Chapter 15
Security of Data Science and Data Science for Security

Bernhard Tellenbach, Marc Rennhard, and Remo Schweizer

Abstract In this chapter, we present a brief overview of important topics regarding the connection of data science and security. In the first part, we focus on the security of data science and discuss a selection of security aspects that data scientists should consider to make their services and products more secure. In the second part about security for data science, we switch sides and present some applications where data science plays a critical role in pushing the state-of-the-art in securing information systems. This includes a detailed look at the potential and challenges of applying machine learning to the problem of detecting obfuscated JavaScripts.

1 Introduction

Giants like Sony, Yahoo, and Anthem Inc., the second-largest US health insurance company, heavily rely on big data and machine learning systems to efficiently store and process huge amounts of data. But large enterprises are not the only ones; there are more and more startups and SMEs whose business model focuses on data-centric services and products. Unfortunately, where there is valuable data, there are also hackers that want to get it or manipulate it for fun and profit. It is therefore important that data scientists are aware of the fact that new services or data products should be designed with security in mind. Many of the popular technologies and algorithms used in their domain are not secure by default. They have to be used with care. For example, recent research showed that access to the public API of a classification service (e.g., face recognition) might be sufficient to steal or invert the underlying model (Tramèr et al. 2016). We refer to these aspects as *security of data science*, that is, issues related to the security of data science methods and applications.

On the other hand, data science methods and techniques help to address some of the most challenging problems in this field such as the management of huge amounts of log data and the identification of anomalies or other clues that might pinpoint

B. Tellenbach (✉) · M. Rennhard · R. Schweizer
ZHAW Zurich University of Applied Sciences, Winterthur, Switzerland
e-mail: bernhard.tellenbach@zhaw.ch

M. Braschler et al. (eds.), *Applied Data Science*,
https://doi.org/10.1007/978-3-030-11821-1_15

activities posing a risk for an organization. It is therefore not surprising that advancements in the field of data science lead to improvements of existing security products. For instance, becoming better at detecting anomalies in credit card transactions, network traffic, user behavior, and other types of data directly results in improved products to protect today's businesses. However, improvements to existing products are not the only outcome of the already fruitful relation of data science and security. It also led to the development of completely new solutions such as next-generation antivirus products (Cylance 2017) . We refer to these aspects as *data science for security*, that is, issues in the security domain that can be approached with data science.

Despite the many benefits of data science, there are also some drawbacks and challenges that come with the rapid evolution of the field. The short development life cycles of new methods and products, be it a software, hardware, or a data product, make it difficult to research whether these methods and products are secure or whether they introduce new security problems and flaws. It is therefore not uncommon (Pauli 2017b) that those methods and products have severe security loopholes. Furthermore, due to the increasingly more centralized storage of large amounts of data, cloud infrastructures and big data applications become attractive targets for attackers. As a result of this, the probability that such infrastructures and applications become the target of an advanced targeted attack with the goal of stealing or manipulating large amounts of data is drastically increased. An advanced targeted attack (ATA) or an advanced persistent threat (APT) (Easttom 2016) is an attack where the attackers put a lot of effort, knowledge, and time into getting and eventually also maintaining access to a system or data. Often, such attacks make use of so-called zero-day exploits. These are exploits that are not yet known to the security industry, which means it is unlikely that signature-based systems can detect them. Detection is further complicated in that the attackers try to be as stealthy as possible.

In addition, data science tools such as machine learning and the growing amount of (publicly accessible) data can also be used by cyber criminals to improve their attack methods and strategies. For example, being able to profile people based on their activities on social media and determining what type and style of social engineering attacks makes them do something they do not want to do would be very useful to cyber criminals.

In the following, we discuss the opportunities and risks of data science in more detail. First, we briefly introduce three key concepts of information security: confidentiality, integrity, and availability. Next, we present a brief overview of important topics related to security of data science and provide more details on some key topics that data scientists should consider to make (applications of) data science more secure. Then, we switch to the topic of data science for security, where we discuss examples of applications of data science in security products. This discussion includes a detailed look at the potential and challenges of applying machine learning to the problem of detecting obfuscated JavaScripts. We then conclude the chapter with a summary of topics every (security) data scientist should keep in mind when working in this field.

