our data set, VCDB, is a collection of publicly disclosed breaches; these incidents have either been detected by external sources (e.g. website defacement), or are incidents which an organization is obligated to report due to the compromise of private customer information (e.g. payment information or health records). Thus, not only incidents are commonly under-reported, but it is also safe to assume the existence of selection bias in the data: each incident type is represented differently

as a result of both availability and variation of detection methods, and the corresponding industries' disclosure policies. This bias could cause a tendency towards flagging and protecting from incidents that are reported more often, in turn resulting in poor protection against less commonly reported incidents.

One way to address this issue is to ignore the frequency at which incident types are reported. In other words, rather than looking at each row in Table 2.5, we could base our decisions on the distribution of risk within each column. For instance, we can make the observation that finance and insurance companies exhibit higher than average risk in social incidents, even though the absolute risk in this category is the second lowest in its respective row. By having different standards, or thresholds, of what signifies high-risk in each category, we can alleviate the impact of potential under-reporting and reporting bias in the data set and prevent the tendency of ignoring rare events by ensuring equal protection among all incident types. Specifically, after training our classifiers and obtaining risk outputs on the input data, we specify thresholds for each incident type separately, such that the reduction in risk is consistent among all types; this is detailed in the next section. Note that this *normalization* of risk scores is possible mainly due to the fact that we are constructing a separate classifier for each incident type.

2.6.1 Interpreting the classifier output

After estimating an organization's risk in each category by feeding its features into our classifier, the next step is to interpret these scores by determining what range of values indicate heightened risk. Based on our discussion in the previous section, this is achieved by computing the ROC curve for each binary classifier on the training set, and choosing the point that corresponds to a predefined true positive rate. We will use the family of thresholds corresponding to these points to determine risky incident types for any arbitrary organization, hereafter referred to as the *risk profile*. Selecting a more conservative set of thresholds (i.e. higher true positive rate) will tighten the business's security by advising it to invest in a larger set of self-protection methods. This selection represents the trade-off between the amount of resources an organization allocates to self-protection, and the reduction in incidents it desires to attain. From this point on when referring to *thresholds* used for deriving the risk profile, we simply mean the family of thresholds acquired for a specific true positive rate. We find these thresholds by looking at the ROC curve of each classifier, and finding the point that corresponds to a specific accuracy (e.g. 80% true positive rate). Note that these thresholds are specific to our incident source (VCDB), through its reporting rates on different incident types. Therefore, an incident data set with

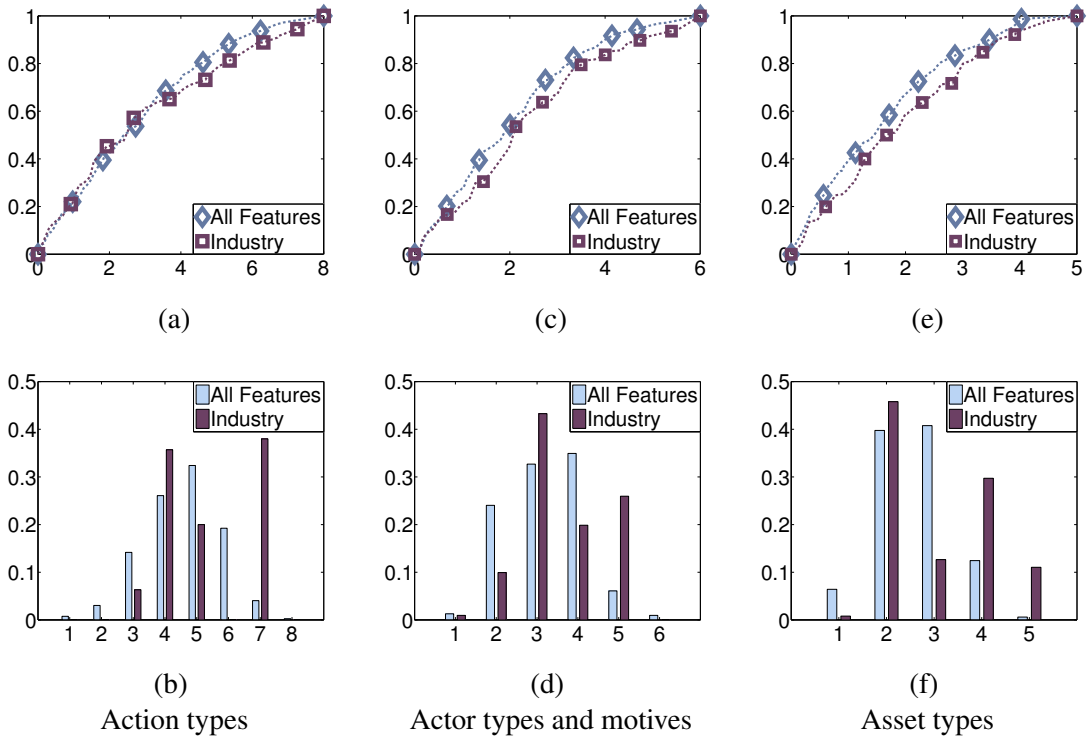


Figure 2.5: Detection rate vs. average number of high-risk incident types (top), and distribution of organizations over the number of types in their risk profiles (bottom).

different reporting rates would yield a new set of thresholds.

2.6.2 Evaluation of risk profiles

For evaluation, we first obtain the risk profiles of organizations in our test samples, for various sets of thresholds. We then calculate the accuracy of our risk assessment model, by counting the number of incidents which belong to one of the risky types forecasted by our risk profiles. An important advantage of our model is in reducing the number of predicted high-risk categories for each organization; achieving the same accuracy by advising organizations to focus on a smaller set of incident types, will help achieve the same level of protection by spending less resources on security through more informed allocation of efforts for self-protection.

Figures 2.5a, 2.5c, 2.5e summarize our results over action, actor, and asset types, respectively. Each point in the plot denotes the accuracy of risk profiles obtained from a particular set of thresholds, versus the average number of risky types forecasted by these profiles. To illustrate the improved performance of using our extended set of features, we have also included the accuracy curve of a predictor using industry information alone (see

Table 2.5). For action, actor, and asset types we can correctly forecast 90% of the incidents in our data set by flagging, on average, 5.6 (70% of incident types), 4.0 (67%), and 3.5 (70%) incident types, respectively. In other words, we can achieve this accuracy by eliminating at least 30% of all incident types. Using only business sector information, the numbers increase to 6.5 (81%), 4.8 (80%), and 3.6 (72%). Our findings suggest that fine-grained attributes of businesses can result in forecasts that are (on average) more accurate than those obtained using industry information alone; this distinction is more visible when predicting over action and actor categories.

Note that for a given point in the plot, the number of high-risk categories in the risk profile can vary across organizations. Figures 2.5b, 2.5d, and 2.5f demonstrate the distribution of organizations over their predicted number of risky types, corresponding to the 80% accuracy point in the top plots. Looking at Figure 2.5b we can see that using all features, there are organizations whose risk profiles only consist of one or two incidents types, while others may include up to seven types.

We present a number of these samples in Table 2.6, whose risk scores have already been discussed in Table 2.5. The first two examples in the table belong to the information sector, and the last two are public administration organizations. We have included the risk profiles for these sample organizations using our extended feature set, as well as the risk profile using only industry. For the information sector, the latter recommends focusing on both types of hacking, as well as social incidents, whereas for public administration it deems all but the second type of physical incidents risky. By contrast, using our extended feature set, we are able to eliminate malware and social incidents as likely threats for Russian Radio, and still provide an accurate risk profile. Similarly for the Internal Revenue Service we are able to narrow down the list of threats to two types without losing accuracy. Macon-Bibb County and Verizon are assessed to have a broad range of risks, more so than their respective industry average would suggest; this highlights that for these organizations they may be attacked on multiple fronts, which may call for a different type of resource allocation strategy. The point is that this type of fine-grained prediction is much more specific to an organization itself rather than using the industry average as a proxy. We also note that in all these cases our risk profile correctly captured the actual incident occurrences (as indicated by an "×").

It is worth noting that the grey cells in Table 2.6 not marked with an "×" are incident types deemed likely by our classifier but unrealized in reality (not observed in our data set). These should not be viewed as a discrepancy; rather, the relationship between a predicted risk profile and actual incident occurrence is analogous to that between a dice with a certain probability of turning up each side and the outcome of tossing the dice in a particular

Industry/Organization	Error	Hacking		Malware	Misuse	Physical		Social
		Comp. Cred.	Other			Theft	Other	
Information								
Russian Radio			×					
Verizon			×					
Public Administration								
Macon-Bibb County	×							
Internal Revenue Service					×			

Table 2.6: Risk profiles for sample organizations, and their corresponding industries' profiles. Gray cells signify high-risk types, and crosses indicate the actual incident.

Industry (number of samples)	Error	Hacking		Malware	Misuse	Physical		Social
		Comp. Cred.	Other			Theft	Other	
Manufacturing (39)	30.8	97.4	51.3	89.7	33.3	28.2	76.9	41.0
Retail Trade (63)	34.9	100.0	46.0	76.2	42.9	9.5	68.3	23.8
Information								
Small (49)	22.5	100.0	100.0	65.3	12.2	8.2	38.8	59.2
Large (41)	36.6	100.0	80.5	70.7	36.6	0.0	51.2	87.8
Finance & Insurance								
Small (53)	66.0	62.3	18.9	75.5	18.9	34.0	75.5	60.4
Large (91)	64.8	41.8	29.7	31.9	67.0	49.4	86.8	75.8
Pro., Sci. & Tech. Svcs (44)	54.6	72.7	27.3	50.0	27.3	45.5	36.4	43.2
Educational Svcs								
Small (27)	81.5	44.4	14.8	63.0	40.7	92.6	25.9	33.3
Large (46)	89.1	34.8	2.2	19.6	41.3	82.6	41.3	26.1
Health Care & Social Asst								
Small (97)	59.8	28.9	7.2	22.7	54.6	95.9	46.4	10.3
Large (97)	93.8	10.3	3.1	7.2	96.9	96.9	42.3	24.7
Accommodation & Food Svcs (33)	72.7	6.1	15.1	48.5	87.9	78.8	54.6	9.1
Public Administration								
Small (41)	95.4	85.4	24.4	22.0	63.4	51.2	9.8	19.5
Large (96)	97.9	32.3	10.4	2.1	93.8	67.7	0.0	55.2
Overall (1426)	61.6	61.9	37.5	32.4	56.9	51.5	38.1	38.9

Table 2.7: Average risk profiles by business sector and size.

random trial. In other words, in the example of the Internal Revenue Service, even though misuse is the only incident that actually occurred, the result suggests that an error event could just as well have happened. This is because in essence our classification constructs risk profiles by extracting details about a business and examining actual incidents that have occurred to other, *similar* companies. In this case, for organizations that share the same business model as the Internal Revenue Service, error and misuse constitute the majority of data breach reports; thus given the information available to us, both incident types are regarded as high-risk.

To close this section, we display the average risk profile over action types of all organizations, as well as average risk profiles over action types for different industry sectors and sizes in Table 2.7. Each number in the table represents the percentage of organizations, for whom the respective incident type is deemed risky. For instance, 61.9% of all organizations have high risk in hacking incidents due to compromised credentials. However, for 100% of organizations in the information sector this type of hacking poses a high threat. The risk profiles are obtained for the 80% accuracy point in Figure 2.5a.

We can identify a number of trends in Table 2.7. As discussed previously, large companies tend to have higher risk in error and misuse. Sectors that are more prone to error include large health care, and both small and large public administration. Large health care and large public administration companies also run a high risk of misuse. Incidents of error exhibit a substantial presence in all business types, the minimum being 21.2% for information companies. Note that overall, all of the incident types are flagged for at least 30% of our samples, even tough their occurrence rate is widely different as evidenced in the last row of Table 2.5. This is due to our choice of ignoring the a priori distribution of incidents, as explained in detail in Section 2.6.

Comparing Tables 2.6 and 2.7 can help provide some insights on how having additional features has helped eliminate (or introduce) possible risks for those sample organizations. For instance, small information companies tend to have lower risk in social incidents, and this has helped us eliminate this category as a possible threat for Russian Radio. We can also see that small public administration and large information companies have a more uniform risk among all types, attributed to the risk profiles for Macon-Bibb County and Verizon, respectively. The Internal Revenue Service, a large public information company, is expected to have less risk in the second type of physical incidents, as well as hacking and malware. Note that one cannot completely explain the generated risk profiles by only looking at business sector and size information alone, as they are a result of analyzing the data set's distribution over all the features in Table 2.2. For instance, large public administration organizations tend to have higher risk in social events than small ones, even

though this incident type has been flagged for Macon-Bibb County and not the IRS. In this case, other features of the IRS have contributed to its lower risk.

2.7 Conclusion

Our results demonstrate how, and to what extent, can business details about an organization help forecast its overall risk of data breach, as well as the relative risk of experiencing different types of data incidents. We observe that it is possible to forecast future security incidents with high accuracy. However, even though there is notable correlation between organization features and the incident signatures in our data set, it is impossible to assert with certainty the types of incident an organization is likely to face. We acknowledge the fact that there is an inherent randomness in incidents suffered by organizations: no business is prone to a single type of incident. As observed in our results, while risk in incidents such as hacking and theft may vary largely across sectors, any organization is likely to experience incidents due to miscellaneous errors. Nonetheless, training our classifiers on a rich and granular set of features can help construct more accurate risk profiles. The feature set used in this chapter provides only high level information about the organization itself, and not its security posture. Even though these features are the easiest to obtain, as they all are publicly available, further information indicative of an organization's security policies will undoubtedly help narrow down its risk profile. Externally observable signals, such as the ones used in [86], as well as inside information, may be used to infer a business's security posture from measurement on its policies, and how it manages its public and private assets.

Note that our model's output is as good as the labels that our incident data set provides. VCDB reports publicly disclosed data breaches, and therefore our model's output is essentially assessing the risk of experiencing a publicly reported incident. Whether these results can generalize for data breaches that were not reported, depends on how representative our incident samples are. There are a number of biases in self-reporting of incidents, and those that are externally detected by a third party. For instance, incidents that involve customer information such as credit card numbers are more frequently reported, and attacks such as website defacement can be easily detected externally. However we might not have a representative sample of incidents that result in the theft of trade secrets, and proprietary information. Furthermore, the discrepancy in reporting rates of different incident types, might lead to underestimation, or overestimation of risk in our assessments. While we alleviate this issue in our treatment of rare events in Section 2.6 for estimating risk distributions, the problem remains for overall risk assessments. Moreover, we only focus on discrepancies for specific incident types (by action, actor, and asset types), and do not take

other factors into consideration. Other variables that may impact the reporting of incidents include region (VCDB mainly focuses on US incidents), and business sector. A more comprehensive source of data breaches can improve our assessments, and allow us to use more sophisticated machine learning methods in order to find factors that influence cyber-risk.

It is worth noting that incident types are often too ambiguous to act upon for a security unaware business operator, hence the need for explicit, actionable security recommendations. Note that there indeed exist frameworks providing such recommendations. For example, the SANS institute's critical security controls [124] provides categorizations of security controls, where each category describes a specific action or policy that can be implemented by a business in order to raise its security levels. Verizon uses this framework to provide general security recommendations in its annual Data Breach Investigations Report, and the SANS institute offers a partial mapping between these controls and the VERIS incident categorizations. Translating risk profiles into actionable security recommendations can further extend the practical utility of our risk profiles. Furthermore, our current data set does not contain information on the monetary impact of each incident type. Obtaining such information, and combining it with the cost of protection for each incident type, will allow one to craft more economically-informed recommendations.

CHAPTER 3

The Effects of Individual User Behavior on End-Host Vulnerability State

3.1 Introduction

Unpatched software vulnerabilities represent a valuable resource for attackers. Exploits for these vulnerabilities can allow miscreants to control the vulnerable hosts remotely (e.g. to bootstrap a botnet or to launch distributed denial of service attacks) or to steal sensitive information (e.g. passwords, private keys, credit card numbers, medical records). Even small populations of personal computers that fail to deploy a patch present a security threat for the whole Internet, as networks of compromised hosts provide the computing infrastructure for cyber crime operations [56, 100] and the stolen sensitive information fuels the economic activities conducted on underground markets [10, 44]. Unpatched vulnerabilities also present a threat for enterprises, as an outward facing machine with an exploitable vulnerability can provide unauthorized access to the company's internal network; for example, the Heatbleed bug has been used in several data breaches following its public disclosure [38, 39, 47]. Moreover, the emergence of exploit kits [56, 76], which provide exploitation capabilities off-the-shelf, makes it easy for attackers to compromise large numbers of machines in an automated fashion.

To counter these threats, software vendors create and disseminate patches that users then install to remove vulnerabilities on their machines. Vendors have also increased the automation of their software updating mechanisms [30, 50] in an attempt to accelerate the patching process to sidestep possible tardiness on the part of the end users.

It follows that the vulnerability state of any given end host at any given time, reflected in the number of *known but unpatched vulnerabilities*, and *unpatched vulnerabilities with known exploits*, is the result of a combination of factors, including (1) the user's updating behavior, (2) the software products' patch release timeliness with respect to the disclosure of vulnerabilities, (3) the update mechanisms employed to deploy patches on user

	Findings	Implications
+	The user behavior can be captured using single-parameter distributions.	Users' willingness to patch is memory-less (i.e. independent of past decisions), and does not vary largely across different countries.
+	The geometric distribution provides a good fit, even for products that use silent updates.	This simple model significantly simplifies analysis of the relationship between end-user patching behavior and the resulting vulnerability state of their machines.
+	Silent updates lead to shorter windows of vulnerability for end-hosts (as expected).	The product vendors can improve the vulnerability state across the user population by adopting a silent updating mechanism.
-	Even with silent updates, the majority of hosts have long windows of vulnerability (somewhat to our surprise).	The large number of security flaws found in popular client-side applications can limit the benefits offered by silent updates.
-	Many machines exhibit long windows of susceptibility to known exploits.	Exploit kits present a direct threat to these hosts.

Table 3.1: Summary of findings. +/- indicate positive and negative impacts on security.

hosts, and (4) the frequency at which vulnerabilities are disclosed and exploits are developed. While the latter three elements have been extensively studied in the literature (see e.g., [5,6,9,12–14,23,26,27,45,83,103,108,116,129,153] on vulnerability disclosure and patch releases, [34,93,99,114,115,149] on patch deployment, and [12,18,19,45,56,122] on exploits) relatively less is known about the impact and the factors influencing the individual user behavior. Prior work in this area has introduced several hypotheses on why users might delay patching vulnerabilities [46,95], reported results from targeted user studies [142], and aggregated patching measurements for individual vulnerabilities over the general population and over selected groups of users [99].

In this chapter, we present a broad field study of individual user behavior. Our study includes a total of more than 400 000 end users over a period of 3 years (from 01/2010 to 12/2012), and their updating activities concerning 1822 vulnerabilities across four software products (client-side applications). The updating automation employed by these applications ranges from prompting users to download or install patched versions to automated (silent) updates, which require minimal user interaction. Our main goal is to understand (1) how users behave on an individual level (as opposed to collectively), e.g. do they fall into a few types, and what features characterize each type, and (2) how different updating behaviors relate to the vulnerability state of their respective machines, and how this relationship differs across software products. Such an understanding would allow us to assess the effectiveness of patching in protecting hosts against cyber attacks and its long-term impact on the cyber-arms race.

To achieve the above goal, we employ a combination of empirical analysis and mathe-

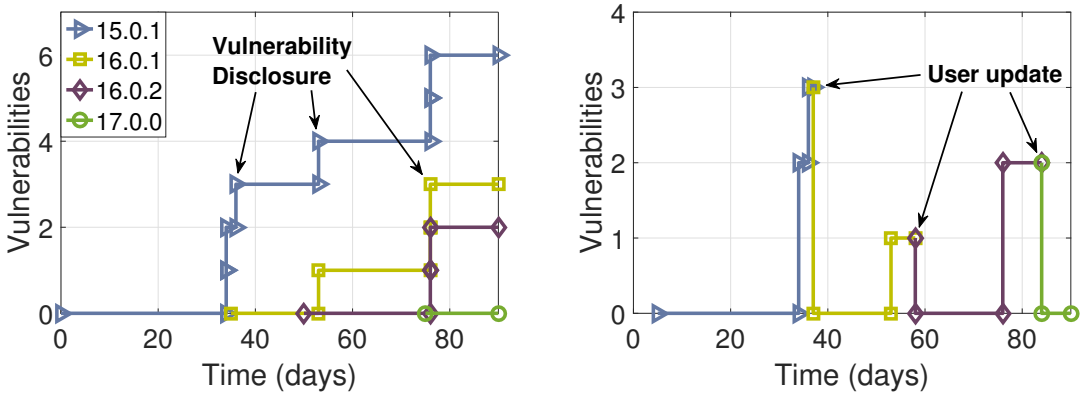
matical modeling. There are a number of significant challenges that arise in data processing to tease out individual updating behavior. Specifically, on the user side, these include obtaining precise measurements and dealing with irregular user behavior such as installation of multiple product lines. For specific products, extracting exact releases dates for successive software releases, and the number of security flaws associated with each application version, is crucial to the accuracy of our analysis. In summary, the work presented in this chapter makes the following three contributions:

- We propose methods for quantifying the user updating behavior from field measurements of patch deployment.
- We conduct a systematic study of vulnerability patching, from the perspective of individual users (rather than individual vulnerabilities), and quantify the corresponding vulnerability state of the users' machines. Table 3.1 summarizes our findings.
- Building on insights from our measurements, we create and evaluate a parametrized model for explaining individual patching behavior, and discuss its implications for end-host security.

This remainder of this chapter is organized as follows. In Section 3.2 we review the vulnerability life-cycle, and present our modeling approach. Section 3.3 describes our data sets, and pre-processing steps taken to prepare them for analysis. Section 3.4 presents our empirical results on modeling user behavior, and its implications for end-host security. We discuss our findings in Section 3.5, and conclude in Sections 3.6 and 3.7.

3.2 Vulnerabilities from the perspective of end-users

Security vulnerabilities are recorded in public databases, such as the National Vulnerability Database (NVD) [105]. In these databases, each vulnerability is assigned a unique identifier, i.e. CVE-ID; for instance, the Heartbleed vulnerability [34] is identified as `CVE-2014-0160`. Consequently, the vulnerability life-cycle is commonly modeled from the perspective of individual vulnerabilities [12, 99], and prior measurement studies similarly focused on quantifying per-vulnerability patching delays [34, 45, 99, 115, 149]. However, because exploit kits include exploits for multiple vulnerabilities and fingerprint the targeted hosts to determine which of these vulnerabilities are left unpatched [56, 76], the individual vulnerability life-cycle does not completely capture the security threats that exploit kits present to end-user machines.



(a) (b)

Figure 3.1: Evolution of number of vulnerabilities in successive Firefox versions (left), and following a user's update events (right). Each color represents a single version.