2 Key Concepts of Information Security

According to the Federal Information Security Management Act of 2002 (2012), the term "information security" means protecting information and information systems from unauthorized access, use, disclosure, disruption, modification, or destruction in order to provide confidentiality, integrity, and availability (CIA):

- *Confidentiality* requires the implementation of authorized restrictions on access and disclosure, which includes measures for protecting personal privacy and proprietary information.
- *Integrity* means guarding against improper modification or destruction, and includes information non-repudiation and authenticity.
- *Availability* finally means ensuring timely and reliable access to and use of information.

For a cloud storage provider for example, confidentiality would mean that data must be stored in encrypted form and that there is a key management scheme in place that makes sure that only authorized entities should be able to decrypt it. In its simplest form, a customer would do the encryption, decryption, and key management in his or her own trusted infrastructure and send and receive encrypted files only. However, this way, the cloud cannot look at the data and functionality such as file indexing and searching. Thus, storing the same file submitted by multiple users only once (de-duplication) cannot be done. To be able to do this, the key must be known to the cloud, which means the cloud should be trusted. To keep the attack surface small, access to the key stored in the cloud must happen on a need-to-know basis and access should be logged. Furthermore, data in transit, when transferred from or to the customer or when moved around in the cloud, should be encrypted as well.

Simply encrypting data is not enough, however, as without integrity protections, the employees of the cloud provider could still modify the encrypted files at the bit and byte level without the customer easily noticing this when decrypting the files and looking at them. And without enough resources to handle peak times or denial-of-service attacks, a customer might be cut off from the data (for some time), which could cause significant financial losses.

Hence, if information infrastructures do not have the desired properties with respect to CIA, they might not work as expected. If these infrastructures are in the big data domain, CIA issues might even be magnified by the velocity, volume, and variety of big data (Cloud Security Alliance 2013b). This will be explored in more detail in the next section.

3 Security of Data Science

In this section, we discuss challenges and solution approaches related to the security of data science methods and applications. Since any data product needs an infrastructure to run on, a piece of software that implements it, data that fuels it, and customers that feel comfortable using it, we provide a brief overview and references to more in-depth material on (1) infrastructure security, (2) software security, (3) data protection, and (4) data anonymization. Furthermore, we discuss (5) examples of attacks on machine learning algorithms.

3.1 Infrastructure Security

Infrastructure security is concerned with securing information systems against physical and virtual intruders, insider threats, and technical failures of the infrastructure itself. As a consequence, some of the more important building blocks to secure an infrastructure are access control, encryption of data at rest and in transit, vulnerability scanning and patching, security monitoring, network segmentation, firewalls, anomaly detection, server hardening, and (endpoint) security policies. Resources such as the NIST special publications series (National Institute of Standards and Technology 2017) or the CIS top 20 security controls (Center for Internet Security 2017) provide guidance and (some) practical advice. However, getting all of this right is far from easy and failing might carry a hefty price tag.

In 2007, for example, Sony was accused of having some serious security vulnerabilities. In an interview, Sony's senior vice president of information security stated: "It's a valid business decision to accept the risk of a security breach. I will not invest $10 million to avoid a possible $1 million loss" (Holmes 2007). The data theft and outage of the PlayStation network in 2011 cost Sony $171 million (Schreier 2011). The Sony Pictures hack in 2014 (Risk Based Security 2014), where personal information of employees were stolen, cost Sony $35 million. Nevertheless, as Sony stated, it is indeed a valid business decision to limit investments in security. But such decisions should be made in full awareness of the value of the assets that are at stake, especially in light of the fact that massive amounts of user accounts or data can pose a very attractive target for cyber criminals: they could steal or destroy it and then ask for a ransom to restore it, they might sell it on the black market, or misuse it to perform other crimes.

The fact that many companies have failed to secure their infrastructure can be considered an anecdotal proof that this is a complex task and should not be done without involving security experts. This is even more true when big data systems are involved, since they might require the implementation of new use-case or product-specific security measures (Moreno et al. 2016). For a checklist of what should be considered when building and securing big data systems, check out the top 100 best

practices in big data security and privacy (Cloud Security Alliance 2016). However, note that many of the best practices also apply to "normal" information systems.

Fortunately, data scientists do rarely have to build and secure an infrastructure from scratch. However, they often have to select, configure, and deploy base technologies and products such as MongoDB, Elasticsearch, or Apache Spark. It is therefore important that data scientists are aware of the security of these products. What are the security mechanisms they offer? Are they secure by default? Can they be configured to be secure or is there a need for additional security measures and tools? Recent events have demonstrated that this is often not the case.

In January 2017, 30,000 MongoDB instances were compromised (Pauli 2017b) because they were configured to accept unauthenticated remote connections. The underlying problem was that MongoDB versions before 2.6.0. were insecure by default. When installed, the installer did not force the user to define a password for the database *admin* account, and the database service listened on all network interfaces for incoming connections, not only the local one. This problem was well known and documented (Matherly 2015), but apparently, many operators of such instances didn't know or didn't care. Just one week later, the same hackers started to attack more than 35,000 Elasticsearch instances with ransomware (Pauli 2017a). Most of these instances were located on Amazon Web Services (AWS) and provided full read and write access without requiring authentication.

It is important to keep in mind that many of these new technologies are designed to facilitate easy experimentation and exploration, and not to provide enterprise-grade security by default. The examples mentioned in the previous paragraph are certainly not the only ones that illustrate this problem. A broader study in the area of NoSQL databases revealed that many products and technologies do not support fundamental security features such as database encryption and secure communication (Sahafizadeh and Nematbakhsh 2015). The general advice here is that before setting up such a technology or product, it is important to check the security features it offers and to verify whether the default configuration is secure enough. If problems are identified, they should be fixed before the product is used.

3.2 Software Security

Software security sets the focus on the methodologies of how applications can be implemented and protected so that they do not have or expose any vulnerabilities. To achieve this, traditional software development life cycle (SDLC) models (Waterfall, Iterative, Agile, etc.) must integrate activities to help discover and reduce vulnerabilities early and effectively and refrain from the common practice to perform security-related activities only toward the end of the SDLC as part of testing. A secure SDLC (SSDLC) ensures that security assurance activities such as security requirements, defining the security architecture, code reviews, and penetration tests, are an integral part of the entire development process.

An important aspect of implementing an SSDLC is to know the threats and how relevant they are for a specific product. This allows prioritizing the activities in the SSDLC. For data products, injection attacks and components that are insecure by default are among the biggest threats. Many data products are based on immature cutting-edge technology. They process data from untrusted sources including data from IoT devices, data from public data sources such as Twitter, and various kinds of user input, to control and use the data product.

For instance, if the code assembles SQL queries by concatenating user input and instructions for the database, this can turn out badly. As an example, consider the following line of code where a SELECT query is built and where *userinput* is provided by the user:

```
String query = "SELECT name, description from Product WHERE name
LIKE '%" + userinput + "%'";
```

If the user (an attacker in this case) specifies the following data as *userinput*,

```
' UNION SELECT username, password FROM User--
```

then the following query is built:

```
SELECT name, description from Product WHERE name LIKE '%' UNION
SELECT username, password FROM User--%'
```

This query is syntactically correct (note that—is used in SQL for comments, which means that the part—*%'* will be ignored by the database system) and will not only return all products, but also all usernames and password that are stored in table *User*.

The solution to this so-called SQL injection problem seems simple: input data must be sanitized so that if the data contains SQL commands, it is just interpreted as textual data and not as a potentially harmful SQL command. Another safeguard to protect from SQL injection is to use only minimal access rights for the technical database user that executes the query. This cannot completely prevent SQL injection, but in case of a vulnerability, it serves as a damage control mechanism to make sure that the amount of data that can be accessed by the attacker is limited.

However, although the mechanisms to prevent SQL injection vulnerabilities are well known, history shows that they are not used consistently in practice—even if incidents related to SQL injection regularly make it into the headlines of mass media. For instance, in 2008, SQL injection was used to steal more than 134 million credit card data records from Heartland Payment Systems (Krebs 2009). Three years later, Global Payment Systems faced the same problem and lost about $92.2 million during the incident (Krebs 2012). Even now, the problem is still around. In 2016, data of 55 million voters were stolen from Comelec, the Philippines Commission on Elections (Estopace 2016), and an SQL injection vulnerability might also have played an important role in the incident of the Panama Papers (Burgees and

Temperton 2016), where 11.5 million financial documents about offshore entities were leaked.