Our goal in this chapter is to study the vulnerability state of individual hosts, resulting from disclosure and patching events across multiple vulnerabilities. Specifically, we aim to estimate and model the duration when each host is *vulnerable to at least one known exploit* and to study the relationship between a user's promptness in installing software updates and their security vulnerabilities over long observational periods. Our model for explaining a host's vulnerability state has two components: the product release dynamics, and the user behavior. In this section we summarize the impact of each of these components and our non-goals.

Product release dynamics The security posture of a product is determined by how often new security vulnerabilities are introduced in the product's code, and how the vendor responds by shipping patches to end-users. At the time of release, a product is prone to a fixed number of (disclosed and undisclosed) security vulnerabilities. As time passes, undisclosed vulnerabilities will be discovered, and the number of known vulnerabilities for the product increases. Therefore, the number of known vulnerabilities for a product is influenced by three processes: new product releases, generation of new software vulnerabilities for each release, and the discovery and patching of vulnerabilities in subsequent versions. Figure 3.1a depicts a sample scenario for four successive releases of Firefox, released at times $t = 0, 35, 50, 75$ ($t = 0$ corresponds to 2012-09-11). Firefox 15.0.1 is prone to six vulnerabilities, all of which are undisclosed at the time of release. However, these vulnerabilities are made public at times $t = 34, 36, 53, 76$ (at $t = 34$ and $t = 76$, two vulnerabilities are disclosed). Firefox versions 16.0.1, 16.0.2, and 17.0.0 resolve three, one, and two of

these vulnerabilities, respectively.

Patches may be released before the official disclosure dates recorded in NVD. For example, Nappa et al. [99] observe that most users patch vulnerabilities in Chrome and Firefox browsers before disclosure. Such users are always protected from exploits that follow vulnerability disclosures. However, when the vulnerability is disclosed before the patch or when a user does not install the patch promptly, the user's machine will be vulnerable. The window of vulnerability spans the interval between the vulnerability disclosure date and the patch installation date. If attackers create an exploit for the vulnerability during this window, the vulnerable hosts will be susceptible to attacks; such exploits are often included in exploit kits, which are available to target large numbers of users on the Internet [56].

User behavior Even if the vendor releases patches for a product before or on the vulnerability disclosure dates, this does not protect users who run outdated product versions. A user's updating behavior reflects how often they update their products. For example, Figure 3.1b illustrates a sample user in our data set who installs the applications depicted in Figure 3.1a at $t = 5, 37, 58, 84$, respectively. Note that with each update, the user inherits the set of vulnerabilities in the new release.

We denote the version number of a specific product by integer i, with $i = 0$ denoting the initial release at time $t = 0$. We denote the release time of version i by T_r^i, and take $R(t) = max\{i : T_r^i \leq t\}$ to be the latest version available at time t. To model the behavior of users, we assume that following the installation of a product, once an update is available, the user waits for a random amount of time to perform an update. For example in Figure 3.1b, an update for the first version is made available at time $t = 35$, and the user initiates a software update at time $t = 37$, therefore the user's delay for the first update event is 2 days. Similarly, the delay for the second, and third update events are 8, and 9 days, respectively. Let T_u^k be the time that the user initiates the kth update event, and take S_u^k to be the delay associated with that event, then:

$$T_u^k = \min\{T_r^i : T_r^i > T_u^{k-1}\} + S_u^k. \tag{3.1}$$

The first term on the right-hand side of Equation 3.1 determines the first software release following T_u^{k-1}, marking the earliest time the user can trigger a software update. Note that the distribution of S_u^k influences the user's updating behavior; smaller values indicate new versions are installed immediately after their release, ensuring that the latest updates are always installed on the machine.

Note that a vendor can also affect the user's behavior by facilitating the installation

of updates, e.g. by providing an automatic update mechanism [30, 50]. Additionally, the purpose of a software update (e.g. to patch security vulnerabilities, to fix other bugs, to introduce new features) may also influence the user's delay in installing the update.

Non-goals We exclude exploits for undisclosed vulnerabilities from our threat model. Sometimes, vulnerabilities are exploited before their public disclosure in zero-day attacks [18]. However, such exploits are not commonly included in exploit kits, as this would likely result in the vulnerability's discovery and public disclosure, and patching cannot prevent these attacks. Additionally, we do not aim to quantify the risk of successful exploitation, which also depends on the use of alternative defenses that can block exploits or make them unreliable, e.g. intrusion prevention systems (IPS), address space layout randomization (ASLR), or data execution prevention (DEP). Instead, we focus on the practical security impact of vulnerability patching, which is the only defense that definitively eliminates the threat posed by the vulnerability in end-hosts.

3.3 Data sets and their processing

Our study draws from a variety of sources that collectively characterize the users' patching behavior, and allow us to assess the fraction of hosts susceptible to vulnerabilities and exploits, at any time. These include Symantec's Worldwide Intelligence Network Environment (WINE) [31], the National Vulnerability Database (NVD) [105], and release notes from software vendors.

3.3.1 Raw data

Table 3.2 includes a summary of our raw data sets, which we will elaborate on in the remainder of this section.

Patch deployment measurements The main data source required for a study on users' patching behavior, is a collection of measurements drawn from a large population of hosts, which can identify when each user has patched a vulnerability associated with the products under examination. The most common approach to obtaining such a data set is to scan the IPv4 address space repeatedly, by sending packets that elicit a specific response only from vulnerable hosts, in order to observe how these host are gradually patched [34,93,114,115,149]. This method requires that (1) the vulnerable program accepts incoming connections from the Internet, and (2) the program runs continuously. Consequently,

	Original data set	Subset used
Hosts	9 122 830	426 031
Analysis Period	02/2008 - 07/2014	01/2010 - 12/2012
Events	170 920 450	11 017 973
Applications	10	4
Files	2579	1279
Vulnerabilities	N/A	1822
Exploit Kits	N/A	14
AV Signatures	N/A	28
Exploited Vulnerabilities	N/A	21

Table 3.2: Summary of data sets

this method is appropriate for measuring the patching of server software (e.g. OpenSSL), but it would introduce substantial biases when examining the behavior of regular users, who typically utilize client-side applications (e.g. Internet browsers, media players, document readers and editors) and who may turn off their computers for extended periods of time. Furthermore, because we aim to model the users' updating behavior, we require a data set that can provide the exact time a user has installed, or updated, a product, and the exact application versions that have been installed at each event. Precisely identifying users across repeated network scans is challenging. For example, when utilizing the IP address as a unique host identifier, network address translation (NAT) or dynamic IP address allocations introduce confounding effects, by respectively combining multiple hosts under one address or by producing multiple addresses for a single host.

To sidestep these challenges, we utilize a corpus of patch deployment measurements collected by Nappa et al. [99], on user hosts including average users, as well as professionals, software developers, and security analysts, and mostly consisting of Windows XP/Vista/7 machines. These measurements were conducted by observing the installation of subsequent versions of different applications, and are derived from the WINE data set provided by Symantec [31]. The corpus includes daily measurements of vulnerable host populations, collected between 02/2008 and 07/2014 on 9 122 830 hosts worldwide. The measurements can be used to map records of binary executables, present on these hosts, to the corresponding application version. The set of security flaws affecting each version are extracted from NVD [105], using the Common Vulnerabilities and Exposures identifier (CVE-ID) of the vulnerability. For this study, we exclude users with unreliable event timestamps (explained in detail in Section 3.3.2). Furthermore, we analyze users' patching behavior over four of the products included in this data set, namely Google Chrome,

Mozilla Firefox, Mozilla Thunderbird, and Adobe Flash Player, and only include hosts that have recorded more than 10 events for at least one of these applications. This results in a data set consisting of 11 017 973 events over 426 031 unique hosts. Note that 99.3% of these events are records starting from 01/2010 until the end of 12/2012, which effectively reduces our study to a span of three years.

Exploits Although an open vulnerability is an important signal indicating the application could be exploited, this potential threat does not necessarily mean that the vulnerability will be exploited. In fact, few vulnerabilities are exploited in the wild; Nayak et al. [102] estimate that the average fraction is 15%. Therefore, in order to analyze the user patching behavior when the exploit is found in the wild, we collect the data from (1) public descriptions of Symantec's anti-virus signatures [136], and (2) metadata about exploit kits from Contagiodump [40]. The signature descriptions portray the threats detected by Symantec's anti-virus products, e.g. trojans, viruses, worms. If a security threat exploits a known vulnerability, the corresponding signature description indicates the CVE identifier of the vulnerability. From these descriptions, we collect 28 unique threats that exploit at least one of the vulnerabilities in our corpus, which contains 15 different vulnerabilities. Contagiodump includes the list of vulnerability exploits included in a number of popular exploit kits, along with the approximate inclusion dates.

Software release notes To find the first time a vulnerability is introduced in a product, and the subsequent patch time, we need to know the exact release dates for all product versions. For Firefox, Flash Player, and Thunderbird, we manually scrape release history logs, either provided on the vendor's website, or collected by a third party, to find out when each version is released to the public. We were unable to locate the historical release notes for other products; we therefore develop an automated technique, described in detail in Section 3.3.2, to infer the release dates for these products from field measurements.

3.3.2 Curated data

Our raw data presents several challenges for modeling the users' patching behavior and for evaluating the corresponding security threats. In this section we explain how we process and integrate the multiple data sets used in this study. Specifically, we explain how we extract reliable event timestamps from the patch deployment measurements, and how we process these events to detect the state of a host, i.e. the set of software installed on the machine, at any given time during our observation period. We describe a novel technique

for inferring software release dates from patch deployment measurements. This technique, coupled with manual scraping of vendor release notes, allows us to map a host's state transitions to the user's updating behavior. We also describe a technique for mining the release notes to determine the purpose of the update (e.g. security patch, new features). Finally, we discuss challenges introduced by irregular user and vendor behavior and how we resolve them.

Event timestamps Events in the WINE data set are characterized by three timestamps: report time (local time when the event took place), submission time (local time when clients submitted the report to Symantec), and server time (the time when server received the report). The server times, which are recorded in a consistent manner, were reported in [99] (we do observe that clients often batch reports and submit them later, often several days after events occurred). In contrast, client timestamps are not recorded in a controlled environment, and may be affected by unsynchronized clocks or by data corruption. To select a set of precise measurements, we define two conditions:

$$|t_{submission} - t_{server}| < 1 \text{ day} \quad (\mathbb{C}1)$$

This condition aims to eliminate hosts with unsynchronized clocks, as the difference between the times when the submission is sent and received should reflect only the time zone difference between the two hosts and the Internet propagation delay. This condition alone does not completely rule out incorrect local time, e.g. `1969-12-31 23:59:59` (which have resulted from data corruption or from the Symantec agent's inability to read the local timestamp). We eliminate these hosts by applying a second condition:

$$|t_{submission} - t_{report}| < 30 \text{ days} \quad (\mathbb{C}2)$$

40 755 519 update events, recorded on 2 535 966 hosts, satisfy both conditions. We further filter out users with fewer than 10 events for all the products included in our analysis. This leaves us with a curated data set of 11 017 973 events from 426 031 hosts.

Exploit release dates Similarly, we filter out all exploit records from Contagiodump that do not have a release date or that have dates with granularity of a year. This results in 13 exploited vulnerabilities present in 14 exploit kits. After combining them with our other sources of exploit information, we obtain exploit release dates for 21 vulnerabilities. For these exploits, the median time between a vulnerability disclosure and an exploit kit targeting the vulnerability is 17 days.

Product	Version	Timestamp
Firefox	3.6.13	2010-12-17
Firefox	4.0.0	2011-01-20
Firefox	3.6.15	2011-04-01
Firefox	3.6.16	2011-04-27
Firefox	4.0.1	2011-05-09
Firefox	3.6.17	2011-05-25
Firefox	5.0.0	2011-07-05
Firefox	7.0.0	2011-09-10

Table 3.3: Sample event sequence from a WINE user

Host states Each update event corresponds to a `(machine-ID, hash, timestamp)` tuple indicating that a certain file has been observed on the host on the given date. We use the method adopted in [99] to map each file hash to a certain product, and its corresponding version number. It follows that each observation can be translated into a tuple containing `(machine-ID, product, version, timestamp)`, allowing us to detect the installation of a software on the host. However, the WINE database provides no information on when a product has been removed, and we have to rely on future observations to extrapolate such events. A naive way to address this issue is to use successive events concerning different versions of the same product. As an example, if Firefox 3.6 is detected on a host, followed by the installation of version 4.0, we may deduce that the user updated the software and the former version is no longer present on the machine. However, it is possible to install multiple lines of the same product in parallel, in which case we would observe updates of both lines concurrently. Table 3.3 includes observations from a single host, which suggest that the user has installed Firefox 3.6 in parallel to other product lines.

We utilize the following heuristic to update the state of a machine after each event observation. Assume that an event at time t signals the installation of version v belonging to product line ℓ, and we have detected the presence of versions $S_{t^-} = \{(\ell_1, v_1), \ldots, (\ell_n, v_n)\}$ on the machine prior to the event. For each ℓ_i in S_{t^-}, we look at all following observations within 6 months of the current event, and if none of them points to the same product line, we remove the (ℓ_i, v_i) pair from S_{t^-}. We then add the (ℓ, v) pair, or update the corresponding pair in S_{t^-} if the same product line is already installed on the host, to obtain the state S_{t^+} after the event.

After processing all events, we obtain a host's state S_t (the set of software installed on the machine), at any time t. We then take the union of vulnerabilities that affect each version in S_t from NVD, as the set of vulnerabilities present on the host. The subset of

vulnerabilities that have already been disclosed, and that have been exploited, represent the machine's security posture at time t. The most important feature we extract from this times series is the portion of time during the observation period that the user has at least one known vulnerability on their machine.

Version release dates For three of the products examined in this chapter (Firefox, Flash Player, and Thunderbird) we can obtain the official release dates for each version by scraping version release notes from the vendor, or release histories collected by a third party. For the remaining product (Chrome), we tap into our patch measurement data to estimate release dates for each version. In previous work, Nappa et al. [99] identify the release date automatically, by selecting the first date when the version appears in WINE. However, we found that this approach can be unreliable in some cases. For example, Firefox has a rapid release cycle, with new official versions released every 6 weeks and three additional channels for early releases [26]. The binary that corresponds to a new version might appear in the wild half a year before it is made available on the release channel. Consequently, we found many instances in our data when update events happen before the actual release date to public. Another issue is that updates can be made available in stages, to different segments of the population, so that a large portion of users update at a later date. For example, Firefox 9.0.1 is documented to be released on 2011-12-21. But in reality, it was available to 10% of update requests from clients between 2011-12-28 and 2012-01-03, then 50% from 2012-01-05, and then all requests from 2012-01-12.[1]

On release dates, we typically observe a high volume of patching events. We thus first rank dates by the number of patching events, and then identify the patch release date as the earliest day among the highest ranking dates. For this study, we extract the top 10 days with the highest number of patching events, and take the earliest to identify the release date. We manually compared our results with a few dates that were reported in release notes, and found matching dates for all the versions we investigated, including Firefox 9.0.1.

Using the curated release dates we can obtain the patch deployment date for each of the CVEs present in our data set. For each product line affected by a CVE, we extract the release date of the first version unaffected by the vulnerability, and take the minimum of these dates to be the patch deployment date. For Chrome, Flash Player, Firefox, and Thunderbird, the median patch deployment date is 2, 8, 1, and 1 day(s) *before* the CVE disclosure. Therefore, software vulnerabilities are typically patched before they are publicly disclosed. However, as we will discuss in Section 3.4.2, this does not provide an adequate margin for users to update their software before the vulnerability is made public.

[1] Private communication with the Firefox release team.

User updates There are certain challenges to applying the model in Section 3.2 to real-world data. Extracting the user's delay in applying updates is non-trivial when there are multiple product lines present on the machine, or when the user downgrades to an earlier release. Moreover, multiple product lines of a software are sometimes developed in parallel by the vendor (e.g. Flash Player 10, and 11 lines), and obtaining the next release following each version is not a trivial task.

To study the frequency of irregular user behavior (parallel lines, version downgrades), we first obtain the number of events that result in the presence of more than one product line on a host. For Chrome, Flash Player, Firefox, and Thunderbird, 0.9%, 4.9%, 1.2% and 0.3% of events lead to the installation of more than one product line. The higher rate for multiple lines in Flash Player can be attributed to the fact that Flash Player plugins differ on various browsers. For example Chrome has an integrated Flash Player [54], whereas on Mozilla an external plugin needs to be installed [94].

For Flash Player, we further analyze the number of vulnerabilities associated with each product line. On average, in the presence of multiple product lines, 79.5% of vulnerabilities come from the lowest product version installed on the machine. Therefore, for our analysis of user behavior, we take the lowest application version on the machine as its current state, and only consider a new event as a version upgrade if it updates the lowest installed version. Note that for evaluating whether a machine is prone to a known vulnerability, we will still use the complete set of installed versions.

We perform a similar analysis on the number of version downgrades, and found that in 1.3% of events, users had downgraded the software on their machines. Given the rarity of these events, we concluded that their impact on our results are negligible.

Finally, for each state transition that updates the lowest installed version, we need to extract the user's delay in applying the software update. As detailed in Section 3.2, we first take the timestamps for the current, and previous update events (denoted by T_u^k and T_u^{k-1}), and extract the the first time an update was published for the previously installed version (denoted by T_r). The user's delay is then $S_u^k := T_u^k - \max(T_u^{k-1}, T_r)$. This means that we measure the users' delay from the day an update is available for the installed version, or the product installation date, whichever comes last; the latter takes effect when the user installs an outdated version. Note that successive software versions do not necessarily follow a chronological order, as multiple product lines are often developed in parallel. For each release, we take the next version, ordered by the version number, in the same line to be the update for that release. For end-of-life releases in each line, we pick the first version in subsequent product lines that has been deployed after the end-of-life release as the next logical update.

3.4 User behavior and its security implications

In this section, we characterize in detail users' patching behavior, and how that maps to their respective machine's vulnerability state, as measured by the fraction of time a known but unpatched vulnerability exists on a machine. This definition can also be extended to exploitable vulnerabilities, i.e. the fraction of time a known, unpatched vulnerability with a known exploits exists on a machine.

3.4.1 Modeling a user's patching delay

After processing our curated patch deployment measurements, we derive the sequence of patching delays (S_u^k) for every `(user, product)` pair. Assuming that the user's update delays are drawn from a probability distribution specific to the `(user, product)` pair, we need to determine its type and estimate its parameters. Note that some updating mechanisms do not require user intervention, such as silent updates in Chrome. However, even a silent updating mechanism may fail to deliver updates promptly (e.g. if the host does not have network connectivity, if the updating daemon encounters an error and exits, or if the user disables silent updates), so we expect that silent updates will also exhibit some variability in a large user population.

The exponential distribution and its discrete counterpart, the geometric distribution, are widely used to model random delays. A geometric distribution for S_u^k indicates that a user will perform an update every day with probability q, chosen independently of the outcomes on previous days. In previous work, the survival function for number of hosts that have not yet applied a security patch has been modeled as an exponential decay process [114, 115]. A geometric distribution for a user's delay in applying a software update leads to the same model for the survival function over the whole population. We test this hypothesis by performing a chi-squared goodness-of-fit test between each sequence and a geometric distribution whose parameter is calculated using a maximum likelihood estimate. The chi-squared test determines if the empirical distribution of the data and the hypothesized distribution, i.e. the geometric distribution, are statistically different, by binning observations and comparing the expected and realized number of observations in each bin. The output of the test is a p-value that indicates whether there is a substantial difference between the expected and observed frequencies. For small p-values we can reject the null hypothesis "the samples are drawn from a geometric distribution", meaning that the sequence is possibly generated by a different family of probability distributions. Table 3.4 summarizes our findings, for each product we have included the number of users tested, and the percentage that return p-values higher than significance levels of 5% and 1%. For

Product	Users	> 0.05	> 0.01
Chrome	167592	87.8%	97.6%
Firefox	21174	74.6%	93.0%
Flash Player	7722	98.2%	99.9%
Thunderbird	1857	86.5%	97.5%

Table 3.4: Chi-squared test results over update delays of different user. We cannot reject the hypothesis that these sequences are drawn from a geometric distribution.

Product	Chrome	Firefox	Flash	Thunderbird
Chrome	1.00	0.11	0.13	0.08
Firefox	0.11	1.00	0.33	0.29
Flash	0.13	0.33	1.00	0.18
Thunderbird	0.08	0.29	0.18	1.00

Table 3.5: Correlation between patch times of products

the test, we ignore users with fewer than 20 update events, as a well established rule of thumb for using the chi-squared test states that the expected frequency in each bin should at least be five [74]. Among the inspected client-side application, the behaviors of Firefox (Chrome) users exhibit the lowest (highest) match with the geometric distribution, with 74.6% (98.2%) of samples receiving a p-value higher than the 0.05 significance level. Nevertheless, our results show that for the majority of users, and across all inspected products, the geometric distribution is a good fit.

We also investigate whether the user patching behavior is consistent across different applications. For instance, does a user who updates Chrome frequently also exhibit lower patching delays for Flash Player? To test this hypothesis, we detect common users for each product pair, and calculate the Pearson correlation coefficient between their average patching delay, measure in days. Table 3.5 displays our results. The patching behavior for Chrome is somewhat independent from the patching behavior corresponding to other applications, since automatic updates minimize the user's control over update events. However for other products (which prompt users for download/installation of updates for a portion of our observation window), we observe higher correlation between the estimated profiles. Note that the highest correlation between profiles of different products is 0.33. While this value is statistically significant, it is not high enough to conclude that a user's delay in one product can be inferred from its profile in others; users that have installed multiple products on their machines, tend to update one more frequently, e.g. as a result of higher usage due to personal preference.

Country	Chrome	Firefox	Flash	Thunderbird
AU	10.6	16.3	30.1	15.1
CA	10.4	15.6	30.7	14.6
DE	10.9	15.3	24.9	14.7
FR	10.4	16.2	28.8	14.4
IT	8.8	15.9	26.1	13.5
JP	13.0	14.2	26.6	16.3
NL	10.4	15.2	28.5	14.7
PL	8.2	13.8	26.9	14.2
UK	9.2	15.7	28.3	13.9
US	10.5	15.5	32.1	15.4
All	9.9	15.6	29.7	15.2

Table 3.6: Average patch times by country.

Additionally, to analyze how a user's geographical location impact their patching delays, we group users by country, and calculate the average time to patch for the top 10 countries in our data set (the countries with the largest numbers of users). Table 3.6 displays our findings; the last row lists averages over the entire population. Note that Chrome uses silent updates to deliver patches to end-users, and therefore has the lowest patch times among all products. Firefox and Thunderbird versions prior to 15.0, and 16.0 (released on 2012-08-28, and 2012-10-09, respectively) download updates in the background and prompt users for installation. Flash Player versions prior to 11.2 (released on 2012-03-28) prompt users to download and install updates, and consequently exhibit the longest patch times. All three products switch to silent updates after the indicated dates, however these changes do not apply to the majority of our samples. For all four products, the patching behavior is remarkably consistent across the top 10 countries in our data set, suggesting that cultural differences among the examined countries do not play a significant role in the user patching behavior.

The above results essentially suggests that the users' response to new product releases are fairly *simple-minded*, in the sense that they can be well-modeled using a one-parameter distribution: an end-user will initiate a software update at every date with probability q, chosen independently of other variables, such as the outcome for previous days, and the types of improvement incorporated into the application since their last update. The average patching delay of the user is then given by $1-q/q$. This single-parameter characterization greatly simplifies our analysis on the security implication of user behavior. In what follows, we examine the relationship between patching delays and the vulnerability state of a given host over long observational periods; note that we shall only rely on sorting users by their

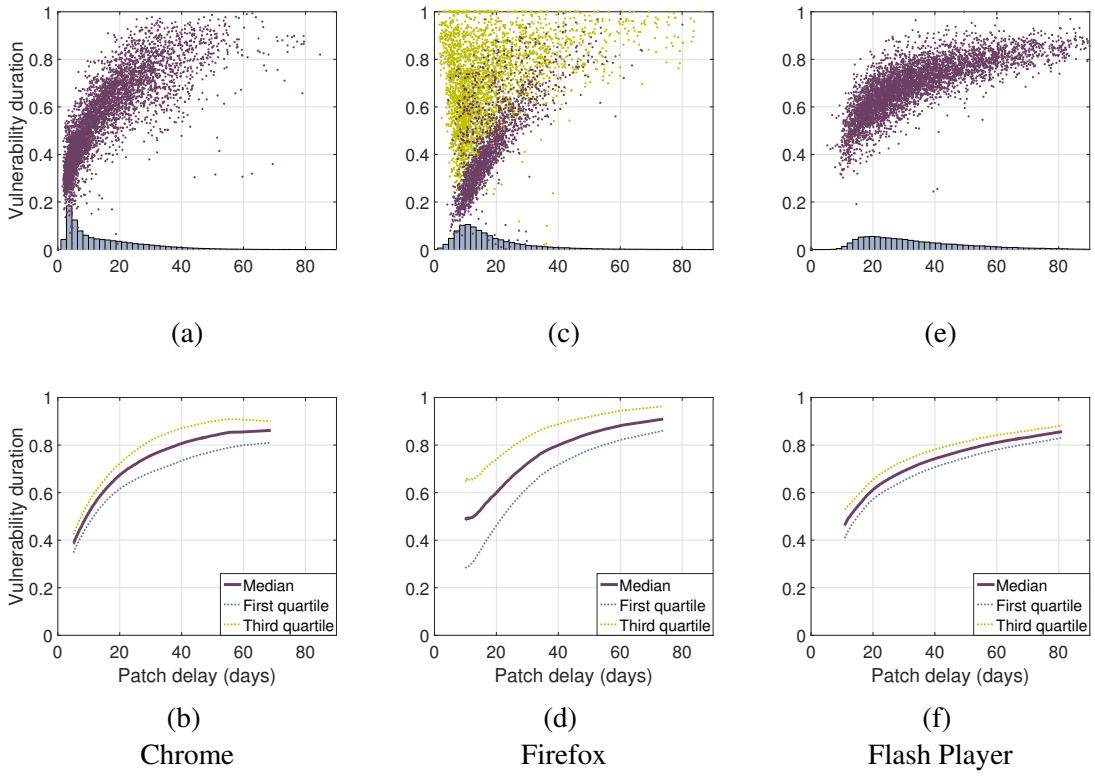


Figure 3.2: Scatter plots of normalized vulnerability duration vs. average user delay in days (top), and the mean, and first and third quartiles for different user types (bottom). Each point in the scatter plots corresponds to a single user. In 3.2c the yellow/red dots are users active in 2010/only active starting 2011.

average patching delay rather than the actual distribution of their delays.

3.4.2 Vulnerability state

We take the fraction of time that a host remains susceptible to at least one known vulnerability as an indicator of its security posture or *vulnerability state*. Note that hosts in our data set are observed during different time periods, which vary both in length and the starting point. Therefore, these indicators are calculated over the observation period specific to each individual host.

Figures 3.2a, 3.2c, and 3.2e display scatter plots of this vulnerability measure for Chrome, Firefox, and Flash Player, respectively. For each figure, we have randomly selected 5000 users, where each point represents one user. A point's x and y coordinates correspond to the average patching delay of that host, and its measured vulnerability state. The histogram at the bottom of each plot shows the distribution of users with respect to

their average patch time; these are generated using the entire population of users with an observation interval at of at least one year, resulting in 140 588 sample points for Chrome, 64 016 for Firefox, and 55 042 for Flash Player. We have not included Thunderbird in this analysis, since the 5243 samples we were able to gather were not enough for an accurate demonstration. The plots suggest that users with higher delays experience longer vulnerability durations compared to those with lower patch times. Furthermore, we see a higher concentration of users who patch quickly for Chrome and Firefox.

Vulnerability state as a function of average patching delay To quantify the relationship between user behavior and vulnerability state, we further group users with similar behavior by sorting them according to their estimated patching delay, and create bins consisting of 500 users. We then calculate the median vulnerability duration, along with the first and third quartiles in each bin, allowing us to quantify the vulnerability state as a function of users' patching behavior; the results are illustrated in Figures 3.2b, 3.2d, and 3.2f. We observe that a user with equal delays in each product experiences similar vulnerability durations. At 20 days, the average user will remain vulnerable for about 60% of the time, at 40 days this value increases to 80%.

Across the three products, Chrome users clearly are more likely to have a lower patching delay (as shown in the histograms), likely the effect of silent updates, whereas Flash users are the most tardy. Interestingly, given the same average delay, the amount of vulnerabilities a user faces is very consistent across all products. Note that these relationships are shown as functions of the average patching delay, which we have taken as a factor for sorting users; this consistency further validates the single-parameter characterization of users, detailed in Section 3.4.1.

Outliers In Figure 3.2c we see high variability in vulnerability durations for users with similar patch times. Upon further inspection, we discovered two vulnerabilities for Firefox, `CVE-2010-0654` and `CVE-2010-1585`, that were published on 2010-02-18 and 2010-04-28, but first patched on 2010-07-20 and 2011-03-01, respectively. As a result, users that have been observed during 2010, have remained vulnerable for most of that year, regardless of their behavior. In Figure 3.2c, we have used a lighter color to display hosts that have been observed at any time during 2010. The rest of the hosts (those that have opted in after 2010), exhibit similar variability to Chrome and Flash Player users.

We also notice other outliers in the scatter plots in Figure 3.2, those with much lower (or higher) vulnerability durations compared to other similar users. Upon further inspection, we concluded that the majority of these sample points belong to users who suddenly stop

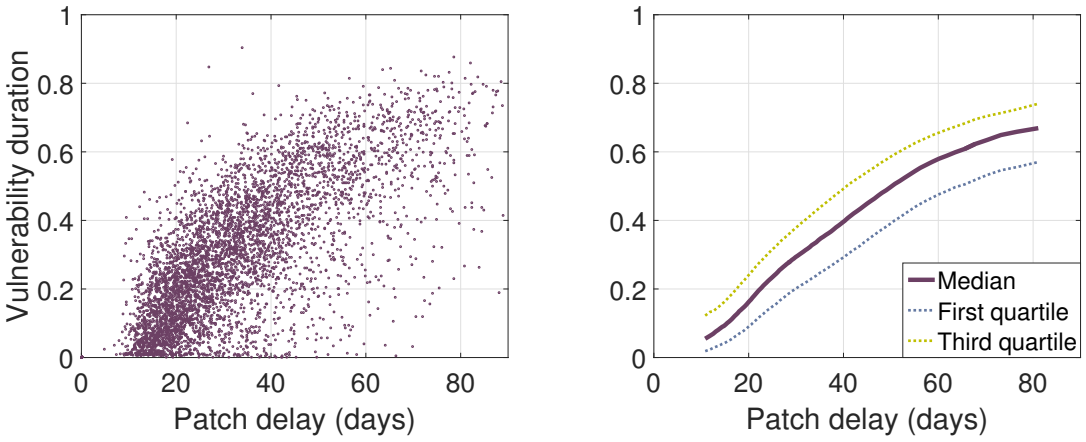
updating their software. If these users get stuck at a vulnerable release, then they will have a higher than normal vulnerability duration, otherwise they will remain safe for most of the observation period. Moreover, for Firefox, a significant portion of outliers are users who install new releases before their official release dates, and are therefore less susceptible than other typical end-users.

Comparison across products and the limited benefit of silent updates We further calculated, for each product, the average vulnerability duration, over all users of each application in our study. Note that these values are affected by the following properties of each product: (1) the distribution of different user types (the histograms in Figure 3.2), and (2) the expected vulnerability duration for different users (Figures 3.2b, 3.2d, and 3.2f). For Chrome, Firefox, Flash Player, and Thunderbird, the average host was susceptible to at least one vulnerability for 53.5%, 59.9%, 68.7%, and 55.7% of days. It follows that the improvement in security provided by different updating mechanisms in these applications is marginal.

A host's vulnerability state is influenced by two conditions. First, for a single vulnerability, the patch should be applied before the vulnerability is publicly disclosed. Nevertheless, even if the user misses the disclosure date, the damage can be minimized by prompt patching, since the amount of time the hosts remains vulnerable is proportional to its patching delay. However, when taking into account successive vulnerabilities, if the user does not apply the patch before the next vulnerability is disclosed, the clock is reset, in the sense that they will now have to apply a new patch to secure their machine against attackers.

Quantitatively, for Chrome and Firefox, our data set includes 124, and 114 vulnerability disclosures between 2010 and 2012, resulting in an average of approximately 10 days between successive disclosures. However, our estimated results show that the average patch times for users of Chrome and Firefox, is 9.9 and 15.6 days, respectively, meaning that users often cannot patch a vulnerability before the next one is discovered. For Chrome, adopting silent updates does not seem to provide the necessary margin to see any significant effect on the vulnerability duration of hosts, meaning an effective solution for vendors is to lower the amount of vulnerable code introduced into the product, in addition to facilitating the updating process for users.

Breakdown of the vulnerability window Note that the vulnerability of a machine can be caused due to the vendor's failure to release a patch before a vulnerability is disclosed, or the user's negligence in installing the patch. To further study the degree of control that users have over the vulnerability state of their machines, we examined the portion



(a) (b)

Figure 3.3: Scatter plot (left) and mean, and first and third quartiles for exploited vulnerabilities of Flash Player.

of the vulnerability window caused by the latter. We found that, summed over all users, for Chrome, Flash Player, Firefox, and Thunderbird, 59.3%, 61.6%, 47.9% and 55.7% of days where a machine was susceptible to a known vulnerability was caused by user negligence. This means that a significant portion of the vulnerability window is incurred by the software vendor, and prompt installation of software updates can roughly cut the vulnerability duration of end-users by half.

3.4.3 Susceptibility to vulnerability exploits

Even though we consider being prone to known vulnerabilities as a security risk, this does not necessarily translate into an imminent threat for the user, as the machine can only be breached through a real exploit. However an unpatched vulnerability that has a known exploit presents a direct threat to the host. If the host has such a vulnerability, and the user navigates to a webpage that embeds targeting and redirection code from a kit that has bundled the vulnerability exploit, the host will likely be infected [56], often without the user noticing the fact. We perform a similar study on the percentage of days that a host remains susceptible to an exploitable vulnerability. Figures 3.3a and 3.3b display the scatter plot and vulnerability trends for 15 exploits of Flash Player, which is installed on 17.4% of the hosts in our data set. We did not have a sufficient amount of exploits for Chrome and Firefox (we were only able to find one known exploit for Chrome, and two for Firefox) to perform a similar analysis.

Comparing these plots to Figure 3.2, we observe the same trend for the correlation

between average patch times and vulnerability states. However, for similar user patching delays, we generally see lower risk for known exploits, compared to disclosed and unpatched vulnerabilities. This is to be expected, given the fact that each Flash version suffers from only a few exploited vulnerabilities at most. Nevertheless, we observe that many hosts in our data set are susceptible to known Flash exploits more than 50% of the time, highlighting the threat that exploit kits present to end-host security, and further supporting our findings from Figure 3.2

3.5 Discussion

Our results represent a first step toward understanding the *deployment-specific barriers* for updating software. We observe that user behavior can be modeled well using a simple and elegant mathematical model, which depends on a single parameter: the user's average patch deployment delay for a given product. We do not observe clusters of users with respect to the patching delay or the vulnerability state. Moreover, the willingness to patch does not vary significantly across different countries. However, users seem to exhibit different patching behaviors for different products, suggesting that vendors may be able to influence the users' patching delays. For example, Figure 3.2 suggests that the vulnerability duration for users of Flash Player exhibits a lower variability than for Chrome and Firefox users, despite the lack of a silent updating mechanism. This consistency may result from the fact that users are compelled to upgrade when sites that provide streaming media content remove backward compatibility for older Flash versions. A deeper understanding of these barriers could guide enhancements to software updating processes, leading to improvements in users' security posture.

The simple model that appears to govern the users' behavior significantly simplifies the analysis of optimal patching strategies. Prior work in this area often makes unreasonable assumptions, e.g., that, once a patch has been created and tested, its deployment is instantaneous [107]. Such analytical modeling frameworks may also enable comparing the impact of various design choices when implementing software updating mechanisms for the large-scale distributed systems of the future (e.g. the Internet of Things), where user behavior is likely to play a key role.

3.6 Related work

Patching behaviors of both end-users and servers, and the role of vendors in the software updating process, have been extensively studied in the literature, through case studies

of specific vulnerabilities, and comprehensive examination of various products and software flaws. Moore et al. [93] probed hosts infected by the Code Red worm and found a slow patching rate until the worm began to spread again. Rescorla [115] studied a 2002 OpenSSL vulnerability and observed two waves of patching: one in response to the vulnerability disclosure and one after the release of the Slapper worm that exploited the vulnerability. Ramos [114] analyzed several remotely-exploitable vulnerabilities and reported a slow decay rate in some cases and some vulnerabilities that did not decay at all. Yilek et al. [149] scanned OpenSSL servers affected by a 2008 key generation vulnerability in the Debian Linux and found a high patch rate in the first 30 days, followed by patching waves for the next six months. Durumeric et al. [34] showed that more than 50% of servers affected by the recent Heartbleed vulnerability in OpenSSL remained vulnerable after three months. Zhang et al. showed that, even after patching OpenSSL, most web sites remained vulnerable to a man-in-the-middle attack because they had not revoked certificates that may have been compromised owing to Heartbleed [151]. The rate of updating is considerably higher for systems that employ automated updates [30, 50]. Dübendorfer et al. [30] suggested that Google Chrome's silent update mechanism is able to update 97% of active browser instances within 21 days. Gkantsidis et al. [50] concluded that 80% of Windows Update users receive patches within 24 hours after their release. Nappa et al. [99], measured vulnerability decay in 10 client-side applications and identified security threats presented by multiple installations of the same program and by shared libraries distributed with several applications.

However, the factors that delay vulnerability patching are not well understood. Schneider et al. [95] proposed several hypotheses, including an under-appreciation of risks and a fear of destabilizing other software. Frei et al. [46] reported a typical Windows user must manage 14 update mechanisms (one for the operating system and 13 for the other software installed) to keep the host fully patched. Vaniea et al. [142] suggested that negative experiences with past updates affect the users' willingness to deploy patches. Tajalizadehkhoob et al. [137] present an empirical analysis of security and patching practices in shared hosting providers, identify four latent factors that capture security efforts, and investigate their impact on compromise rates.

On vulnerability discovery models, Alhazmi et al. [5] examine four different vulnerability discovery models, and fit the proposed models using actual data from three operating systems; they are then evaluated using chi-squared tests and the Akaike information criteria; demonstrating that four of the examined models are a good fit for real world data. Anbalagan et al. [9] model the discovery of vulnerabilities as a Poisson process. The vulnerability disclosure interval is thus exponentially distributed, and the parameter of the

exponential distribution is chosen to follow the Gamma distribution. The model is further tested using Firefox and SeaMonkey.

3.7 Conclusion

In this chapter we have conducted an in-depth analysis of the dynamics between vendors and consumers when it comes to software security. To the best of our knowledge, this is the first study on how individual behavior can influence the security state of a user's machine over long observational windows, where the continuous discovery of software vulnerabilities, patch deployment by vendors, and the installation of patches, create windows of opportunities for malicious entities to exploit open vulnerabilities on a user's machine. We have shown that frequent updating, and steps taken by vendors to speed up the installation of software patches on hosts provides marginal benefits, especially when the rate at which new vulnerabilities are introduced into the product's code is high. Consequently, developers should exercise due diligence when shipping new products to end-users, as the detrimental effects of releasing vulnerable applications to the public often cannot be eliminated by prompt patch deployment.

Although we have shown that users' behavior can effectively be explained using a simple single-parameter model, we are not able to build similar profiles for vendors. This is partly due to lack of a large data set on software vulnerability cycles. The set of unique vulnerability disclosures and patch deployments concerning the products under examination was to small to carry out a comprehensive study on product behavior. Such an analysis could close the loop when assessing the security posture of an end-user, by predicting the host's vulnerability state across different products, or for new products entering the market. Furthermore, predicting a user's profile by compiling a set of features that can explain their behavior, such as profession and Internet connectivity, is another direction for extending the current study.

CHAPTER 4

Numerical Fingerprinting of Internet Hosts

4.1 Introduction

The development of open-source network scanners [35, 106] has provided researchers with large amounts of raw information on Internet hosts, which can be used to reveal misconfigured servers, uncover hosts susceptible to specific software vulnerabilities, and collect statistics and trends on adoption of various protocols and technologies. Moreover, the diverse set of measurements on individual or groups of hosts, resulting from these global scans of the Internet, can be leveraged in machine learning models for a wide variety of learning tasks, including supervised classification, e.g. for quantifying maliciousness, or for forecasting infection, as well as unsupervised modeling of factors that drive the observed attributes of an arbitrary host.

At the same time, neural networks and deep learning methods have gained popularity in the area of AI and machine learning, due to their success in many supervised and unsupervised learning tasks. Numerous deep learning techniques have been developed in the past few years, outperforming the previous state-of-the-art learning methods in their respective fields [4, 51, 52], based on their performance (e.g. accuracy for classification tasks, or log-likelihood estimates for generative models) on standard data sets. For instance, convolutional networks have been successfully applied to many image recognition tasks, and recurrent neural networks have gained popularity for natural language processing.

In this study, we develop a framework for scalable analysis of Internet hosts by leveraging advances in both network scanning and machine learning algorithms. This framework can prove useful in a variety of applications involving, for example, visualization of data sets, supervised classification, and quantifying hosts similarities. We demonstrate these using three specific case studies: (1) detection of (actively or potentially) malicious IP addresses, (2) inferring missing attributes of a host, and (3) categorizing networks.

Our proposed framework can be used on any network scan data set; in this chapter, in

order to develop and evaluate the performance of our framework, we tap into Censys [32], a large database of raw information obtained by regular scans on 20 different ports, providing us with parsed scan results, in addition to routing and geolocation information, on roughly 150 million IPv4 addresses. 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 Censys records, which are stored in the human-readable JSON format. Such a *binary representation* can drastically reduce the memory and computation required for analysis of large collections of IP addresses, at the cost of only minimal information loss. It is worth mentioning that while we develop this method for JSON documents, the same concept can be applied to find numerical representations of other documents with tree-like structures, e.g. HTML/XML, as well. We shall then examine the efficacy of our binary representations in a supervised learning setting.

Furthermore, we explore the use of generative stochastic models on our binary representations of the information gathered by global scanners, in order to find low-dimensional numerical representations, or *embeddings*, of Internet hosts. The explored techniques attempt to learn efficient representations of observed attributes, by learning stochastic encoders and decoders that convert between the original data vectors (i.e. binary representation) and their low-dimensional embeddings. We then evaluate and compare two candidate algorithms for dimensionality reduction, using the reconstruction loss (by passing samples through the encoder and decoder) of trained models over a held-out test set as a performance criterion. More specifically, we train and compare two types of unsupervised models: (1) restricted Boltzmann machines (RBM) [59, 133], a probabilistic graphical model for learning binary latent embeddings, and (2) variational autoencoders (VAE) [73, 117], a recent method developed for training deep graphical models on unlabeled data, which can be utilized to map our proposed binary representations of hosts to continuous latent variables. Our experiments demonstrate the advantage of VAEs in constructing low-dimensional, yet high precision, embeddings of our binary representations, owing to their deep structure and the use of continuous latent variables. Note that for image recognition, VAEs have been shown to be capable of extracting meaningful features and generating synthetic samples in image processing tasks, e.g. from the MNIST database of handwritten digits [73]. Hence, for our main analysis, we train multiple VAEs on our binary representations of hosts for varying number of dimensions in the latent space, and show that we can learn meaningful representations with as little as two dimensions, and very rich representations for up to 50 dimensions. These latent representations can also be interpreted as factors that drive the observed measurements, such as location, type of the host (e.g. web/mail server, router,

end-user, and so on), or even more abstract factors such as negligence of network administrators that may, for instance, result in open ports or expired SSL certificates.

In addition to enabling faster computation, a main advantage of our low-dimensional embeddings is that they are amenable to visualization. In particular, when mapping IP addresses to two latent variables, the embeddings can be drawn as scatter plots or by visualizing the density of points, e.g. using kernel density estimation (KDE). Furthermore, for specific groups of hosts, such as different types of web servers, or devices, we can employ principal component analysis (PCA) to project higher dimensional embeddings onto two dimensions, thus enabling us to visualize any subdivision of the global population. Such a tool can be utilized by analysts and network administrators to inspect the distribution of any collection of hosts, e.g. those belonging to a particular organization, or blacklisted due to a certain malicious activity, in order to identify clusters of IP addresses sharing a common trait, or to compare multiple networks.

The output of our technique is therefore two types of representations, or *fingerprints*, of Internet hosts: (1) high-dimensional, yet sparse, binary feature vectors, and (2) low-dimensional latent embeddings that can explain the behavior of the observed variables, i.e. binary representations, with high precision. The presented methodology can then be used to automatically define and share features across many applications, thus removing the need for extracting hand-curated features, e.g. for risk forecasting [86]. Additionally, instead of focusing on a specific aspect of a host, such as techniques for fingerprinting operating systems [130, 131, 146], or physical, e.g. Internet of Things (IoT), devices [43, 75, 84], our proposed method produces representations that combine attributes across many aspects and protocols. This then leads to flexible fingerprints that are adoptable for a variety of learning tasks. We explore a number of these applications in this study, summarized as follows:

Supervised classification We evaluate both types of fingerprints for supervised classification, more specifically for detection of malicious IP addresses and prediction of hosts exhibiting high risk of turning malicious in the future. We use two IP address blacklists, namely PhishTank [112], and hpHosts [63], for training and evaluating a set of classifiers, and comparing the predictive power of models over binary/latent representations. We demonstrate that lower dimensional embeddings can results in faster models, while requiring more pre-processing, i.e. computing and storing the latent embeddings. However, we observe that our binary fingerprints result in the most accurate predictions, achieving a true positive rate of 90%, while keeping the false positive rates at low as 6.3% and 8.6%, for PhishTank and hpHosts, respectively. Note that the evaluated models can process hosts much faster than content-based approaches [3, 134, 152], suggesting an efficient divide-

and-concur method for fast identification of highly suspicious samples.