Clearly, SQL might not see widespread use in big data systems. New technologies such as NoSQL databases are far more prominent. However, their security history does not look much better, as a recent paper demonstrated similar issues with injection attacks as SQL (Ron et al. 2016).

One reason why it is difficult to get rid of such vulnerabilities is that preventive measures have to be considered by the developers and integrated into the code. If they are not aware of such risks and security is not a crucial part of the SDLC they are employing, it is very likely that vulnerabilities creep into the code because countermeasures are missing completely or are implemented incorrectly. There exists also no magic bullet in the sense of tools or formal proofs that can easily verify whether a piece of software is secure, although there exist tools that can detect some vulnerabilities. A good overview in this context is provided in (Software Testing Help 2017).

In general, the following steps help to address common software security problems when building a (software) product:

- Make sure that third party technology or products used are as mature as possible.
- Make sure that third party technology or products used offer a broad spectrum of security features and access controls options.
- Make sure that you have an SSDLC in place.

A good starting point to learn more about how to develop secure software are the SSDLC models of Microsoft (Microsoft 2017b) and the Open Web Application Security Project OWASP (OWASP 2017a). For more specific advice on what to consider when developing web services and web applications, OWASP (2017b) or Li and Xue (2014) offer well-suited sources. OWASP (2017b) lists the top 10 (web-) application security risks and provides technical guidance on how to test for them and how to avoid them. Five important takeaways from there are that developers should check their web applications and services for the following problems:

- Incorrect or lack of input validation and data sanitation so that an attacker can trick an interpreter or query engine to do things that were not intended.
- Incorrect implementation of authentication and session management.
- Exposure of sensitive data because of problems like (1) insufficient or missing data encryption at rest and in motion, (2) password stores that do not use strong adaptive and salted hashing functions with a work factor (e.g., PBKDF2[1] or bcrypt[2]), or data leakage in log files.
- Incorrect implementation of the mechanisms to restrict what an authenticated user is allowed to do. For example, checks whether a user has the right permissions to execute an action might be done for all actions that a user can trigger via URL

[1] https://tools.ietf.org/html/rfc2898#page-9

[2] https://www.usenix.org/legacy/events/usenix99/provos/provos_html/node5.html

entries that are exposed in the web-interface—but not for actions that could be triggered by accessing portions of a website that are not exposed by such entries (forceful browsing).
- Use of insecure configurations as a result of insecure default configurations, incomplete or ad hoc configurations, outdated configurations, open cloud storage, misconfigured HTTP headers, verbose error messages containing sensitive information, or other root causes.

3.3 Data Protection

A core activity in data science is the processing of (large amounts of) data. For most processing tasks, the data must be available in unencrypted form. This has two main drawbacks. The first one is that when security measures such as access control fail, attackers can easily steal the data and make use of any information it contains. To make this more difficult, the data should always be stored in encrypted form. This way, the attacker must steal the data when it is being processed or manage to steal the keys used to encrypt it.

The second drawback is that the vast amount of processing power available in data centers around the world cannot be exploited if the data contains confidential information or is subject to data protection laws prohibiting the processing by (foreign) third parties. For such cases, it would have to be possible to do the processing in the encrypted space. Searchable encryption and homomorphic encryption (Prasanna and Akki 2015) offer interesting properties with this regard.