Inferring missing attributes We also explore the application of supervised learning techniques in inferring attributes of a host that can potentially be masked by a network administrator. For instance, the name of software deployed on a server may optionally be included within headers/banners. We examine whether it is possible to infer this information from other observed attributes of a host, including measurements from other ports, and the location of said host. Our results demonstrate that masked attributes can be inferred with high precision, unless all relevant information, i.e. headers/banners from other protocols, that may be used for inference is masked as well. However, we show that even with thorough masking of information from all protocols, the information in question may be leaked through unmaskable attributes, such as a server's locations, and the set of services offered to clients. Our second application therefore demonstrates a possible security threat, as miscreants can selectively target hosts that use software with known exploitable vulnerabilities, even if the fact that the host is using this vulnerable software is not directly disclosed, in order to compromise Internet-facing machines in an automated fashion.

Host similarity Our latent variable model can also be considered a non-linear dimensionality reduction technique, where the distance between points in the latent space can be used as a similarity metric. This technique enables fuzzy matching of IP addresses, which, when coupled with the low dimensionality of these fingerprints, allows us to use algorithms such as k-nearest neighbors (k-NN) to query for similar hosts from a large corpus of IP addresses. Thus, we also use k-NN based classifiers for detection/prediction of malicious hosts, and inferring missing attributes. Our results show that these models can often achieve similar performance to other state-of-the-art classification algorithms, supporting our claim that the use of deep models can produce a meaningful, and robust, similarity measure for IP addresses.

Network signatures Finally, we explore techniques for aggregating our host-level information over network boundaries, namely autonomous systems (AS), in order to extract numerical representations of networks. Such a characterization of a macroscopic unit, using fine-grained attributes obtained at the host-level, can be used to capture many granular aspects of the underlying entities (e.g. businesses or ISPs), that were previously not feasible. We explore this technique for detecting various types of ASes, namely business, consumer, education, government, and information (hosting and content delivery) networks. We observe a significant performance boost as compared to such a categorization at the

IP-level, owing to the ability of our technique in leveraging the joint distribution of hosts in a network (collection) of hosts.

One of the drawbacks of most common security analysis of the Internet is that they often tend to focus on one or a small set of properties. This microscopic view often prevents us from gaining a broad macroscopic view of the Internet across many features. The techniques presented in this chapter can be utilized to make generalized statements about hosts based on features acquired from various, often orthogonal, types of measurements. Moreover, once the numerical fingerprints of hosts have been obtained by computing and storing binary/latent fingerprints offline, it is possible to use fast models in order to perform classification/inference in an online fashion. This would make it feasible to perform scalable analysis of large data sets of IP addresses, without the need for special tools such as distributed computing platforms.

The remainder of the chapter is organized as follows. In Section 4.2 we go over the data sets used in our study, and the pre-processing steps taken to prepare them for analysis. In Section 4.3 we explain in detail how we convert Censys records to binary vectors in order to feed them into a machine learning algorithm. In Section 4.4, we go over the matchematical model of VAEs and RBMs, and evaluate/compare their performance for finding latent embeddings, and visualization of Internet hosts. We examine applications of our technique for the aforementioned case studies in Section 4.5, discuss our findings in Section 4.6, and conclude in Sections 4.7 and 4.8.

4.2 Data sets

For this study, we use IP intelligence collected by Censys [32] to gather information about arbitrary IPv4 addresses. The information offered by Censys includes regular global scans over 20 different ports, geolocation information provided by the MaxMind GeoLite2 database [89], and autonomous system (AS) and routing information provided by Merit Network [92], and Team Cymru [138].

The Censys database contains JSON records reporting all the information available for a given IP address, including parsed fields from well-known protocols, such as HTTP headers (port 80), parsed SSL certificates (ports 443, etc.), latitudes and longitudes, AS numbers, and so on. This information is updated regularly, with full dumps of what is known about all discoverable hosts on the public Internet, and is available for download by researchers. For this study, we use five global snapshots, collected on 7/1, 7/16, 8/1, 8/16, and 11/1 of 2017; each scan includes roughly 150 million records. We concatenate and randomly sample from the first four snapshots to generate the different data sets that are used in

our experiments for training/evaluation of our host-level machine learning algorithms. We use the snapshot obtained on 2017-11-01 for evaluating our method of quantifying network signatures, in order to align our measurements with AS boundary information, and obtained labels, to be described shortly. Unless otherwise stated, all data sets used throughout the remainder of this chapter have been selected by sampling from the four snapshots from July and August of 2017, and are independent of data sets used in other sections.

We collect IP addresses listed by PhishTank [112] and hpHosts [63], on every day during July and August of 2017, which we will use to produce labels to train classifiers for identifying malicious hosts. These data sets include records on 123 823 and 15 026 unique IP addresses for PhishTank and hpHosts, respectively. Note that the former is a blacklist focusing on phishing, while the latter also targets ad/tracking and malware websites. We will discuss the effectiveness of our methodology for profiling both types of malicious servers in Section 4.5.

For obtaining AS boundaries, we utilize the MaxMind GeoIP2 ISP database [89], containing a snapshot of IPv4 address blocks belonging to 58 622 ASes from 2017-10-31. In addition to block definitions, this database also offers the name of each included network, which we will use for manually inspecting predictions in Section 4.5. Finally, to procure labels for classification, we query the MaxMind GeoIP2 precision services [90], providing us with user types corresponding to queried IP addresses, which we will process to obtain labels for the inspected learning task.

4.3 Feature extraction

To extract features from records collected from Censys, we develop a semi-automated algorithm to parse JSON documents, learn their structure, and generate binary features that describe each document. In the remainder of this section, we will present our general algorithm, and elaborate on how we have tuned it for our data set to obtain meaningful features from Censys records.

4.3.1 Encoding tree-like documents into numerical vectors

Tree-like document models such as JSON, or HTML/XML, 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 4.1 illustrates an example for describing IP addresses. Each leaf (or field) in the tree contains an attribute of the IP address, e.g. its origin country, while other nodes (interme-

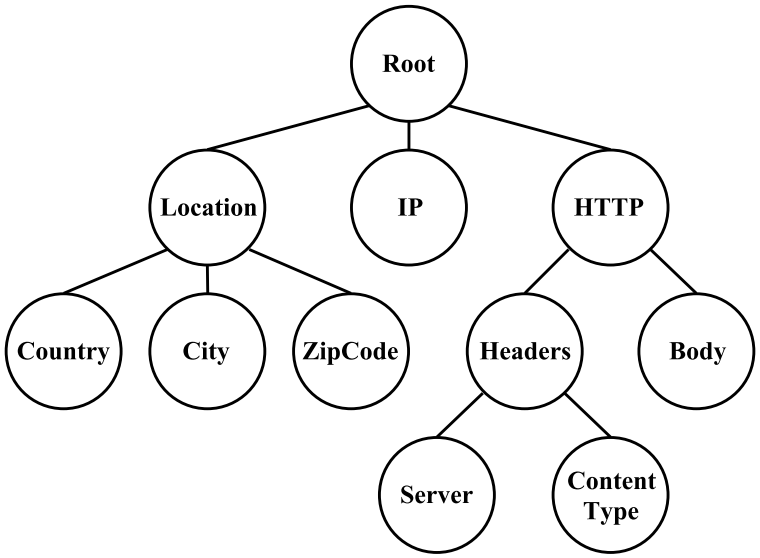


Figure 4.1: An example tree-like document describing a host.

diate fields) are sub-documents that encapsulate other nodes/leaves. Such data models can be used to serialize documents into human-readable text, or facilitate programmatic data manipulation. However, machine 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.

Note that 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 vectors 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 corresponding to Censys records contains more than 2000 leaves, with a maximum depth of 11. 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 propose a novel algorithm that can automatically recognize the structure of sample documents, and generate feature vectors based on a learned schema. However, 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, such as HTML/XML.

JSON (JavaScript Object Notation) is an open-standard format for describing and serializing structured data. 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.TLS.Certificate.Subject.CN`, containing the common name of the subject of an SSL certificate obtained from the TLS handshake of the HTTPS protocol on port 43. We use a simplified version of the JSON schema [67] 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 for describing the underlying document tree:

- *type*: Defines the data type(s) that the document can take, and is a subset of the supported data types. For this study we use the following data types: null, boolean, number, string, dictionary, and array.
- *properties*: Defined for dictionaries, a set of (key → value) mappings, where keys are names of the properties that may be included within the dictionary, and values are valid JSON schema that describe the property.
- *items*: Defined for arrays, a JSON schema for items of an array. We assume that all items conform to the same schema.
- *values*: Additionally, for numbers and strings, we append the list of encountered values for each field to the schema. We will later use these for generating features.

For learning a schema, we apply the simple recursive procedure defined by Algorithm 1, for every sample document in the examined data set. Once we have iterated over all documents in our training set, we traverse the built schema, and define features for every (intermediate) field `X`, based on the following rules:

- *type*: When |schema.type| > 1, i.e. the object can take multiple types, we use one-hot encoding to reflect the type of the object as a set of binary features, e.g "`X is number`" or "`X is string`".
- *dict*: For dictionaries, we convert each property in the dictionary into the feature "`X has property Y`", taking a value of one if the given key is present. We also keep track of required fields (those that are present in every document), so as to avoid redundant features.

Algorithm 1 Extending a JSON schema

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

- *boolean*: For booleans, we copy the value as is to define the feature "`X = True`".
- *number*: For numerical values, we bin sample values and use one-hot encoding to express the bin corresponding to each sample: "`X ∈ [A, B)`".
- *string*: For strings, we tokenize the recorded values, and tokens into the binary features "`X has token Y`". For categorical variables (to be described shortly), we simply use one-hot encoding for each unique value, i.e. "`X = Y`".

Note that alternatively, numbers can also be mapped as is to a single feature. However, this produces mixed feature types (i.e. both binary and real-valued features), which can be problematic when used in machine learning models without proper normalization. Therefore, for this study we have employed binning combined with one-hot encoding to avoid this issue. We recursively apply the above rules to the learned schema over sample documents and all its sub-schema, and concatenate all the extracted features to construct a binary feature vector.[1]

4.3.2 Fine-tuning

While the algorithm presented above can summarize documents in numerical vectors out-of-the-box and with minimal supervision, one can tune its parameters, motivated by domain expertise, to extract more meaningful features for a specific application. For instance, one

[1]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, leveraging our earlier assumption that all array items conform to the same schema.

can control the number of generated features, by pruning sparse fields from the schema, and setting the number of bins for numerical fields, or filter tokens that are too short/sparse. Below we summarize the operating parameters that we utilize for Censys. These parameters have been set to avoid features that are sparser than 0.05% (500) of all sample documents in the data set used for feature extraction, comprised of one million records, resulting in roughly 10 000 features. It is possible to relax this constraint to produce more features, however that would in turn result in slower machine learning models, and may also lead to over-fitting due to the increased number of free parameters.

First, we prune the learned schema by dropping fields that are present for less than 0.05% (500) of all inspected samples. We bin numerical fields into $\min(100, n/500)$ bins, where n is the number of recorded values for the field being inspected. We also convert timestamps to unix times, and treat the resulting value as a numerical field.

For text fields, 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 tokenizing non-categorical text, we utilize 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.

Finally, we ignore fields that cannot not produce meaningful features, such as the IP address of the host, SHA and MD5 hashes, and other fields that can act as unique identifiers, as they behave like random noise and are not meaningful attributes in a machine learning setting. We also ignore HTTP and CWMP bodies, and timestamps added by the scanner (this doesn't include timestamps such as validity start and end dates for certificates).

Our set of rules and parameters results in 11 156 binary features. For a specific sample, features with a value of one can be regarded as tags that have been associated with the corresponding host. We have included descriptions for a small number of our features in Table 4.1, demonstrating how we can extract data from various sections of Censys records, expressing both the structure of documents (e.g. open ports, usage of protocols), as well as granular attributes extracted from field values (e.g. origin country, certificate validity periods). 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

Field name	Data type	Attribute
Root	Dictionary	has property “FTP”
CWMP.Headers	Dictionary	has property “Server”
DNS.OpenResolver	Boolean	=True
Location.Latitude	Number	$\in [33.82, 34.02)$
HTTPS.TLS.Certificate.Validity.End	Number	$\in$ [2019-12-12, 2020-01-01)
Location.Country	String	= “United States”
IMAP.Banner	String	has token “dovecot”

Table 4.1: Sample features extracted from Censys records. The first and second columns contain the path to a specific (intermediate) field in the learned document tree (separated by dots), and its data type. The last column describes a single binary feature extracted from said field.

transform JSON documents to their binary fingerprints on a single thread of an Intel Core i7-7770K processor; this number is computed using batches consisting of 1000 records.

Note that the sparsity of our binary fingerprints is both due to the nature of extracted features (i.e. one-hot encoded values, and tokenized strings), as well as the structure of the underlying documents. A typical server offers only a small subset of all protocols probed by Censys scanners. Hence, sample records often occupy a small sub-tree of a larger document tree, learned through the procedure detailed in Algorithm 1. Nevertheless, contrary to other types of tree-like documents such as webpages’ HTML codes, where each page can have its own unique structure, individual fields are shared between many samples, allowing our technique to capture their data through the described binary tags. This property of our data set results in information heavy fingerprints, as evident by the length of resulting feature vectors, the number of tags associated with a typical host, and the dimensionality of latent representations, as we will discuss in the next section.

4.4 The latent variable model

In order to model the distribution of samples (hosts) in our data set and extract latent features from them, we compare two different latent variable models, namely variational autoencoders [73, 117], and restricted Boltzmann machines [59, 133]. Both models provide a framework for generative modeling, and can be used to learn low-dimensional embeddings of high-dimensional data. These two candidates have been selected due to their compatibility with binary observations, while making different assumptions on the underlying data generating distribution (i.e. using binary and continuous latent states). Note that our main objective in applying a latent variable model to the extracted attributes detailed in the pre-

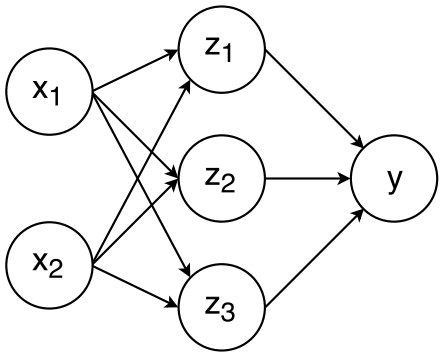


Figure 4.2: A simple MLP with a single hidden layer.

vious section, is to reduce the dimensionality of our representations, which in turn reduces the memory and computational requirements of subsequent algorithms. Hence, we shall rely on the loss associated with encoded (latent) embeddings (to be described in Section 4.4.4), in order to rank-order and select the best model from the inspected candidates.

In this section we will provide a brief overview of neural networks, and their potential in modeling complex non-linear relationships, followed by the mathematical model of VAEs and RBMs. We will then evaluate and compare the proposed models' effectiveness in finding low-dimensional fingerprints of IP addresses, as well as the utility of VAEs for visualizing collections of hosts.

4.4.1 Neural networks

Artificial neural networks [51] are constructed by combining multiple layers of units called neurons, or perceptrons. A perceptron is a simple function that takes a vector of values as input and returns a scalar output by applying a linear transformation followed by a non-linearity (i.e. activation function), e.g. the logistic function $(1+e^{-x})^{-1}$ for sigmoid neurons, or the rectifier function $\max(0, x)$ for the rectified linear unit (ReLU). A layer of perceptrons is then constructed by putting multiple neurons side-by-side for a vector of outputs. Such a layer can be formulated as $\boldsymbol{y} = f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$, where $\boldsymbol{x}$ and $\boldsymbol{y}$ are the input and output vectors, $\boldsymbol{W}$ and $\boldsymbol{b}$ are the weight matrix and the vector of biases, and $f(\cdot)$ is the non-linearity. A neural network is then constructed by combining multiple layers of neurons, using the output from one layer as input for the next one. Figure 4.2 illustrates a simple network, consisting of an input and output layer, and a single hidden layer. This structure, also called a multilayer perceptron (MLP), or a feed-forward neural network, can then be used

to define general non-linear functions, using parameters $\boldsymbol{\theta}$, where $\boldsymbol{\theta}$ is the weights and biases from all layers of the MLP. In fact, the *universal approximation theorem* [62] states that with a single hidden layer, sufficiently many hidden units, and a continuous, bounded, and non-constant activation function, MLPs can approximate functions on compact subsets of $\mathbb{R}^n$ with arbitrary precision. However, in practice deep models consisting of multiple hidden layers can approximate complex functions using far fewer units than one with only a single hidden layer.

Once the structure for a MLP is specified, one can *train* the weights and biases for a specific machine learning task using gradient descent. This is achieved by first defining a loss function $\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})$ over the desired output $\boldsymbol{y}$ and actual output $\hat{\boldsymbol{y}}$; typical loss functions include the mean squared error for real-valued outputs, and cross-entropy for binary outputs. The parameters of the MLP can then be updated using an estimate of the gradient $\nabla_{\boldsymbol{\theta}} \, \mathbb{E}_{\boldsymbol{x} \sim p_{data}}[\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})]$. In practice, neural networks are trained using variations of stochastic gradient descent (SGD), obtaining approximations to the aforementioned gradient by averaging the loss function over a random mini-batch of real-world observations.

The ability of neural networks in learning complex functions makes them adaptable to a wide variety of tasks. Note that while the universal approximation theorem guarantees the existence of a solution with arbitrary precision, it does not touch upon on the learnability of such a solution, i.e. the globally optimal point with respect to $\boldsymbol{\theta}$. In other words, the loss function is generally not convex, and therefore SGD is not guaranteed to find the global optimum. Nevertheless, when care is taken in designing the network, such as choosing the non-linearity, the cost function, and initialization of parameters, many deep models continue to outperform traditional machine learning algorithms.

4.4.2 Variational autoencoders

Autoencoders are a type of MLP network that can learn efficient representations (encodings) of data vectors, along with reconstructions (decodings) of the input data from those representations. Hence, an autoencoder is trained to minimize the objective function $\mathcal{L}(\boldsymbol{x}, \hat{\boldsymbol{x}})$, where $\boldsymbol{x}$ is the original data vector, $\hat{\boldsymbol{x}}$ is the reconstructed input, and $\mathcal{L}$ is a loss function for computing the reconstruction error, e.g. squared error for real-valued data, and cross-entropy for binary data.

A variety of methods have been developed to train deep autoencoders, for instance greedy layer-wise pre-training of restricted Boltzmann machines followed by a fine-tuning stage [61], and training denoising autoencoders by corrupting the input vectors, therefore forcing the network to learn more robust features [147]. Variational autoencoders combine

ideas from deep learning and statistical inference in order to train directed graphical models that can find latent representations of high-dimensional inputs, and generate synthetic samples that are similar (in distribution) to real-world examples.

Let $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{z} \in \mathbb{R}^m$ denote a vector of visible variables, and continuous hidden (latent) variables, respectively. Let $p_{\boldsymbol{\theta}}(\boldsymbol{z})$, $p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})$ be the prior for $\boldsymbol{z}$, and the conditional distribution of $\boldsymbol{x}$ given $\boldsymbol{z}$, where both are chosen from a parametric family of distributions, with $\boldsymbol{\theta}$ specifying the parameters. Therefore, $\boldsymbol{z}$ drives the generation of $\boldsymbol{x}$ through the conditional distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})$. Inference is performed by drawing from the posterior density $p_{\boldsymbol{\theta}}(\boldsymbol{z} \mid \boldsymbol{x})$, given by Bayes' rule:

$$p_{\boldsymbol{\theta}}(\boldsymbol{z} \mid \boldsymbol{x}) = \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z}) p_{\boldsymbol{\theta}}(\boldsymbol{z})}{p_{\boldsymbol{\theta}}(\boldsymbol{x})} = \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z}) p_{\boldsymbol{\theta}}(\boldsymbol{z})}{\int_{\boldsymbol{z}} p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z}) p_{\boldsymbol{\theta}}(\boldsymbol{z}) d\boldsymbol{z}} \,. \tag{4.1}$$

In general, the integral on the RHS of Equation 4.1 is intractable, and variational Bayesian methods [17] use an approximation of the true posterior given by $q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})$. The idea is to choose $q_{\boldsymbol{\phi}}(\cdot)$ from a family of distributions with a simpler form than the true posterior, selecting a distribution that minimizes the dissimilarity between the true and approximate posteriors. The most common form of variational Bayes uses the Kullback-Leibler (KL) divergence [80] of the true posterior from the approximate as a measure of dissimilarity, given by:

$$D_{KL}\Big[q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}) \,\|\, p_{\boldsymbol{\theta}}(\boldsymbol{z} \mid \boldsymbol{x})\Big] = \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z} \mid \boldsymbol{x})}\right] = \int_{\boldsymbol{z}} q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}) \log \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z} \mid \boldsymbol{x})} \,. \tag{4.2}$$

In the context of autoencoders, $q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})$ and $p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})$ denote the *recognition* model (encoder), and the *generative* model (decoder), where $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$ are their respective parameters, i.e. weights and biases in a MLP. The log-likelihood of the model can be written as follows:

$$\begin{aligned} \log p_{\boldsymbol{\theta}}(x) &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\ \log \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z} \mid \boldsymbol{x})}\ \right] + \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})}\right] \\ &= D_{KL}\Big[q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}) \,\|\, p_{\boldsymbol{\theta}}(\boldsymbol{z} \mid \boldsymbol{x})\Big] + \mathcal{L}(x) \,, \end{aligned} \tag{4.3}$$

where $\mathcal{L}(x)$ can be refactored as:

$$\begin{aligned} \mathcal{L}(\mathbf{x}) &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})\right] + \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})}\right] \\ &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})\right] - D_{KL}\Big[q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}) \,\|\, p_{\boldsymbol{\theta}}(\boldsymbol{z})\Big] \,. \end{aligned} \tag{4.4}$$

Note that since in Equation 4.3, $\log p_{\boldsymbol{\theta}}(x)$ is fixed with respect to $\boldsymbol{\phi}$, maximizing $\mathcal{L}(x)$ is

equivalent to minimizing the KL divergence term, which in turn minimizes the divergence of the approximate from the true posterior. The second term on the RHS of Equation 4.4 denotes the Kullback-Leibler (KL) divergence [80] of the approximate posterior $q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})$ from the prior $p_{\boldsymbol{\theta}}(\boldsymbol{z})$. The two terms in Equation 4.4 can be interpreted as the sum of a reconstruction loss and a penalty term. The first term tries to minimize the error given by the cross-entropy between $\boldsymbol{x}$ and its reconstruction, and the penalty term penalizes deviation from the prior defined on $\boldsymbol{z}$, most commonly the isotropic Gaussian distribution. 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, a problem suffered by autoencoders trained without any regularization.

Note that $\mathcal{L}(\boldsymbol{x})$ is commonly referred to as the variational lower bound, or the evidence lower bound (ELBO), and is a lower bound on the data log-likelihood according to the modeling distribution, derived from Jensen's inequality:

$$\mathcal{L}(\boldsymbol{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})}\right] \leq \log \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})}\right] = \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \,. \tag{4.5}$$

Furthermore, when drawing multiple samples from the approximate posterior, Burda et al. [20] propose importance weighing to obtain a tigher lower bound for $\log p_{\boldsymbol{\theta}}(\boldsymbol{x})$ as the objective function:

$$\mathcal{L}(\mathbf{x}) = \mathbb{E}_{\boldsymbol{z}^{(1)},\ldots,\boldsymbol{z}^{(k)} \sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{1}{k} \sum_{i=1}^{k} \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}^{(i)})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}^{(i)} \mid \boldsymbol{x})}\right] . \tag{4.6}$$

In order to train the model, one needs to differentiate and optimize the ELBO with respect to both $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$. However, obtaining the gradient with respect to $\boldsymbol{\phi}$ is non-trivial, and Kingma et al. [73] use a re-parameterization trick in order to get around this issue, by approximating the required expectations and their gradients. Assume that the prior $p_{\boldsymbol{\theta}}(\boldsymbol{z})$ follows a multivariate Gaussian distribution with zero mean and unit variance, in other words $p_{\boldsymbol{\theta}}(\boldsymbol{z}) \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and the approximate posterior also follows a factorial Gaussian distribution, i.e. $q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}) \sim \mathcal{N}\left(\mu_{\boldsymbol{\phi}}(x), \sigma^2_{\boldsymbol{\phi}}(\boldsymbol{x})\right)$, where $\mu_{\boldsymbol{\phi}}(\boldsymbol{x})$, and $\sigma^2_{\boldsymbol{\phi}}(\boldsymbol{x})$ are outputs from the recognition MLP. We can then draw from the approximate posterior by taking:

$$\boldsymbol{z} = \mu_{\boldsymbol{\phi}}(\boldsymbol{x}) + \sigma_{\boldsymbol{\phi}}(\boldsymbol{x}) \odot \boldsymbol{\epsilon} \,, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \,. \tag{4.7}$$

This re-parameterization transfers the randomness in $\boldsymbol{z}$ to $\boldsymbol{\epsilon}$ and allows one to back-propagate derivatives through the network for training the model. Finally, for binary $\boldsymbol{x}$, the

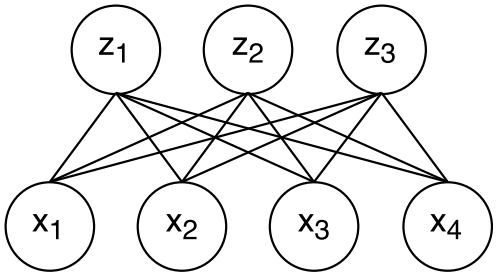


Figure 4.3: Graphical depiction of a restricted Boltzmann machine's Markov random field. The bottom and top rows correspond to visible and hidden variable, respectively, and edges indicate direct interaction between two variables. Note that visible (hidden) states are independent from each other, given the hidden (visible) vector.

reconstruction $p_\theta(\boldsymbol{x} \mid \boldsymbol{z})$ can be defined to follow the Bernoulli distribution, with parameters taken from the output of the generative network.

4.4.3 Restricted Boltzmann machines

Restricted Boltzmann machines are another generative stochastic neural network model, that learn the probability distribution of inputs through a set of binary latent variables. Similar to Section 4.4.2, let $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{z} \in \mathbb{R}^m$ denote a vector of visible, and hidden (latent) variables, respectively. A RBM is an energy-based model, meaning the underlying probability distribution can be expressed as:

$$p(\boldsymbol{x}, \boldsymbol{z}) = \frac{e^{-E(\boldsymbol{x},\boldsymbol{z})}}{Z} \quad \text{with} \quad Z := \sum_{\boldsymbol{x},\boldsymbol{z}} e^{-E(\boldsymbol{x},\boldsymbol{z})} \,, \tag{4.8}$$

where $p(\boldsymbol{x}, \boldsymbol{z})$ is the joint probability mass function corresponding to the RBM, and $E(\boldsymbol{x}, \boldsymbol{z})$ is the *energy* of the configuration $(\boldsymbol{x}, \boldsymbol{z})$. Z is also referred to as the *partition function*, ensuring that $\sum_{\boldsymbol{x},\boldsymbol{z}} p(\boldsymbol{x}, \boldsymbol{z}) = 1$. Note that the partition function of a RBM is intractable, therefore in contrast to a VAE, it is not possible to directly estimate $p(\boldsymbol{x}, \boldsymbol{z})$, as well as its marginal density $p(\boldsymbol{x})$.

Figure 4.3 illustrates the undirected graphical model, or Marokov random field [71], corresponding to a RBM. Each vertex in this graph corresponds to a variable (visible or hidden), with edges indicating direct interaction between two vertices. Thus, the energy function can be formulated as:

$$E(\boldsymbol{x}, \boldsymbol{z}) = -\boldsymbol{a}^T \boldsymbol{x} - \boldsymbol{b}^T \boldsymbol{z} - \boldsymbol{x}^T \boldsymbol{W} \boldsymbol{z} \,, \tag{4.9}$$

where $\boldsymbol{W} \in \mathbb{R}^{n \times m}$, $\boldsymbol{a} \in \mathbb{R}^n$, and $\boldsymbol{b} \in \mathbb{R}^m$ indicate the weights and biases of the RBM. Note that since there are no direct connections between visible (hidden) units in Figure 4.3, each unit is independent from other variables given the hidden (visible) vector; in other words:

$$p(\boldsymbol{z} \mid \boldsymbol{x}) = \sigma(\boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{b}) \,, \tag{4.10}$$

$$p(\boldsymbol{x} \mid \boldsymbol{z}) = \sigma(\boldsymbol{W} \boldsymbol{z} + \boldsymbol{a}) \,, \tag{4.11}$$

where $\sigma(\cdot)$ denotes the logistic function $(1 + e^{-x})^{-1}$. The simple form of Equations 4.10 and 4.11 makes it extremely efficient to sample from these two conditional densities, which is then leveraged for training a RBM. The marginal distribution $p(\boldsymbol{x})$ can also be obtained by summing over the latent space, i.e. over all possible $\boldsymbol{z}$, and can be rewritten as:

$$p(\boldsymbol{x}) = \frac{e^{-\mathcal{F}(\boldsymbol{x})}}{Z} \quad \text{with} \quad \mathcal{F}(\boldsymbol{x}) := -\log \sum_{\boldsymbol{z}} e^{-E(\boldsymbol{x}, \boldsymbol{z})} \,, \tag{4.12}$$

where $\mathcal{F}(\boldsymbol{x})$ is called the *free energy*; note that Z can also be expressed as a function of the free energy:

$$Z = \sum_{\boldsymbol{x}} e^{-\mathcal{F}(\boldsymbol{x})} \,.$$

Combining Equations 4.9 and 4.12, it is shown that the free energy of a RBM reduces to the following closed form solution:

$$\mathcal{F}(\boldsymbol{x}) = -\boldsymbol{a}^T \boldsymbol{x} - \sum_j \log\left(\boldsymbol{1} + \exp\left(\boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{b}\right)\right)_j \tag{4.13}$$

For training a RBM, one is required to obtain an estimate of the gradient of the log-likelihood $\log p(\boldsymbol{x})$ with respect to parameters $\boldsymbol{\theta} := (\boldsymbol{W}, \boldsymbol{a}, \boldsymbol{b})$, given by:

$$\begin{aligned} -\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{x}) &= \nabla_{\boldsymbol{\theta}} \mathcal{F}(\boldsymbol{x}) + \nabla_{\boldsymbol{\theta}} \log Z \\ &= \nabla_{\boldsymbol{\theta}} \mathcal{F}(\boldsymbol{x}) - \mathbb{E}_{\boldsymbol{x} \sim p(\boldsymbol{x})} \nabla_{\boldsymbol{\theta}} \mathcal{F}(\boldsymbol{x}), \end{aligned} \tag{4.14}$$

where the expectation is taken over the model's probability mass function $p(\boldsymbol{x})$. In order to estimate the second term in the RHS of Equation 4.14, one can sample from the model distribution using Gibbs sampling. Naively, this can be achieved by selecting a random visible vector $\boldsymbol{x}_0$, and alternating between Equations 4.10 and 4.11 in order to obtain the Markov chain $\boldsymbol{x}_i$. For large k, the vector $\boldsymbol{x}_k$ can then be taken as a sample from the model distribution. However, this technique results in slow estimators due to the large number of steps required for *burning in* the Markov chain.

The contrastive divergence (CD) [59] algorithm alleviates the above issue by initializing the chain using samples from a probability distribution that is fairly close to the model distribution, i.e the data distribution. In other words, the same mini-batch of real-world observations used for SGD, are also leveraged for initialing a set of Markov chains and obtaining samples from the model distribution $p(\boldsymbol{x})$; Equations 4.13 and 4.14 can then be utilized for updating the parameters $\boldsymbol{\theta}$. This technique is refereed to as the CD-k algorithm, with k denoting the number of Gibbs sampling rounds used for drawing from the model distribution. In practice, CD-1 has been shown to work surprisingly well [60], especially when the main objective is not to learn a good density model, but to construct embeddings of visible vectors, as is the case with the current study.

4.4.4 Evaluation

Variational autoencoders For our evaluation of VAEs, we will compare two different structures: (1) a VAE with simple linear transformations (i.e. no hidden layers) for the recognition and inference networks, and (2) a VAE with two hidden layers consisting of 1000 units each for both networks. We shall then compare models with linear and deep structures, to denote how models of IP addresses 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 [123] to all layers excluding the output layer, and use Equations 4.4 and 4.6 with 10 draws from the approximate posterior for training and evaluation, respectively. For training we use adaptive moment estimation (Adam) [72], an optimization technique that keeps separate adaptive learning rates for each parameter, and allows the model to converge faster with sparse input. We use a learning rate of 0.0005, mini-batch sizes of 100, and train each model for 10 epochs.

Restricted Boltzmann machines For training RBMs we use CD-1, and train our models for 10 epochs, using the Adam optimizer with a learning rate of 0.001, and mini-batch sizes of 100. Note that when performing a bottom-up pass using Equation 4.10, or a top-down pass (Equation 4.11) with reconstructions as inputs, one can use the real-valued probabilities themselves, or stochastically pick a zero or one prior to plugging values into the corresponding equation. Upon the recommendation of [60], we sample values for a bottom-up pass, while using probabilities for Equation 4.11 in order to reduce sampling noise. The former acts as a strong regularizer, by ensuring that a hidden unit can convey at most one bit, and not a real value, to visible units. Note that when using the last update of hidden

units for measuring the free energy, we also use probabilities to avoid sampling noise.

Comparison We train multiple models for different numbers of latent variables, using a training set of 10 million samples from Censys. We then evaluate the trained models using a separate data set containing one million records. Tables 4.2 and 4.3 summarize the performance of the resulting models. For VAEs, 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 can see that deep models do not benefit from increasing the number of latent dimensions beyond 50; interestingly, their performance slightly decreases for 100 latent variables, possibly because the model is converging to a bad local optimum. Note that this value is missing from Table 4.3, due to the intractable nature of the partition function in Equation 4.8.

To further inspect the performance of these models as autoencoders, we examine the reconstruction error over our testing data set. To obtain reconstructions for VAEs, we first run these samples through the encoder to find latent representations, and then use those representations to draw from the decoder. Note that the output of the decoder is not a binary vector, but a vector of probabilities representing the probability that each bit should be set to one. Furthermore, since decoding relies on the stochastic variable $\boldsymbol{\epsilon}$, 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. For RBMs we obtain reconstructions by conducting one Gibbs sampling round, i.e. the same process used for CD-1.

From Table 4.2b, we can see a clear advantage for using deep networks, especially for small numbers of latent variables. However, even with 200-dimensional embeddings, we see that a linear VAE exhibits 20.0% less reconstruction error, i.e mean error in bits, than a deep VAE with 50 latent variables. Note that principal component analysis (PCA) is a more common technique for dimensionality reduction with linear transformations; however, PCA assumes that $\boldsymbol{x}$ consists of real-valued variables, and therefore it is not an appropriate model for our binary representations. The models in Table 4.2a are more similar to a generalization of PCA for exponential families of probability distributions [28], which can be adapted to binary data, i.e. logistic PCA. However, VAEs also makes explicit assumptions on the prior for latent variables, and are trained using a variational framework.

Comparing Tables 4.2 and 4.3, we also observe that VAE embeddings exhibit higher precision, especially for smaller number of latent variables. Note that since RBMs do not leverage deep networks, it is more appropriate to compare their performance to the linear models in Table 4.2a; nevertheless, our results suggest that even linear VAEs outperform

# of latent features	ELBO	Mean error (bits)	Percentage error				
			Mean	Percentiles			
				50	75	90	95
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
100	-58.1	4.7	6.5	4.2	9.1	15.8	21.2
200	-56.4	3.6	5.6	3.4	7.8	13.7	18.8

(a) Linear recognition/generative networks

# of latent features	ELBO	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
100	-32.2	3.3	3.4	1.4	4.9	9.3	13.3

(b) Deep recognition/generative networks

Table 4.2: Negative log-likelihood (in nats), and reconstruction error for VAE models with linear (top) and MLP (bottom) 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.

# of latent features	Mean error (bits)	Percentage error				
		Mean	Percentiles			
			50	75	90	95
10	61.8	55.6	53.3	73.9	92.9	109.4
20	48.9	38.1	33.3	52.8	74.1	86.7
50	24.5	17.8	12.7	25.6	40.5	50.7
100	12.7	7.9	4.3	11.1	21.2	29.1
200	5.4	2.6	0.0	3.2	8.1	12.6

Table 4.3: Reconstruction error for RBM models.

RBMs in terms of reconstruction loss. This is to be expected, as real-valued latent variables have the potential to convey much more information than binary states. This property, in addition to the compatibility of VAEs with deep structures, results in models that reach the same fidelity as RBMs with 4-10 times less latent dimensions. Moreover, since latent embeddings produced by both types of models loose the sparsity of visible vectors, i.e. our binary representations detailed in Section 4.3, they can no longer be stored efficiently using sparse matrix formats. As an example, 200-dimensional embeddings occupy roughly 60% more memory than our sparse representations.[2] Hence, motivated by our results in this section, for the case studies presented in Section 4.5 we will only evaluate latent fingerprints produced by models in Table 4.2b.

For our implementation of a deep VAE in TensorFlow [139], using a Nvidia GeForce GTX-1080-Ti GPU, each training epoch on 10 million samples takes roughly one hour to complete. For a trained model, it takes 40 microseconds (per sample) to transform a record (find its latent representation), and 190μs (220μs) to reconstruct a sample with a single draw (10 draws) from the decoder. These numbers are computed for our most accurate VAE model with 50 latent variables, and averaged over batches consisting of 100 samples; we observe similar times for other numbers of latent variables.

4.4.5 Visualization

When mapping binary fingerprints to two-dimensional latent embeddings, we can visualize collections of hosts by drawing scatter plots, or an estimate of their density, over the encoded samples. Figure 4.4 illustrates the overall distribution of latent variables for our 2-D VAE model. This figure is generated over a random selection of 10 000 IP addresses, and drawing the kernel density estimate (KDE) of the resulting samples. As expected, we observe that the latent representations approximately follow a Gaussian distribution, i.e. we have mapped sparse binary vectors to well-behaved decoupled variables. Note that due to the binary nature of RBM embeddings, they cannot produce efficient representations in low-dimensional spaces, as is evident from our results in Table 4.3.

One of the objectives of VAEs, and autoencoders in general, is to also provide a measure of similarity between points, quantified by the distance between samples in the latent space. For instance, when training 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. To examine whether this is also

[2]Note that this number is unique to the present problem, as a different data set can result in binary fingerprints with a different sparsity structure.

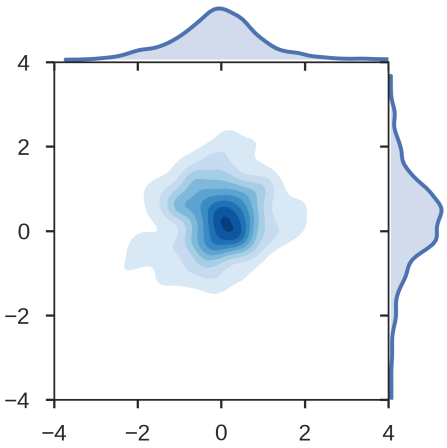


Figure 4.4: Bivariate and marginal distributions for two-dimensional VAE representations of Internet hosts, resembling an isotropic Gaussian distribution.

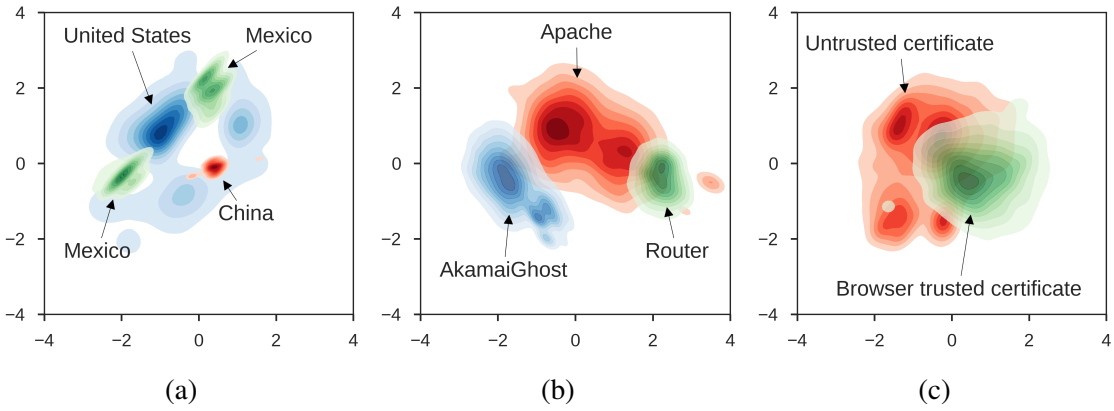


Figure 4.5: 2-D representations of IPs from different countries (left), different HTTP servers (center), and correctly configured and misconfigured HTTPS servers (right). Figure 4.5a uses our 2-D representations, and for Figures 4.5b and 4.5c we use PCA projections from 50-dimensional fingerprints.

the case for IP addresses, we look at two-dimensional representations of different classes of IPs. For this exercise we choose location, more specifically the country the IP address belongs to, different types of HTTP servers, and also whether the host serves a browser trusted or untrusted (such as self-signed) SSL certificate over the HTTPS protocol.

Figure 4.5a displays our results for three different countries, namely United States, China, and Mexico, by illustrating the bivariate kernel density estimate of samples' 2-D representations. Note that not all samples fall within the drawn contour lines; however, the plots show where most samples are concentrated. As expected, we can see that similar IP addresses in terms of originating country fall within close proximity of each other. We

further observe that IPs from China form a small, concentrated cluster. This is possibly due to the fact that these hosts exhibit low variability in their measurements, hence the training algorithm deems it sufficient to only allocate a small range of points for these samples.

Figure 4.5a illustrates that when examining a subsection of the global population, e.g. IPs from different countries, or different types of servers, points generally become concentrated into one or multiple clusters. In this setting, for further examination of points within the aforementioned sub-populations, it makes sense to find a transformation that can re-normalize data points for creating interpretive plots. For web servers and SSL certificates, note that we are essentially examining a smaller subdivision of hosts, i.e. those that respond to HTTP and HTTPS requests, respectively. Since we are working with real-valued and well-behaved numerical embeddings, our experiments suggested that training fast linear transformations using PCA is sufficient for this purpose. Furthermore, since PCA can also be used for dimensionality reduction, we can train our normalizers to project our higher dimensional embeddings onto two dimensions, thus leveraging the precision offered by high dimensionality for characterizing sub-populations. Note that this technique presents another advantage of VAE models over RBMs, as PCA is not an appropriate choice for projecting binary RBM embeddings.

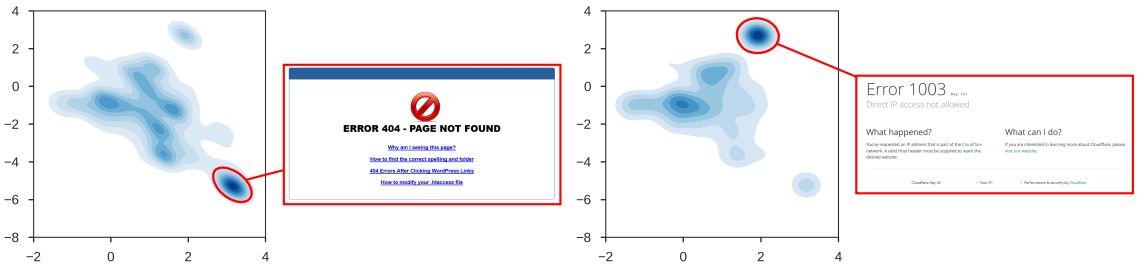
Following this technique we have illustrated different categories of (secure) web servers in Figures 4.5b and 4.5c, using projections from 50-dimensional latent fingerprints. While the depicted clusters do slightly overlap, they show a clear difference in distributions of the encoded representations, motivating our claim that these are meaningful fingerprints.

4.5 Applications

In this section we explore applications of our numerical representations of IP addresses for three concrete case studies, namely for detecting (potentially) malicious servers, inferring hidden attributes of a host, and characterizing networks. Unless otherwise stated, all of our results on latent fingerprints have been obtained using VAE embeddings corresponding to deep models from Table 4.2b.

4.5.1 Quantifying maliciousness

IP addresses that are known for engaging in malicious activities, such as phishing or spamming, can be reported through various independent blacklists. In this section, we utilize reported malicious websites in order to assign reputations to arbitrary hosts, by training a series of supervised models, or classifiers, and evaluating their performance. We use



(a) PhishTank (b) hpHosts

Figure 4.6: Density for two-dimensional representations of blacklisted IP addresses from PhishTank (left), and hpHosts (right). Hosts from the highlighted clusters correspond to the depicted landing pages, taken from HTTP content recorded by Censys.

two blacklists, namely PhishTank [112] and hpHosts [63], to construct labeled samples for training/evaluation. Note that PhishTank is a blacklist that focuses mainly on identifying phishing websites, while hpHosts also targets ad/tracking and malware websites.