Searchable encryption (SE) introduced by Song et al. (2000) [see Bösch et al. (2014)] for an overview of different approaches) can be divided into many different subgroups. The core logic mostly consists of building an encrypted keyword search index on the client side. A search is then performed using trapdoor functions. A trapdoor function is a function that is easy to compute in one direction, but that is difficult to compute in the inverse direction unless one knows a secret. The most basic algorithms allow only queries with a single keyword and have performance issues when new data is added. If data is frequently modified, removed, or added, dynamic data search algorithms are required. Fuzzy-keyword search extends the algorithm to tolerate (some) spelling mistakes. There are also methods that support multiple keywords per query. SE offers methods to perform ranked search, for example, by taking the access history of a user and the access frequency into account. Although some research prototypes have been developed and partly also made available for general use and experimentation [e.g., Popa et al. (2011)], several limitations must be overcome before SE can be used widely in practice. One of these limitations is that algorithms based on secret (symmetric) key cryptography usually require a key exchange over a secured channel and offer only limited search capabilities compared to traditional search engines. Another one is that public key cryptography-based approaches are insufficient for modern big data systems because of substantial computational overhead.

Homomorphic encryption (HE) addresses the challenge to perform general computations on encrypted data. HE allows performing simple operations such as additions, multiplications, or quadratic formulas on ciphertext. It generates an encrypted result, which when decrypted, delivers the same result as if the operations were performed on the plaintext. This offers the ability to run calculations on untrusted devices without giving up on data privacy. Craig Gentry (2009) described the first Fully Homomorphic Encryption (FHE) scheme. This scheme allows performing any desirable function on encrypted data. Unfortunately, FHE is currently far away from practical use, as it increases memory consumption and processing times of even basic operations by about 6–7 orders of magnitude (Brown 2017). Therefore, Somewhat Homomorphic Encryption (SwHE) techniques are proposed. Compared to FHE, they provide better efficiency but do not support all operations [see, e.g., Gentry et al. (2012)]. On the implementation side, there are some HE research prototypes available such as by Halevi (2017). However, given the current state of HE technology, it is expected that several years of further research are required before HE is ready for productive use.

3.4 Privacy Preservation/Data Anonymization

In many cases, data science analyzes data of human individuals, for instance, health data. Due to legal and ethical obligations, such data should be anonymized to make sure the privacy of the individuals is protected. Data anonymization basically means that any data record in the data set should not be easily linkable to a particular individual. Obvious solutions include stripping the real name or the detailed address of individuals from the records, but experience teaches that this is usually not enough to truly anonymize the data.

For instance, in 2006, Netflix started an open competition with the goal to find algorithms that allow predicting user ratings for films. As a basis, Netflix provided a large data set of user ratings as training data, where both users and movies were replaced by numerical IDs. By correlating this data with ratings from the Internet Movie Database, two researchers demonstrated that it is possible to de-anonymize users (Narayanan and Shmatikov 2008). Another example is the Personal Genome Project, where researchers managed to de-anonymize about 90% of all participants (Sweeney et al. 2013). Their basic approach was to link information in the data records (birth date, gender, and ZIP code) with purchased voter registration lists and other publicly available information.

To overcome these issues, a more scientific approach toward anonymization is required. The question is the following: Is it possible to modify data such that the privacy of the participants is fully protected without losing the essence of the data and therefore its utility? In this context, “privacy protection” means that an attacker should not be able to learn any additional information about the individuals than what is directly provided by the data records, even when this data is correlated with other information. Past and more recent research activities have provided several

approaches that can help to achieve this, including generalization (Sweeney 1997) and suppression (Cox 1980), *k*-anonymity (Samarati and Sweeney 1998), and differential privacy (Dwork 2006). Each method has its advantages and drawbacks.

Suppression is a basic form of trying to achieve anonymity by either deleting attributes or substituting them with other values. Generalization describes the approach to blur data by replacing specific values with categories or ranges of values. An attribute containing the age of a person is then translated to a range, so 33 may result in 30–39. Combining these methods can lead to *k*-anonymity, which means that each record cannot be distinguished from at least $k - 1$ other records when considering the personally identifying information in the records.

As an example, assume that a data set includes data records of individual. Each record includes gender, age range, and disease from which the person is suffering. Assume there are three records with gender female and age range 50–59. This basically corresponds to 3-anonymity, as these three records cannot be distinguished from one another based on the attributes gender and age range. *k*-anonymity also has its limitations, especially if the diversity of the non-anonymized attributes is low. In the previous example, let us assume that the disease is heart-related in all three cases. This implies that if an attacker knows that Angela, who is 55 years old, is included in the data set, then he directly knows that she is suffering from heart-related health problems, as all female persons between 50 and 59 in the data set are suffering from it.