For each of the Censys snapshots included in our study, we take hosts that have been blacklisted after the corresponding date, but before the next series of measurements, as the set of known malicious IPs for that period. For instance, the set of blacklisted IPs for July 1-15 constitute labels for the snapshot collected on July 1. For PhishTank and hpHosts, this results in 43 641 and 401 264 labeled samples, 94.4% and 91.8% of which have corresponding measurements in Censys, respectively. This mismatch is possibly due to the time discrepancy between labels and network measurements for hosts with dynamic IP addresses, or because these hosts do not respond to any of the scanners' probes.[3]

4.5.1.1 Visualizing malicious hosts

We first illustrate the two-dimensional representation of these IP addresses in Figure 4.6, where we can identify multiple clusters of hosts. It is possible that each is indicative of a specific type of host, due to similarities between how IP addresses hosting these pages are configured. We thus inspected the recorded HTTP content for 100 randomly picked hosts from each cluster, i.e. samples within the highlighted contour lines, and found that all of them correspond to the depicted pages. Both pages are default ISP landing pages, namely for HostGator (Figure 4.6a) and Cloudflare (Figure 4.6b). Interestingly, for the inspected samples from Figure 4.6a we also found a few pages following the same theme but in Portuguese, and for Figure 4.6a we came across landing pages for Baidu, Cloudflare's

[3]These web servers must at least serve content on either HTTP or HTTPS. However, servers may block requests that do not include the URL of the requested page, as is the case with Censys' network probes.

partner in China. It is worth noting that none of our features are extracted from the contents of a website, yet the resulting clustering clearly indicates concentration of contents.

Note that for these samples we are not seeing the actual phishing content, possibly because the malicious content is hidden behind a URL not known by Censys scanners. Nevertheless, the concentration of blacklisted IPs around these points suggests that they correspond to a higher likelihood of serving malicious content, either deliberately or due to infection. By training an estimator that can recognize such relationships, and leverage them in order to distinguish between malicious and benign samples, we can then use the trained model to assign reputations to arbitrary hosts.

4.5.1.2 Evaluation of supervised techniques

In addition to the selected blacklisted IPs, we randomly pick 800 000 samples (200 000 from each snapshot) from Censys that have not been blacklisted, and append them to our data sets as a set of benign samples. We shall then train supervised estimators with the objective of discerning between (potentially) malicious and benign hosts. Note that this experiment enables our models to perform both detection and forecasting, depending on whether a host is already serving malicious content when it is probed, or if it exhibits high risk of turning malicious in the two week period following each snapshot. Thus, we hold-out samples from August 16-31 for estimating the predictive performance of our models, and use samples between July 1 and August 15 for training/testing on a 50/50 split. We evaluate several supervised algorithms, summarized below; for each algorithm we train three classifiers on binary, and 10-D and 50-D latent embeddings:

- *Random forest*: Random forests [128] are an ensemble of decision trees, that reduce over-fitting by averaging over multiple estimators.
- *Gradient-boosted trees*: We also use XGBoost [25], a recent supervised model based on decision trees, that uses gradient-boosting to improve upon individual trees.
- *Multilayer perceptron*: In addition to tree-based models, we train fully-connected MLPs with two hidden layers. We regularize our MLPs using batch normalization [65], and dropout [135].

After tuning the hyper-parameters of the above models, we found that XGBoost consistently outperformed other algorithms, and therefore our results are reported using gradient-boosted trees. We train estimators using 100 trees, a learning rate of 0.1, and a maximum depth of 20 for each individual tree. We further regularize our 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.

As demonstrated in the previous section, the distance between two hosts in the latent space can be used as a similarity metric. Furthermore, the low dimensionality of these fingerprints allows us to utilize the k-nearest neighbors algorithm (k-NN) for classification. Compared to tree-based models and neural networks, k-NN is better suited for inspecting predictions, since by associating samples with multiple similar and labeled hosts, scores can be accompanied with real-world examples that are known for serving malicious content, such as the examples provided in Figure 4.6. Therefore, we are also including results for k-NN models with $k = 50$, trained over 10-dimensional embeddings; using higher dimensions results in very slow models due to inefficiency of k-NN for high-dimensional vectors. This experiment also provides further validation for the utility of latent fingerprints for fuzzy-matching, i.e. measuring the similarity, between Internet hosts.

4.5.1.3 Discussion of results

The performance of the resulting classifiers is shown in Table 4.4. The output of each model for an arbitrary sample is a number between zero and one, quantifying the likelihood that the underlying host is malicious, or will turn malicious in the near future. One can then discretize a model's output by comparing it against a constant threshold, and evaluate the accuracy of estimations over a held-out test set. We have included true positive rates (TPR), and false positive rates (FPR) for different operating points (corresponding to different thresholds), the area under the curve (AUC) score, and the speed (time to evaluate a single sample) of each model in Table 4.4. The latter is computed using one thread on an Intel Core i7-7770K processor. For each statistic, we are reporting its mean and standard deviation over five different runs, since each run can result in a slightly different model due to inherent randomness in training procedures, and the choice of train/test data sets. Note that the AUC score measures the overall classification accuracy by measuring how accurately a classifier can rank-order samples, i.e. the probability that a randomly drawn bad (malicious) sample is scored higher than a randomly drawn benign one.

Moreover, for assessing the predictive accuracy of our technique, we evaluate our models over blacklisted hosts from August 16-31, that have not been blacklisted in any of the preceding dates. For PhishTank and hpHosts, this results in 893 and 6860 previously unobserved IP addresses, respectively. For these examples, the measurements collected by Censys correspond to hosts that have not turned malicious, or have not yet been detected by the inspected blacklist. Hence, they allow us to assess how our technique can forecast maliciousness, and generalize to previously unobserved samples. We have included

Features		Binary	Latent (10-D)		Latent (50-D)
	TPR	XGBoost	k-NN	XGBoost	XGBoost
FPR (%)	50%	0.3 ± 0.0	0.4 ± 0.0	0.4 ± 0.0	0.3 ± 0.0
	80%	2.2 ± 0.1	3.0 ± 0.1	3.1 ± 0.1	2.7 ± 0.1
	90%	6.0 ± 0.3	7.7 ± 0.1	7.1 ± 0.2	6.3 ± 0.2
	95%	10.7 ± 0.4	13.7 ± 0.4	12.1 ± 0.5	10.9 ± 0.4
AUC (%)	Overall	97.9 ± 0.1	96.1 ± 0.1	97.6 ± 0.1	97.9 ± 0.1
	Prediction	97.1 ± 0.1	94.9 ± 0.2	96.7 ± 0.1	97.0 ± 0.1
Time/sample		$\sim 0.25ms$	$\sim 0.7ms$	$\sim 15\mu s$	$\sim 15\mu s$

(a) PhishTank

Features		Binary	Latent (10-D)		Latent (50-D)
	TPR	XGBoost	k-NN	XGBoost	XGBoost
FPR (%)	50%	0.8 ± 0.0	1.0 ± 0.0	0.8 ± 0.0	0.7 ± 0.0
	80%	3.5 ± 0.0	4.8 ± 0.1	4.6 ± 0.0	4.0 ± 0.0
	90%	7.4 ± 0.1	9.2 ± 0.1	9.3 ± 0.1	8.6 ± 0.1
	95%	12.2 ± 0.1	14.3 ± 0.1	14.5 ± 0.1	13.7 ± 0.1
AUC (%)	Overall	97.4 ± 0.0	96.3 ± 0.0	96.9 ± 0.0	97.2 ± 0.0
	Prediction	95.8 ± 0.0	93.6 ± 0.1	94.9 ± 0.1	95.3 ± 0.1
Time/sample		$\sim 0.35ms$	$\sim 0.7ms$	$\sim 20\mu s$	$\sim 20\mu s$

(b) hpHosts

Table 4.4: Performance/speed of classifiers trained using PhishTank (top), and hpHosts (bottom). When using latent fingerprints, k-NN models use 10-dimensional VAE embeddings, while XGBoost models are trained on both 10-D and 50-D representations. AUC scores are reported over all test samples, and previously unblacklisted IPs from 08/16/2016 to 08/31/2017.

the AUC of our classifiers over these samples in Table 4.4. Ideally, one would want this measure to be equal to the overall AUC, however in practice we observe slightly lower values. Nevertheless, our results suggest that we can also forecast malicious websites with plausible accuracy, thus resulting in reputations that are reliable and robust.

Our results in Table 4.4 show that the examined tree-based models perform roughly the same in terms of overall and predictive performance, with models trained on binary representations outperforming other classifiers, possibly due to the lossy nature of our latent embeddings. We also observe that the higher fidelity of 50-dimensional embeddings results in slightly more accurate estimates compared to those obtained from 10-dimensional representations. Another distinction between the models in Figure 4.4 is their speed. We observe that latent embeddings result in estimators that are more than 10 times faster than other models, though they require a GPU and more pre-processing for computing the latent fingerprints. However, this distinction is only relevant in a large production/research environment serving many clients, where latent fingerprints can be computed offline and then shared across multiple applications. Nevertheless, all depicted models are capable of processing thousands of hosts per second, enabling large-scale analysis of IP addresses for fast identification of suspicious hosts.

Finally, we see that k-NN models are slower, and slightly less performing than tree-based algorithms. However, as we mentioned earlier in the section, they can produce more interpretive predictions, and can be used in conjunction with gradient-boosted trees, for inspecting individual predictions.

Our results in this section demonstrate how to utilize our fingerprints in a supervised learning setting. While we have only evaluated our fingerprints for a specific application, the same approach can be similarly applied to other domains. For instance, the ability to detect unsecured or infected devices is another interesting application of the presented work as, e.g., infected IoT devices can form botnets, such as the Mirai botnet [11], for launching distributed denial-of-service (DDoS) attacks. It is also worth noting that none of our features are extracted from the contents of a website. Content-based methods [3, 134, 152], have been extensively studied for detecting phishing or malicious websites, but are often slower than the techniques presented here. This naturally suggests an efficient divide-and-concur method, whereby highly suspicious samples are first identified without using the content itself, and are subsequently examined by more intensive content processing. It would also be interesting to see whether our results can boost the performance of content-based algorithms, or other approaches such as techniques for detecting malicious URL redirections [85, 91, 132].

4.5.2 Inferring masked attributes

Another interesting application of our fingerprints, is their utility in inferring missing or masked attributes by using cross-correlation and higher order interactions between features of an IP address. For instance, assume that a web server is masking the software used for serving content, which can 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, specially zero-day exploits, to compromise the host. However, it may still be possible to infer the installed application from other attributes that have not been masked, e.g. servers in a certain region might be more inclined to use Apache, or Microsoft products.

In order to test this hypothesis, we can manually mask reported attributes, e.g. headers/banners, of hosts from available measurements, and train a supervised model for inferring them back from the rest of our features. A drawback of this technique is that it does not take into consideration behavioral differences in privacy-seeking and other types of administrators. For example, a security expert who is more inclined to hide various headers/banners, might also have different preferences on which software they deploy on their respective servers. Nevertheless, the proposed technique can measure how these maskable attributes can be inferred back from other visible features, on hosts for which this information is currently being exposed.

For this experiment, we manually inspect tokens that were extracted from the Server header during our feature extraction process, and recognize 23 types of web servers. These categories will 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 already reporting the product deployed on their respective machines. We select 200 000 hosts with non-empty Server headers, label and then remove the examined header from all of our samples, and compute binary and latent fingerprints 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 fingerprints, we also remove all metadata extracted from the masked header.

The above technique imitates a rather naive way of masking attributes. In practice, security-conscious administrators can potentially hide headers and banners of other ports and protocols on their respective machines, in order to make the task of inference more difficult. For instance, the use of Microsoft products on other protocols may also suggest their deployment on the inspected one. Hence, we also repeat the proposed technique by mask-

ing all headers/banners (as well as metadata), to obtain a series of examples corresponding to a more meticulous masking strategy. Finally, we train supervised classifiers for inferring the aforementioned categories of web servers, using an even split for training/testing. To make our estimators robust to both types of information masking, we train them on both sets of samples, and report their performance on each set, separately.

Table 4.5 displays our results. We have included the overall accuracy of trained classifiers, as well as their performance over the four most frequent server products in our data set (excluding *AkamaiGhost*[4]), namely *Apache*, *Microsoft*, *NGINX*, and *Lighttpd*. Similar to the previous section, we train models on binary fingerprints, as well as 10-dimensional and 50-dimensional latent representations. The hyper-parameters of the inspected models are also the same as those used in Section 4.5.1, though we found that using trees with a maximum depth of 10 was sufficient for this experiment. Since we are performing multi-class classification, the output of the presented models for an arbitrary sample i ($1 \leq i \leq n$) is a set of probabilities, summing up to one ($p_{i,j} : \sum_{j=1}^{k} p_{i,j} = 1$), indicating the likelihoods corresponding to different categories, i.e. web server products. Take $y_i \in \{1 \dots k\}$ to be the ground-truth labels, and $\hat{y}_i := \arg\max_j p_{i,j}$ to be the predicted label, i.e. the most probable class, for sample i. Then the true positive rate (TPR), false positive rate (FPR), and precision (PRC) over the k^{th} class are defined as follows:

$$\begin{cases} \text{TPR}(k) &= \dfrac{|\{i :\ y_i = k,\ \hat{y}_i = k\}|}{|\{i : y_i = k\}|} \\ \text{FPR}(k) &= \dfrac{|\{i :\ y_i \neq k,\ \hat{y}_i = k\}|}{|\{i : y_i \neq k\}|} \\ \text{PRC}(k) &= \dfrac{|\{i :\ y_i = k,\ \hat{y}_i = k\}|}{|\{i : \hat{y}_i = k\}|} \end{cases}, \tag{4.15}$$

where $|\cdot|$ denotes the cardinality of a set. From Table 4.5a it is clear that we can infer the hidden information with both high precision and recall (true positive rate), even for sparse categories (e.g. Lighttpd), with a maximum overall accuracy of 97.9% when naive masking is employed. We observe a lower accuracy of 80.9% for more throughly masked fingerprints. Note that the reported precisions indicate the reliability of classifiers' outputs, i.e. the percentage of *detected* hosts that actually correspond to the inspected category. For instance, using binary fingerprints in Table 4.5a (Table 4.5b), we can detect 88.1% (60.8%) of Lighttpd servers with 95.7%, (89.8%) precision; although we are recalling labels at a lower

[4]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 set. Since these are a very specific type of host, they result in true positive rates and precisions higher than 99% for all models in Tables 4.5a and 4.5b.

		Binary	Latent (10-D)		Latent (50-D)
		XGBoost	k-NN	XGBoost	XGBoost
Apache (30.1%)	TPR	97.9 ± 0.1	87.9 ± 0.3	91.1 ± 0.2	95.9 ± 0.1
	FPR	1.5 ± 0.0	9.2 ± 0.2	10.9 ± 0.2	4.5 ± 0.1
	PRC	96.5 ± 0.1	80.5 ± 0.3	78.4 ± 0.3	90.2 ± 0.2
Microsoft (14.6%)	TPR	96.9 ± 0.2	89.7 ± 0.3	90.0 ± 0.2	93.6 ± 0.3
	FPR	0.4 ± 0.0	2.2 ± 0.1	1.7 ± 0.0	1.0 ± 0.0
	PRC	97.9 ± 0.1	87.5 ± 0.3	90.2 ± 0.2	94.1 ± 0.1
NGINX (14.5%)	TPR	96.9 ± 0.1	66.9 ± 0.6	65.2 ± 0.8	87.4 ± 0.2
	FPR	0.6 ± 0.0	3.0 ± 0.1	2.4 ± 0.1	1.2 ± 0.0
	PRC	96.7 ± 0.1	79.1 ± 0.4	82.3 ± 0.4	92.4 ± 0.2
Lighttpd (2.3%)	TPR	88.1 ± 0.5	72.5 ± 0.8	71.2 ± 1.0	81.4 ± 0.9
	FPR	0.1 ± 0.0	0.4 ± 0.0	0.3 ± 0.0	0.2 ± 0.0
	PRC	95.7 ± 0.4	82.2 ± 0.2	86.1 ± 0.8	92.5 ± 1.2
Accuracy		97.9 ± 0.0	86.3 ± 0.1	87.3 ± 0.1	94.4 ± 0.1
Time/sample		$\sim 0.5ms$	$\sim 0.35ms$	$\sim 0.25ms$	$\sim 0.25ms$

(a) Masked HTTP Server header

		Binary	Latent (10-D)		Latent (50-D)
		XGBoost	k-NN	XGBoost	XGBoost
Apache (30.1%)	TPR	86.3 ± 0.4	80.8 ± 0.4	80.3 ± 0.3	83.9 ± 0.4
	FPR	12.0 ± 0.4	12.3 ± 0.4	12.9 ± 0.2	11.1 ± 0.2
	PRC	75.7 ± 0.6	74.0 ± 0.5	72.9 ± 0.4	76.6 ± 0.4
Microsoft (14.6%)	TPR	77.6 ± 0.5	73.6 ± 0.9	76.6 ± 0.4	78.0 ± 0.6
	FPR	4.5 ± 0.1	4.5 ± 0.1	5.1 ± 0.1	4.7 ± 0.1
	PRC	74.6 ± 0.5	73.7 ± 0.4	72.0 ± 0.4	74.1 ± 0.2
NGINX (14.5%)	TPR	68.0 ± 1.0	59.4 ± 0.8	58.9 ± 0.7	65.2 ± 0.5
	FPR	2.6 ± 0.1	4.4 ± 0.1	4.5 ± 0.2	3.5 ± 0.1
	PRC	81.7 ± 0.6	69.4 ± 0.5	69.2 ± 0.6	76.1 ± 0.6
Lighttpd (2.3%)	TPR	60.8 ± 0.9	59.4 ± 1.1	53.2 ± 0.5	61.7 ± 0.5
	FPR	0.2 ± 0.0	0.7 ± 0.0	0.4 ± 0.0	0.3 ± 0.0
	PRC	89.8 ± 0.6	67.6 ± 1.5	73.8 ± 1.0	81.1 ± 1.4
Accuracy		80.9 ± 0.1	77.0 ± 0.1	77.0 ± 0.1	79.8 ± 0.1
Time/sample		$\sim 0.45ms$	$\sim 0.3ms$	$\sim 0.25ms$	$\sim 0.25ms$

(b) Masking all headers/banners

Table 4.5: True positive rate (TPR), false positive rate (FPR), and precision (PRC) for detection of different web (HTTP) server products. The reported results correspond to predictions of a single model over samples where only the *Server* header is being masked (top), or a more thorough masking of headers/banners from all protocols (bottom).

Field name	Contribution
Ownership (AS) information	24.1%
Geolocation	17.8%
Port 443 (HTTPS)	20.5%
Port 80 (HTTP)	20.3%
HTTP headers	12.6%
Port 22 (SSH)	4.1%
Port 25 (SMTP)	2.1%
Port 23 (Telnet)	1.6%
Other ports	3.7%

Table 4.6: Break-down of contribution from different fields for detection of different web (HTTP) server products. Contribution of individual features extracted from a field (e.g. all features extracted from ownership information) are aggregated to obtain the cumulative importance of said field.

rate, the false positive rate remains low enough to retain the precision of our predictions, thus resulting in models that are robust to different levels of information masking.

Similar to our results in Section 4.5.1, classifiers using latent variables are consistently less performing than those using binary fingerprints. However, in contrast to the previous section, we do not observe a significant advantage in term of speed for estimators trained on latent embeddings. Additionally, we notice a bigger gap between the accuracy of k-NN and tree-based models. Nevertheless, the accuracy of k-NN models (86.3% and 77.0%) suggests that our technique for finding similar hosts continues to produce plausible results.

For models trained on binary fingerprints, we can also measure the contribution of each variable to the trained model.[5] We can then aggregate feature importances that fall under any node in the document tree, resulting in the importance of specific fields in Censys documents (e.g. data observed on different ports) for the decision making process. For instance in Figure 4.1, the contribution of features extracted from the Content-Type header are accumulated to obtain the contribution of this field to the classifier, which is in turn added to the cumulative contribution of all HTTP headers for prediction. For classifiers examined in the first column of Table 4.5, we have included a break-down of the average contribution from different sections of the document in Table 4.6. We observe that ownership (AS) and geolocation properties have a major impact on classifier predictions, indicating that hosts

[5]The utilized technique for generating feature importances [57], computes the contribution of features to an individual decision tree, by iterating over all (non-leaf) nodes in the tree, and calculating "the reduction in node purity from the split, and attributing it to the feature that was split on", according to the following description: https://stats.stackexchange.com/questions/162162. Feature importances for the entire ensemble are then computed by averaging over all estimators.

belonging to different regions or ASes tend to behave differently when it comes to choosing web server products. We also notice that data from HTTP headers (except the Server header itself), as well as other information extracted from the HTTP response, also present high importance in the resulting models. We further observe a high contribution from attributes of the HTTPS protcol, the secure version of HTTP, given the close coupling between the two. Data from the remaining protocols, the most notable being SSH, are only minor contributors to our estimators; however, overall they constitute 11.5% of contributions to the decisions of trained classifiers. This further motivates the use of a diverse feature set, including cross-protocol information, for inference tasks regarding Internet hosts, even those that may only target a single port as is the case with the study herein.

While we have evaluated our methodology for one case study, the same concept can be applied to other protocols, such as FTP, different types of mail servers, or for OS fingerprinting. Furthermore, it would be interesting to acquire independent measurements of hidden attributes, in order to evaluate how our 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.

4.5.3 Network signatures

Thus far we have inspected the utility of our host fingerprints for classification at the IP address level. However, another potential application of our numerical fingerprints, is aggregating them over arbitrary boundaries, in order to obtain signatures for networks of IP addresses. Characterizing collections of hosts, for instance Internet-facing servers in any autonomous system (AS), or an organizational network, has previously been applied for forecasting cyber-risk [86]. Constructing network signatures by aggregating our host fingerprints at the network level, can capture many granular aspects of networks; this can in turn lead to more accurate models for characterizing the underlying entities, e.g. businesses, or ISPs. The aggregated representations of networks can then be used similarly to their host-level counterparts, e.g. for supervised classification.

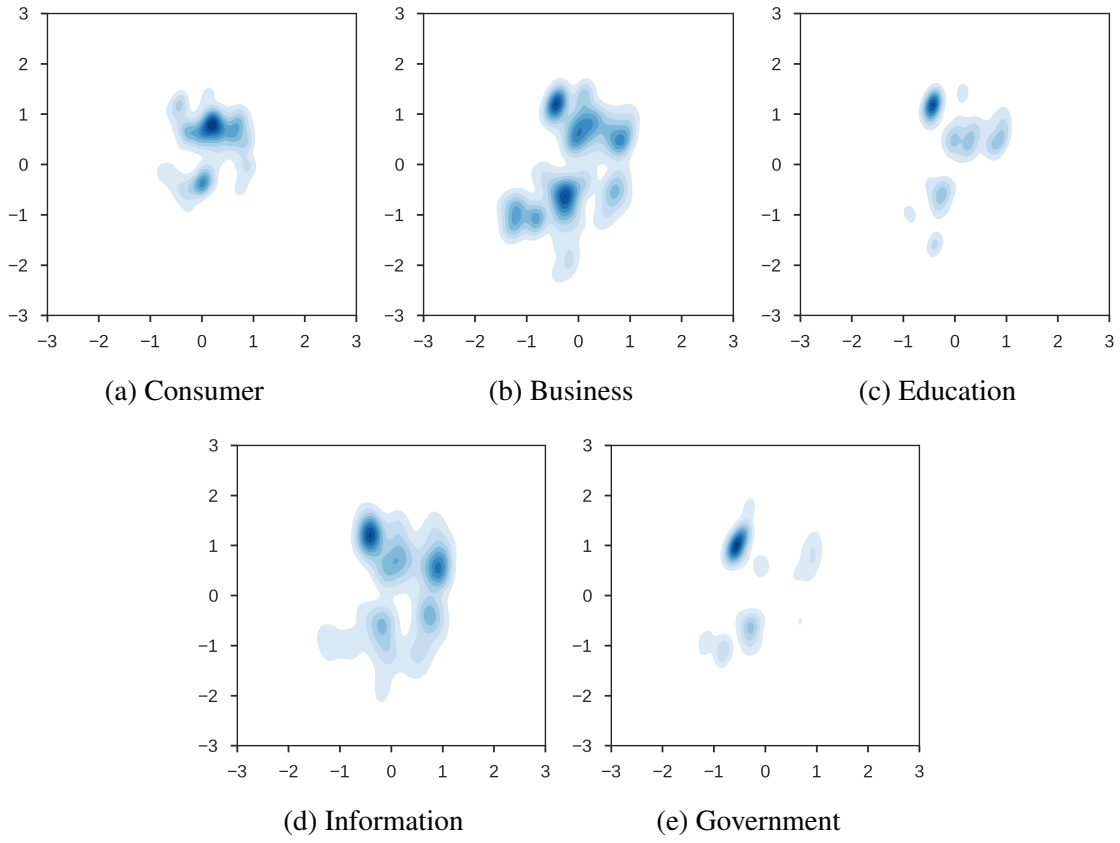
In this section, we will examine this technique for detection of different categories of networks. More specifically, we will use information provided by MaxMind, to train a multi-class classifier at the AS level, in order to identify `business`, `consumer`, `education`, `government`, and `information` networks. For aggregation boundaries, we obtain descriptions of ASes from the MaxMind GeoIP2 ISP database [89]. We use a snapshot collected

Category (MaxMind)	Frequency	Category	Frequency
Residential	50.4%	Consumer	53.0%
Cellular	2.3%		
Traveler	0.3%		
Dialup	0.1%		
Business	32.6%	Business	32.6%
College	4.6%	Education	5.2%
School	0.6%		
Library	0.1%		
Hosting	2.6%	Information	2.9%
Content delivery network	0.2%		
Search engine spider	0.1%		
Government	1.3%	Government	1.4%
Military	0.1%		
Unknown	4.8%	Unknown	4.8%

Table 4.7: Categorization of user types for queried IP addresses from MaxMind, and the corresponding categories for the study herein.

on 2017-10-31, containing address blocks assigned to 58 622 ASes by the Regional Internet Registries, to group probed hosts from a Censys snapshot obtained on 2017-11-01. It is worth noting that networks with zero, or very few *visible* hosts (i.e. hosts with corresponding records in Censys) cannot be characterized accurately. Therefore, we ignore networks with less that 20 visible IP addresses, resulting in 41 027 ASes. The ignored networks contain only 0.1% of probed hosts in Censys, and correspond to 2.0% of the allocated IPv4 address space, i.e. combined size of all 58 622 ASes; autonomous systems with no visible IPs correspond to 0.7% of the allocated space.

Label curation To procure labels for classification, we utilize the MaxMind GeoIP2 precision services [90]. Note, however, that the aforementioned service provides user types at the IP address level. Therefore, we query this database using five sample IPs from 8372 randomly picked autonomous systems. We use the mapping provided in Table 4.7 to convert user types returned by MaxMind, to one of five aforementioned categories, in addition to an `unknown` category; we have also included the frequency of each user type in Table 4.7. We then label each AS using the most reported category (from the third column in Table 4.7). For the examined networks, we observe that 7327 (91.9%) exhibit the same user type for all queried IP addresses, while 7679 (96.3%) and 7957 (99.8%) return the same category for 4/5 and 3/5 IPs, respectively. This indicates that for the majority of ASes, our



(a) Consumer (b) Business (c) Education

(d) Information (e) Government

Figure 4.7: 2-dimensional representations of IPs from different categories of autonomous systems. Each plot is drawn using 20 000 sample IP addresses from the depicted category. The observed differences in the structure of these networks can be used by a supervised learning algorithm to distinguish between them.

ground-truth data set does not provide more granularity than the AS level, and aggregating our fingerprints at this level is not far too coarse for the purpose of this study. For the remainder of our analysis, we also remove networks with an unknown label, resulting in a data set comprised of 7972 samples. We use 80% of our samples for training, and evaluate trained estimators over the remaining held-out test samples.

Aggregation In order to obtain a numerical representation of each network in our curated data set, we first obtain binary fingerprints of hosts from the 2017-11-01 Censys snapshot, and group them according to the aggregation boundaries provided by MaxMind. However, given that large ASes can contain millions of discoverable IP addresses, we select up to 1000 random hosts from each network, using out-of-bag sample without replacement, as a representative subset; this significantly reduces computational and memory requirements for this type of analysis. Figure 4.7 depicts the 2-dimensional visualization of each of

the five AS categories; each figure has been generated by concatenating sample IPs from ASes under the inspected category, and drawing the distribution for 20 000 sample points from the resulting set. As is evident from Figure 4.7, we see a clear distinction between the illustrated categories of IP addresses. For instance, hosts belonging to businesses and information companies exhibit more diversity, while the other three categories form more concentrated clusters; this is possibly due to the fact that the former typically include various types of servers for providing services to clients, e.g. web and mail servers, which can potentially manifest more variability in their probed configurations.

For classification, we explore various techniques for aggregating our host-level fingerprints, including averaging binary (latent) fingerprints, binning low-dimensional (2-10) latent embeddings and reporting the resulting histograms, clustering latent fingerprints and concatenating the centroid and density of the obtained clusters, and neural network models that process a collection of latent fingerprints to produce a multi-class output. However, our results suggest that simply averaging our binary fingerprints, i.e. computing the percentage of hosts that exhibit each of the 11 156 binary features, produces the most accurate representation for the current application. This is partly owing to the granularity of our binary tags, as we will discuss in more detail later in this section. We further append the size of the AS, and the number and percentage of active IP addresses, to the extracted features, resulting in a feature set comprised of 11 156 variables.

Evaluation For this experiment, we use gradient-boosted trees consisting of 100 estimators with a learning rate of 0.1. To reduce model complexity and prevent over-fitting, we limit the depth of decision trees to five, and train each estimator on a random subset containing 80% of the training set, and using 80% of available features. We also set the minimum number of samples that a decision tree leaf can represent to 10; this parameters further reduces over-fitting by preventing the training algorithm from further splitting such nodes. Additionally, given that we are using a relatively small training set, we also use recursive feature elimination to reduce the number of utilized features to 1000. This is achieved by training an initial estimator on all 11 159 features, computing the contribution of each variable to the resulting classifier according to the algorithm detailed in Footnote 5, and eliminating the least important features, along with those that are not used in any of the trained estimators; the described procedure is continued until we reach the targeted number of dimensions.

Table 4.8 includes the results of our evaluation; for each statistic we have included its mean and standard deviation for five different runs. We observe an overall accuracy of 73.7 ± 0.9 for the trained supervised models. Note that while we do not observe compa-

Category	Recall (%)	Precision (%)	Predicted categories (%)				
			Consumer	Business	Education	Information	Government
Consumer (55.7%)	78.9 ± 1.3	79.9 ± 0.9	78.9 ± 1.3	20.0 ± 1.1	0.7 ± 0.2	0.3 ± 0.2	0.2 ± 0.1
Business (34.4%)	72.1 ± 1.7	64.6 ± 1.5	25.2 ± 1.5	72.1 ± 1.7	1.6 ± 0.4	0.4 ± 0.3	0.7 ± 0.2
Education (5.4%)	57.7 ± 4.6	77.0 ± 2.2	22.3 ± 2.1	19.5 ± 3.0	57.7 ± 4.6	0.0 ± 0.0	0.5 ± 0.9
Information (3.0%)	39.2 ± 7.1	78.4 ± 6.4	32.1 ± 5.4	28.8 ± 11.0	0.0 ± 0.0	39.2 ± 7.1	0.0 ± 0.0
Government (1.4%)	44.5 ± 7.3	65.7 ± 9.3	13.6 ± 9.5	40.9 ± 11.1	0.9 ± 1.8	0.0 ± 0.0	44.5 ± 7.3

Table 4.8: Recall (true positive rate), precision, and breakdown of the predicted classes, for different AS categories. Scores are reported over five different runs, and correspond to an overall accuracy of 73.7 ± 0.9.

rable accuracies to those obtained from the previous case studies, Table 4.8 demonstrates a clear correlation between the extracted signatures of networks, and the examined categories of ASes. Furthermore, our results significantly improve upon a more naive approach to the task at hand: simply training a classifier at the IP addresses level, and averaging predictions over AS boundaries, yields an accuracy of roughly 50%; this further supports our technique for characterizing networks. For instance, observing only a few indicators, such as hosts that utilize `.edu` domains, can provide strong evidence for an educational network. Thus, it is more appropriate to aggregate measurements prior to feeding them to a supervised model, in order to examine networks as a whole, and not at the host-level. Interestingly, inspecting the top 100 features in our trained models, we observe the examination of top-level domains extracted from banners and subject common and alternative names of certificates. The importance of fine-grained characteristics of networks for the examined task, can also explain the inferior performance of models that leverage latent fingerprints, as the aforementioned attributes are possibly lost as a result of the lossy nature of latent embeddings, as well as the aggregation process.

Note that for educational, information, and government networks, there exist a small number of mis-categorized test samples, despite the rarity of such networks. We inspected these examples for one of our classifiers, corresponding to an AUC of 73.3%. Interestingly, for mis-classified educational, information, and government networks, we found that 10/13, 6/6, and 9/10 of the examined ASes were either given an incorrect label in our ground-truth data set, or the underlying organizations conducted business at some capacity in the predicted industries. We have included a number of these examples in Table 4.9, where we have included the AS description from Maxmind, the number of visible hosts in the network from Censys, and the ground-truth and predicted labels. For instance, the Energy Sciences Network (ESnet) is the United States Department of Energy's dedicated science network; hence, it can be categorized as both an educational, and a government net-

Autonomous system description	Size	MaxMind category	Predicted category
Colombian National Research and Education Network	418	Consumer	Education
Kalamazoo Regional Educational Service Agency (RESA)	237	Business	Education
Independent College Enterprise, Inc.	60	Consumer	Education
Holy Family University	34	Business	Education
Vivio Technologies	4660	Business	Information
Argon Data Communication (ArgonHost)	1264	Consumer	Information
North Hosts Limited	244	Consumer	Information
United States Department of Justice	1370	Business	Government
Energy Sciences Network (ESnet)	502	Education	Government
Spacenet, Inc. (absorbed by SageNet in 2014)	40	Consumer	Government

Table 4.9: Examined mis-categories networks. The presented samples either correspond to an error in the MaxMind data set, or identify a business with an ambiguous industry.

work. Furthermore, Spacenet (currently part of SageNet following its acquisition in 2014), provides services to businesses, as well as government agencies.

The presented cases in Table 4.9 shed light on the difficulty of the examined classification task, even for a human observer, which can further explain the lower accuracies observed in this section, as compared to the previous supervised problems. Furtermore, it shows how our trained classifier can be used to selectively audit and correct conspicuous errors in the curated ground-truth labels. Finally, the same concept can be applied to other classification problems at the network level, e.g. assessing the security posture of organizational networks, to compliment our study in Chapter 2. It would be interesting to investigate whether signatures obtained from latent embeddings can prove useful for other experiments, as they can be used to reveal certain clusterings of (unsecured) hosts in a network, as we have seen with a number of examples for 2-dimensional representations in this chapter; a shortcoming of signatures extracted from binary fingerprints.

4.6 Discussion

In the previous section, we provided examples on how to use the fingerprints obtained in Sections 4.3 and 4.4, in conjunction with learning algorithms, for three real-world applications. Moreover, we evaluated our framework using Censys as the main data source for providing demographic and probed measurement over the entire IPv4 space. Note, however, 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 4.3, and the applications for which it is ultimately utilized. In practice, research environments and production systems can install the proposed tools on top of their own databases, thus adding visualization and machine learning capabilities to their existing toolkits.

4.6.1 Privacy

The privacy-preservation of machine learning algorithms is often a concern for big data systems [36, 101]. Note that our measurements have been collected on the public Internet, and are not as sensitive as other types of documents, such as medical records. However, one may still want to anonymize samples in a specific collection, e.g. the identity of samples used in a machine learning model. For each of the four global IPv4 snapshots in this study, we found that ~17% of obtained fingerprints were also unique identifiers for their respective IP addresses. Hence, our binary fingerprints are not appropriate representations for the purpose of privacy. However, the lossy nature of our latent fingerprints, discussed in Table 4.2, make them an appealing choice. One can also further add noise to anonymized samples in order to gain additional privacy, which can easily be achieved for latent embeddings due to the continuous nature of these variables. A full analysis of the trade-off between privacy and accuracy of trained models and statistics collected on anonymized data sets is an interesting extension of the current work.

4.6.2 Fingerprinting IPv6 hosts

More users are adopting IPv6 [53], with more than 25% of Alexa Top 1000 websites currently reachable over IPv6 [148], where obtaining global measurements is infeasible. While we do not directly evaluate our methods over this space, the proposed techniques can also be applied to measurements collected on IPv6 hosts, e.g. through targeted scanning [96]. This can be achieved by reusing IPv4 models, or training new models on a representative subset of IPv6 hosts, in order 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 fast analysis.

4.7 Related work

There has been an increasing number of studies using machine learning techniques to process Internet measurement as well as social media data. For instance, vulnerability and

Twitter data are analyzed in [122] to train classifiers for detecting software exploits. A semi-supervised learning method was used in [118] to process social media data to identify un-reported data breaches. A supervised learning approach was used in [86] on Internet host measurement data to perform data breach prediction. 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 fingerprints for a diverse set of applications. Most importantly, our techniques retain the scalability required for large-scale analysis.

Previous work on fingerprinting include [130, 131, 146] for OS fingerprinting, [43, 75, 84] for profiling physical devices, and [21] for automated fingerprint generation. However, instead of focusing on specific attributes, e.g. the installed operating system, producing unique fingerprints for device identification, or customizing probes for a specific application, we propose a more general approach for fingerprinting hosts by consolidating an existing, though diverse, set of measurements. Furthermore, we generate machine learning compatible representations, which can then be customized for a specific application using state-of-the-art learning techniques.

On detection of malicious websites and intelligent phishing classification, Zhang et al. [152] examine the content of webpages, using the TF-IDF algorithm to extract information from the content and detect phishing websites. Afroz et al. [3] use content similarities in order to spot malicious sites imitating the appearance of legitimate websites. Soska et al. [134] evaluate 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. [85], and Mekky et al. [91], find malicious websites by examining HTTP redirections. Shibahara et al. [132] 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. Our proposed method for identifying malicious servers can perform both detection and forecasting by comparing the recorded fingerprint of a host with previously blacklisted servers, and can complement other approaches by tapping into an orthogonal source of measurements.

4.8 Conclusion

In this chapter, we have demonstrated a novel approach to distilling an extremely broad range of characteristics of IP addresses into two different types of hosts fingerprints, while still maintaining the richness of the high-fidelity characteristics. We develop a novel

method to extract binary feature vectors from tree-like documents that describe attributes of Internet-facing hosts, and utilize recent developments for unsupervised modeling of large-scale data to find low-dimensional representations of our high-dimensional, though sparse, binary fingerprints. For our latent variable model, we compare two different generative models, namely variational autoencoders and restricted Boltzmann machines. Through empirical evaluation of trained models, we conclude that a variational autoencoder is a more appropriate choice for training deep graphical models that can identify higher order interactions between different measurements, and use those patterns to encode compact, yet representative, numerical fingerprints. These encoded features can be thought of as latent variables that drive the signals observed by Censys scanners, such as location, in addition to human factors summarizing the behavioral patterns and (malicious) intentions of those managing the observable Internet hosts, e.g. network administrators. To the best of our knowledge, this is the first study on producing flexible numerical fingerprints, which can then be customized through machine learning algorithms.

The latent representations learned as a result of our methodology can be used for visualizing hosts, in order to help researchers, security analysts and networking experts to identify and inspect patterns for clustering of IP addresses, and recognizing outliers. We also explore the application of these tools for three practical case studies: (1) quantifying the maliciousness of hosts, (2) inferring hidden attributes of IP addresses, more specifically for detecting the deployed product for serving web (HTTP) content, and (3) categorizing autonomous systems, by inspecting the joint distribution of hosts in a network. The ability to manipulate and analyze our numerical fingerprints using fast learnable transformations, e.g. using gradient-boosted trees, enables an entire new class of security analysis that were previously not feasible, or could only be performed inefficiently, such as on-demand fingerprinting of specific software, or physical devices.

Note that a drawback of current fingerprinting techniques, is their inability in detecting similar hosts. Thus, throughout our experiments, we have evaluated the performance of k-NN classifiers, in order to examine the efficacy of distances in the latent space (using 10-dimensional embeddings) for quantifying host similarities. Our results indicate that our numerical fingerprints also enable fuzzy-matching of hosts, producing a similarity measure that is robust and suitable for various applications. This property, along with the low dimensionality of latent embedding, allows us to efficiently use the nearest-neighbors algorithm in order to query for similar hosts within a large corpus of fingerprints, which can then be leveraged to inspect and interpret predictions.

For estimators that utilize binary tags for inferring hidden attributes and categorizing networks, we also inspect the resulting classifiers to gain insight into the contribution of

individual features, and to identify major drivers in the decision making process. However, one drawback of algorithms based on latent embeddings, is the loss of interpretability, as with virtually all deep learning models. While the latent variables contain a multitude of knowledge on each arbitrary host, it is often unclear what each one is representing. It is up to the person using these tools to interpret the results for specific tasks, by using a combination of visualization and spot checking of samples. We have shown an example of how to alleviate this issue by associating hosts with similar and labeled IP addresses, using the nearest-neighbors algorithm, a capability offered by latent fingerprints due to their low dimensionality. Nonetheless, we emphasize the need for the proposed techniques, to complement other more interpretable models, in tasks for which scalability and feasibility become an issue. Furthermore, highly sparse features, measured by the fraction of hosts exhibiting certain characteristics, and new software and protocols introduced at later times would not be captured by our methodology; to alleviate these issues we would need to re-train models over more recent snapshots, or tweak hyper-parameters in order to allow for more sparse features.

CHAPTER 5

Conclusion

5.1 Summary of contributions

In this thesis, we have presented a number of microscopic and macroscopic security-oriented studies (with further applications in network monitoring), combining measurements on various types of entities, with statistical tools and machine learning methods, in order to assess their security posture and privacy, detect maliciousness, and make high-level deductions based on our findings. We choose our tools in accordance with the complexity of the tasks at hand, the volume of our data sets and labeled samples, and the quality and quantity of available information. Our statistical analysis of entities security reveals two major insights: (1) the need for interpretable designs, which in turn lead to actionable recommendations and forecasts, and (2) the significance of transparency and data availability for producing objective and unbiased models, for the betterment of security. Our main contributions and findings are summarized below.

Fine grained data breach prediction In Chapter 2, we use publicly available information about businesses, combined with labels curated from previously disclosed data breaches, in order to train a set of classifiers for assessing risk of experiencing data incidents. In addition to estimating the overall likelihood of suffering from a data breach, we also estimate risk distributions over multiple categorizations, including incident types (hacking, error, misuse, etc.), actors (internal, external, partners), and affected assets. Our assessments can help organizations choose an optimal level of investment in preventative and self-protection measures, as well as guide the allotment of resources, and identify weakest links. Furthermore, the public availability of utilized features, enabled third parties, such as insurance underwriters, to assess the security for designing more optimal cyber-insurance contracts term, including premiums and coverage levels.

For overall risk estimation, we obtain a true positive rate of 90%, while while keeping

false positives as low as 11%, and an area under the curve (AUC) score of 95%; these statistics are collected over a held-out test set. For estimating risk distributions, we use a combination of prediction histograms and case studies, in order to demonstrate how our methodology can produce uniform or concentrated risk profiles based on the nature of the underlying business. For the latter, organizations can use our findings to focus on a smaller set of incident types, thus achieving the same level of protection by spending less resources on security through more prudent prioritization.

Modeling end-user patching behavior In Chapter 3, we tap into field measurements of patch deployment, in order to quantify and model user behavior, when it comes to applying software patches. We leverage various techniques for extracting precise ground-truth information indicative of a user's delay in installing software patches, for four client-side applications: Google Chrome, Mozilla Firefox and Thunderbird, and Adobe Flash Player. These include finding exact product release dates, and overcoming challenges regarding parallel product lines. Our findings suggest that user behavior is memory-less (i.e. the delay in applying patches follows an exponential distribution); we use statistical tests, namely the chi-squared test, to support our claims. Furthermore, we observe that user behavior is largely consistent across different countries.

Following the simple-mindedness of user behavior, we further measure its effects on the end-host vulnerability state, over an extended observation window spanning many vulnerability disclosures. We observe that silent updates do lead to shorter windows of vulnerability; however, even with silent updates, the majority of hosts have long windows of susceptibility. This is partly due to the memory-less property of user behavior, combined with the large number of software flaws that affect the examined products, which results in long windows of susceptibility that can be exploited by miscreants for compromising user machines. We also observe this phenomena for exploited vulnerabilities of Flash Player, though at a lesser extent.

Numerical fingerprinting of Internet hosts In Chapter 4, we conduct a fine-grained analysis of Internet hosts, using recent advances in network scanning, to distill a wide range of available measurements on the IPv4 address space, into compact numerical fingerprints. We adopt a novel approach to host fingerprinting, by producing two types of machine learning compatible representations: (1) high-dimensional, yet sparse, binary fingerprints in the form of various tags that have been associated with a hosts, and (2) low-dimensional latent embeddings of our high-dimensional representations. To this end, we develop a technique for extracting binary feature vectors from tree-like documents containing a wide variety of

measurements, both in breadth (number of scanned ports/protocols) and depth (attributes of a specific protocol), on probed IP addresses. Furthermore, we use a recent method for training deep generative stochastic models, namely variational autoencoders [73, 117], in order to perform dimensionality reduction on binary fingerprints.

We examine the utility of binary and latent fingerprints for supervised learning tasks, namely detecting/predicting malicious hosts, and inferring installed web server products (when this information is not directly disclosed). We show that the use of low-dimensional fingerprints can results in faster models, while requiring more advanced hardware, and preprocessing. Furthermore, our latent fingerprints are amenable to visualization, allowing collections of hosts to be visualized for readily inspecting distributions, and detecting clusters. Another capability of our low-dimensional representations, is that they can be used to quantity host similarities, resulting in an efficient implementation of the nearest neighbors algorithm for querying similar IP addresses from a large corpus of sample hosts. Finally, we aggregate our fingerprints at the network level, in order to categorize autonomous systems; this demonstrates the applicability of our technique for characterizing networks, and conducting macroscopic studies on the Internet.

5.2 Future directions

Our proposed framework for scalable analysis of Internet hosts can be extended in a variety of ways, e.g. applying it to other domains such as private networks, or partial scans on the IPv6 address space. One particularly interesting application of this technique is to develop a solution for protecting servers against unsecured/infected devices on the Internet. Currently, amplification attacks, and botnets consisting of compromised machines and IoT devices, are used by cyber-criminals to launch distributed denial of service (DDoS) attacks. Protecting against such attacks is fairly challenging; blackholing provides a viable DDoS mitigation strategy, though at the cost of losing legitimate traffic. A more effective protection method requires the ability to distinguish between benign and malicious traffic in real-time during an attack. Our host fingerprinting framework can potentially provide a scalable solution for this purpose, by obtaining signatures of unsecured/infected devices, or malicious networks, and producing real-time rules for filtering out malicious traffic during an attack. Curating data sets of previous DDoS campaigns, training machine learning algorithms for the identification of their sources, and evaluating their efficacy for protecting against future attacks, is a potential extension to the current work.

Finally, while machine learning, specifically neural network models, have been extensively applied for image, text, and voice recognition, their utility for processing structured

(and often nested) documents is fairly unexplored. Another extension to our presented techniques in Section 4, is applying them to other forms of data, such as webpages, health records, or business directories. Note, however, that while our binary fingerprints have been fairly successful in extracting information from Internet probes, this is not necessarily the case for other data sets. For instance, webpages' HTML codes can follow widely varying, often unique structures; in the absence of a shared schema, our proposed framework fails to extract meaningful representations from tree-like documents. Addressing these challenges and extending our methods is an interesting extension to the current work, providing valuable techniques for scalable manipulation of documents under a machine learning setting, for application in security, monitoring, and other domains.

BIBLIOGRAPHY

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

[2] Alessandro Acquisti, Allan Friedman, and Rahul Telang. Is there a cost to privacy breaches? An event study. *International Conference on Information Systems*, pages 1563–1580, 2006.

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

[4] Chirag Agarwal, Mehdi Sharifzhadeh, Joe Klobusicky, and Dan Schonfeld. CrossNets: A new approach to complex learning. *arXiv preprint arXiv:1705.07404*, 2017.

[5] Omar H Alhazmi and Yashwant K Malaiya. Modeling the vulnerability discovery process. In *IEEE International Symposium on Software Reliability Engineering*, pages 129–138. IEEE, 2005.

[6] Omar H Alhazmi, Yashwant K Malaiya, and Indrajit Ray. Measuring, analyzing and predicting security vulnerabilities in software systems. *Computers & Security*, 26(3):219–228, 2007.

[7] Amazon Web Services. Alexa top sites. https://aws.amazon.com/alexa-top-sites.

[8] Amazon Web Services. Alexa Web Information Service. http://aws.amazon.com/awis.

[9] Prasanth Anbalagan and Mladen Vouk. Towards a Bayesian approach in modeling the disclosure of unique security faults in open source projects. In *IEEE International Symposium on Software Reliability Engineering*, pages 101–110. IEEE, 2010.

[10] Ross Anderson, Chris Barton, Rainer Böhme, Richard Clayton, Michel JG Van Eeten, Michael Levi, Tyler Moore, and Stefan Savage. Measuring the cost of cybercrime. In *The Economics of Information Security and Privacy*, pages 265–300. Springer, 2013.

[11] Manos Antonakakis, Tim April, Michael Bailey, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Damian Menscher, Chad Seaman, Nick Sullivan, et al. Understanding the Mirai botnet. In *USENIX Security Symposium*, pages 1093–1110, 2017.

[12] William A Arbaugh, William L Fithen, and John McHugh. Windows of vulnerability: A case study analysis. *Computer*, 33(12):52–59, 2000.

[13] Ashish Arora, Ramayya Krishnan, Anand Nandkumar, Rahul Telang, and Yubao Yang. Impact of vulnerability disclosure and patch availability - An empirical analysis. In *Workshop on the Economics of Information Security*, volume 24, pages 1268–1287, 2004.

[14] Ashish Arora, Rahul Telang, and Hao Xu. Optimal policy for software vulnerability disclosure. *Management Science*, 54(4):642–656, 2008.

[15] Amazon Web Services (AWS). http://aws.amazon.com.

[16] Benjamin Aziz. Towards open data-driven evaluation of access control policies. *Computer Standards & Interfaces*, 2017.

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

[18] Leyla Bilge and Tudor Dumitraș. Before we knew it: An empirical study of zero-day attacks in the real world. In *ACM Conference on Computer and Communications Security*, pages 833–844. ACM, 2012.

[19] Mehran Bozorgi, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. Beyond heuristics: Learning to classify vulnerabilities and predict exploits. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 105–114. ACM, 2010.

[20] Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*, 2015.

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

[22] Katherine Campbell, Lawrence A Gordon, Martin P Loeb, and Lei Zhou. The economic cost of publicly announced information security breaches: Empirical evidence from the stock market. *Journal of Computer Security*, 11(3):431–448, 2003.

[23] Hasan Cavusoglu, Huseyin Cavusoglu, and Srinivasan Raghunathan. Emerging issues in responsible vulnerability disclosure. In *Workshop on the Economics of Information Security*, 2005.

[24] Huseyin Cavusoglu, Birendra Mishra, and Srinivasan Raghunathan. The effect of Internet security breach announcements on market value: Capital market reactions for breached firms and Internet security developers. *International Journal of Electronic Commerce*, 9(1):70–104, 2004.

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

[26] Sandy Clark, Michael Collis, Matt Blaze, and Jonathan M Smith. Moving targets: Security and rapid-release in Firefox. In *ACM Conference on Computer and Communications Security*, pages 1256–1266. ACM, 2014.

[27] Sandy Clark, Stefan Frei, Matt Blaze, and Jonathan Smith. Familiarity breeds contempt: The honeymoon effect and the role of legacy code in zero-day vulnerabilities. In *Computer Security Applications Conference*, pages 251–260. ACM, 2010.

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

[29] DMOZ. The Open Directory Project (ODP). http://www.dmoz.org.

[30] Thomas Duebendorfer and Stefan Frei. Web browser security update effectiveness. In *International Workshop on Critical Information Infrastructures Security*, pages 124–137. Springer, 2009.

[31] Tudor Dumitraș and Darren Shou. Toward a standard benchmark for computer security research: The Worldwide Intelligence Network Environment (WINE). In *Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, pages 89–96. ACM, 2011.

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

[33] Zakir Durumeric, David Adrian, Ariana Mirian, James Kasten, Elie Bursztein, Nicolas Lidzborski, Kurt Thomas, Vijay Eranti, Michael Bailey, and J Alex Halderman. Neither snow nor rain nor MITM...: An empirical analysis of email delivery security. In *Internet Measurement Conference*, pages 27–39. ACM, 2015.

[34] Zakir Durumeric, James Kasten, David Adrian, J Alex Halderman, Michael Bailey, Frank Li, Nicolas Weaver, Johanna Amann, Jethro Beekman, Mathias Payer, et al. The matter of Heartbleed. In *Internet Measurement Conference*, pages 475–488. ACM, 2014.

[35] Zakir Durumeric, Eric Wustrow, and J Alex Halderman. ZMap: Fast Internet-wide scanning and its security applications. In *USENIX Security Symposium*, volume 8, pages 47–53, 2013.

[36] Cynthia Dwork. Differential privacy. In *Encyclopedia of Cryptography and Security*, pages 338–340. Springer, 2011.

[37] Benjamin Edwards, Steven Hofmeyr, and Stephanie Forrest. Hype and heavy tails: A closer look at data breaches. *Journal of Cybersecurity*, 2(1):3–14, 2016.

[38] Pete Evans. Heartbleed bug: RCMP asked Revenue Canada to delay news of SIN thefts. http://www.cbc.ca/news/business/heartbleed-bug-rcmp-asked-revenue-canada-to-delay-news-of-sin-thefts-1.2609192, 2014.

[39] Pete Evans. Heartbleed bug undoes web encryption, reveals Yahoo passwords. http://www.cnet.com/news/heartbleed-bug-undoes-web-encryption-reveals-user-passwords, 2014.

[40] Exploit kits. http://contagiodump.blogspot.com.

[41] Farzaneh Farhadi, Hamidreza Tavafoghi, Demosthenis Teneketzis, and Jamal Golestani. A dynamic incentive mechanism for security in networks of interdependent agents. In *International Conference on Game Theory for Networks*, pages 86–96. Springer, 2017.

[42] Sadegh Farhang and Jens Grossklags. When to invest in security? Empirical evidence and a game-theoretic approach for time-based security. *arXiv preprint arXiv:1706.00302*, 2017.

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

[44] Jason Franklin, Adrian Perrig, Vern Paxson, and Stefan Savage. An inquiry into the nature and causes of the wealth of Internet miscreants. In *ACM Conference on Computer and Communications Security*, pages 375–388, 2007.

[45] Stefan Frei. *Security Econometrics*. 2009.

[46] Stefan Frei and Thomas Kristensen. The security exposure of software portfolios. In *RSA Conference*, 2010.

[47] Sam Frizell. Devastating Heartbleed flaw was used in hospital hack. http://time.com/3148773/report-devastating-heartbleed-flaw-was-used-in-hospital-hack, 2014.

[48] Josh Fruhlinger. Petya and NotPetya: The basics. https://www.csoonline.com/article/3233210.

[49] Josh Fruhlinger. WannaCry ransomware explained: What it is, how it infects, and who was responsible. https://www.csoonline.com/article/3227906.

[50] Christos Gkantsidis, Thomas Karagiannis, and Milan VojnoviC. Planet scale software updates. *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 36(4):423–434, 2006.

[51] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[52] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

[53] Google. IPv6 statistics. https://www.google.com/intl/en/ipv6/statistics.html.

[54] Google Chrome Help. Use or fix Flash audio & video. https://support.google.com/chrome/answer/6258784.

[55] Lawrence A Gordon, Martin P Loeb, and Lei Zhou. The impact of information security breaches: Has there been a downward shift in costs? *Journal of Computer Security*, 19(1):33–56, 2011.

[56] Chris Grier, Lucas Ballard, Juan Caballero, Neha Chachra, Christian J Dietrich, Kirill Levchenko, Panayiotis Mavrommatis, Damon McCoy, Antonio Nappa, Andreas Pitsillidis, et al. Manufacturing compromise: The emergence of exploit-as-a-service. In *ACM Conference on Computer and Communications Security*, pages 821–832. ACM, 2012.

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

[58] Christopher Hines. Why companies must adopt the "assume mentality" when it comes to breaches. https://blog.cloudsecurityalliance.org/2015/02/27/why-companies-must-adopt-the-assume-mentality-when-it-comes-to-breaches.

[59] Geoffrey E Hinton. Training products of experts by minimizing contrastive divergence. *Training*, 14(8), 2006.

[60] Geoffrey E Hinton. A practical guide to training restricted Boltzmann machines. In *Neural Networks: Tricks of the Trade*, pages 599–619. Springer, 2012.

[61] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[62] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

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

[64] IBM. Ponemon cost of data breach study. https://www.ibm.com/security/data-breach.

[65] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[66] Ian T Jolliffe. Principal component analysis and factor analysis. In *Principal Component Analysis*, pages 115–128. Springer, 1986.

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

[68] Karthik Kannan, Jackie Rees, and Sanjay Sridhar. Market reactions to information security breach announcements: An empirical analysis. *International Journal of Electronic Commerce*, 12(1):69–91, 2007.

[69] Mohammad Mahdi Khalili, Parinaz Naghizadeh, and Mingyan Liu. Designing cyber insurance policies in the presence of security interdependence. In *Workshop on the Economics of Networks, Systems and Computation*. ACM, 2017.

[70] Mohammad Mahdi Khalili, Parinaz Naghizadeh, and Mingyan Liu. Designing cyber insurance policies: Mitigating moral hazard through security pre-screening. In *International Conference on Game Theory for Networks*, pages 63–73. Springer, 2017.

[71] Ross Kindermann and J Laurie Snell. *Markov Random Fields and Their Applications*, volume 1. American Mathematical Society, 1980.

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

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

[74] Donald E Knuth. *The Art of Computer Programming, Volume II: Seminumerical Algorithms, 2nd Edition*. Addison-Wesley, 1981.

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

[76] Vadim Kotov and Fabio Massacci. Anatomy of exploit kits. *Engineering Secure Software and Systems*, 7781:181–196, 2013.

[77] Krebs on Security. Anthem breach may have started in April 2014. https://krebsonsecurity.com/2015/02/anthem-breach-may-have-started-in-april-2014.

[78] Krebs on Security. The Equifax breach: What you should know. https://krebsonsecurity.com/2017/09/the-equifax-breach-what-you-should-know.

[79] Krebs on Security. The Target breach, by the numbers. https://krebsonsecurity.com/2014/05/the-target-breach-by-the-numbers.

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

[81] Richard J La. Effects of degree correlations in interdependent security: Good or bad? *IEEE/ACM Transactions on Networking*, 2017.

[82] Aron Laszka, Mark Felegyhazi, and Levente Buttyan. A survey of interdependent information security games. *ACM Computing Surveys*, 47(2), 2015.

[83] Da Young Lee, Mladen Vouk, and Laurie Williams. Using software reliability models for security assessment - verification of assumptions. In *IEEE International Symposium on Software Reliability Engineering Workshops*, pages 23–24. IEEE, 2013.

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

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

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

[87] Fabio Massaccia, Joe Swierzbinskic, and Julian Williamsb. Cyberinsurance and public policy: Self-protection and insurance with endogenous adversaries. In *Workshop on the Economics of Information Security*, 2017.

[88] Arunesh Mathur, Josefine Engel, Sonam Sobti, Victoria Chang, and Marshini Chetty. "They keep coming back like zombies": Improving software updating interfaces. In *Symposium on Usable Privacy and Security*, pages 43–58, 2016.

[89] MaxMind. GeoIP2 databases. https://www.maxmind.com/en/geoip2-databases.

[90] MaxMind. GeoIP2 precision services. https://www.maxmind.com/en/geoip2-precision-services.

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

[92] Merit network. https://www.merit.edu.

[93] David Moore, Colleen Shannon, and Kimberly Claffy. Code-Red: A case study on the spread and victims of an Internet worm. In *ACM SIGCOMM Workshop on Internet Measurment*, pages 273–284. ACM, 2002.

[94] MozillaZine knowledge base. Flash. http://kb.mozillazine.org/Flash.

[95] Deirdre K Mulligan and Fred B Schneider. Doctrine for cybersecurity. *Daedalus*, 140(4):70–92, 2011.

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

[97] Parinaz Naghizadeh Ardabili. On the provision of public goods on networks: Incentives, exit equilibrium, and applications to cyber. 2016.

[98] NAICS association. http://www.naics.com.

[99] Antonio Nappa, Richard Johnson, Leyla Bilge, Juan Caballero, and Tudor Dumitraș. The attack of the clones: A study of the impact of shared code on vulnerability patching. In *IEEE Symposium on Security and Privacy*, pages 692–708. IEEE, 2015.

[100] Antonio Nappa, Zhaoyan Xu, M Zubair Rafique, Juan Caballero, and Guofei Gu. Cyberprobe: Towards Internet-scale active detection of malicious servers. In *Network and Distributed System Security Symposium*, pages 1–15, 2014.

[101] Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large sparse datasets. In *IEEE Symposium on Security and Privacy*, pages 111–125. IEEE, 2008.

[102] Kartik Nayak, Daniel Marino, Petros Efstathopoulos, and Tudor Dumitraș. Some vulnerabilities are different than others. In *International Workshop on Recent Advances in Intrusion Detection*, pages 426–446. Springer, 2014.

[103] Stephan Neuhaus, Thomas Zimmermann, Christian Holler, and Andreas Zeller. Predicting vulnerable software components. In *ACM Conference on Computer and Communications Security*, pages 529–540. ACM, 2007.

[104] Neustar. IP Intelligence. https://www.neustar.biz/services/ip-intelligence.

[105] NIST. National Vulnerability Database (NVD). https://nvd.nist.gov.

[106] Nmap: The network mapper. https://nmap.org.

[107] Hamed Okhravi and David Nicol. Evaluation of patch management strategies. *International Journal of Computational Intelligence: Theory and Practice*, 3(2):109–117, 2008.

[108] Andy Ozment and Stuart E Schechter. Milk or wine: Does software security improve with age? In *USENIX Security Symposium*, pages 93–104, 2006.

[109] Ranjan Pal, Leana Golubchik, Konstantinos Psounis, and Pan Hui. Will cyber-insurance improve network security? a market analysis. In *IEEE International Conference on Computer Communications*, pages 235–243. IEEE, 2014.

[110] Paulo Passeri. Hackmageddon. http://hackmageddon.com.

[111] Karl Pearson. LIII. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

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

[113] Privacy Rights Clearinghouse. Data breaches. https://www.privacyrights.org/data-breaches.

[114] Terry Ramos. The laws of vulnerabilities. In *RSA Conference*, 2006.

[115] Eric Rescorla. Security holes... Who cares? In *USENIX Security Symposium*, pages 75–90, 2003.

[116] Eric Rescorla. Is finding security holes a good idea? *IEEE Security & Privacy*, 3(1):14–19, 2005.

[117] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic back-propagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

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

[119] Sasha Romanosky, Lilian Ablon, Andreas Kuehn, and Therese Jones. Content analysis of cyber insurance policies: How do carriers write policies and price cyber risk? In *Workshop on the Economics of Information Security*, 2017.

[120] Sasha Romanosky, David Hoffman, and Alessandro Acquisti. Empirical analysis of data breach litigation. *Journal of Empirical Legal Studies*, 11(1):74–104, 2014.

[121] Sasha Romanosky, Rahul Telang, and Alessandro Acquisti. Do data breach disclosure laws reduce identity theft? *Journal of Policy Analysis and Management*, 30(2):256–286, 2011.

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

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

[124] SANS Institute. CIS Critical Security Controls. https://www.sans.org/critical-security-controls.

[125] Armin Sarabi, Parinaz Naghizadeh, Yang Liu, and Mingyan Liu. Prioritizing security spending: A quantitative analysis of risk distributions for different business profiles. In *Workshop on the Economics of Information Security*, 2015.

[126] Armin Sarabi, Parinaz Naghizadeh, Yang Liu, and Mingyan Liu. Risky business: Fine-grained data breach prediction using business profiles. *Journal of Cybersecurity*, 2(1):15–28, 2016.

[127] Armin Sarabi, Ziyun Zhu, Chaowei Xiao, Mingyan Liu, and Tudor Dumitraş. Patch me if you can: A study on the effects of individual user behavior on the end-host vulnerability state. In *International Conference on Passive and Active Network Measurement*, pages 113–125. Springer, 2017.

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

[129] Muhammad Shahzad, Muhammad Zubair Shafiq, and Alex X Liu. A large scale exploratory analysis of software vulnerability life cycles. In *International Conference on Software Engineering*, pages 771–781. IEEE, 2012.

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

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

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

[133] Paul Smolensky. Information processing in dynamical systems: Foundations of harmony theory. Technical report, University of Colorado Boulder Computer Science Department, 1986.

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

[135] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[136] Symantec Corporation. Listing of threats & risks. https://www.symantec.com/security_response/landing/azlisting.jsp.

[137] Samaneh Tajalizadehkhoob, Tom van Goethem, Maciej Korczyński, Arman Noroozian, Rainer Böhme, Tyler Moore, Wouter Joosen, and Michel van Eeten. Herding vulnerable cats: a statistical approach to disentangle joint responsibility for web security in shared hosting. *arXiv preprint arXiv:1708.06693*, 2017.

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

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

[140] The Web Application Security Consortium. Web Hacking Incident Database. http://projects.webappsec.org/w/page/13246995/web-hacking-incident-database.

[141] Olivier Thonnard, Leyla Bilge, Anand Kashyap, and Martin Lee. Are you at risk? Profiling organizations and individuals subject to targeted attacks. In *International Conference on Financial Cryptography and Data Security*, pages 13–31. Springer, 2015.

[142] Kami E Vaniea, Emilee Rader, and Rick Wash. Betrayed by updates: How negative experiences affect future security. In *SIGCHI Conference on Human Factors in Computing Systems*, pages 2671–2674. ACM, 2014.

[143] VERIS Community Database (VCDB). http://vcdb.org.

[144] The VERIS framework. http://veriscommunity.net.

[145] Verizon Enterprise. Data Breach Investigations Report (DBIR). http://www.verizonenterprise.com/verizon-insights-lab/dbir.

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

[147] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *International Conference on Machine Learning*, pages 1096–1103. ACM, 2008.

[148] World IPv6 launch. http://www.worldlaunch.org/measurements.

[149] Scott Yilek, Eric Rescorla, Hovav Shacham, Brandon Enright, and Stefan Savage. When private keys are public: Results from the 2008 Debian OpenSSL vulnerability. In *ACM SIGCOMM Conference on Internet Measurement*, pages 15–27. ACM, 2009.

[150] Derek Young, Juan Lopez, Mason Rice, Benjamin Ramsey, and Robert McTasney. A framework for incorporating insurance in critical infrastructure cyber risk strategies. *International Journal of Critical Infrastructure Protection*, 14:43–57, 2016.

[151] Liang Zhang, David Choffnes, Dave Levin, Tudor Dumitraș, Alan Mislove, Aaron Schulman, and Christo Wilson. Analysis of SSL certificate reissues and revocations in the wake of Heartbleed. In *Internet Measurement Conference*, pages 489–502. ACM, 2014.

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

[153] Mingyi Zhao, Jens Grossklags, and Peng Liu. An empirical study of web vulnerability discovery ecosystems. In *ACM Conference on Computer and Communications Security*, pages 1105–1117. ACM, 2015.