The basic idea of differential privacy is that the actual values of the attributes of any single record in the data set should only have a very limited effect on the outcome of any analysis performed on the data. If this is the case, an attacker, when querying the data set, cannot learn anything about a specific individual in the data set as the received outcome is possibly independent of the actual attributes of this individual. This is basically achieved by adding some noise to the result before it is presented to the analyst. For example, let us assume that there are 100 records of 100 persons in a data set and the attacker knows of 99 persons whether they have a heart-related disease or not (we assume that 33 of them have such an issue), but he doesn't know this of the remaining person, which we name Alice. If the attacker performs the query "how many persons have a heart-related disease," then he directly knows Alice's condition: If the query returns 33, Alice has no heart-related problem, if it returns 34, Alice has a heart-related issue. When using differential privacy, the query would not return the actual value, but it would distort it a little bit, that is, the query would return a value in the neighborhood of 33 or 34, such as 30, 32, or 35. What's important is that the returned value does not indicate whether the true value is 33 or 34, which implies the attacker cannot learn anything about Alice's condition.

Obviously, any data anonymization method has its price as it has a negative impact on the quality of the data and the precision of the results when doing data analysis. Suppressing and generalizing data removes information, which means that the results of any analysis performed on the data will become less precise. And in the case of differential privacy, we usually get results that are "close to the correct result," but that usually do not correspond to the exact result. But this is the price of

protecting the privacy of the involved individuals and this also implies that in practice, it is important to carefully balance the privacy of the individuals and the required precision of the analyses. A concise overview about anonymization methods is given by Selvi and Pushpa (2015). Detailed information about privacy-preserving data publishing and corresponding research can be found in the survey by Fung et al. (2010).

3.5 Machine Learning Under Attack

The combination of sophisticated algorithms and untrusted data can open the door for different kinds of attacks. In the 2010 Flash Crash (Kirilenko et al. 2017), the New York Stock Exchange experienced a temporary market loss of one trillion dollar caused by market manipulations. The trader Navinder Singh Sarao rapidly placed and canceled orders automatically so that high-frequency trading firms interpreted the signs incorrectly. In the beginning, they bought the spoof orders and absorbed the sell pressure. Few minutes later, these long-term positions were forcefully sold leading to a feedback loop. In times of big data, trading algorithms often take news feeds like business publications, SEC filings, and Twitter posts into account to make predictions. In 2013, this led to a loss of $130 billion in stock value due to a fake Twitter message from the associated press about an explosion in the White House (Foster 2013).

Mozaffari-Kermani et al. (2015) propose a method to generate data, which, when added to the training set, causes the machine learning algorithms to deliver wrong predictions for specific queries. Thus, this method could, for example, be used to compromise the effectiveness of a system to diagnose cancer or to identify anomalies in computer networks. Their method consists of two algorithms. The first one creates data sets that statistically belong to the attacked class but are labeled like the target class to which a bias should be created. The second algorithm then evaluates which data set has the highest impact on the degradation of the model. For their method to work well, the attacker must know the statistics of the training data set, the feature extraction process, and the machine learning algorithms used. However, the only true requirement is knowledge on the feature extraction process that maps a sample onto a feature vector. If the training set is not public or based on publicly available data and cannot be stolen, an attacker could construct a proxy training data set by querying the predictor with artificial test instances and by observing its responses (Nelson et al. 2008). And if the machine learning algorithm is not known, their approach can be modified to cope with this case at the cost of some of its effectiveness. A good countermeasure to this kind of attack is the use of a threshold value for the returned accuracy metrics. At first, one might think that because an attacker must be able to add training data to the training set, this poisoning attack is rather impractical. However, in many cases, the training data and/or its labels do come from untrusted sources or can at least be influenced by them. And even if the sources

are trusted, consider that an attacker might have hacked one or multiple of those sources because they were easier to hack than the system with the data product itself.

In recent years, machine-learning-as-a-service (MLaaS) has become a huge trend. Tech giants such as Amazon Web Services (2017), Google (2017), Microsoft (2017a), and many others offer customers to create and run machine learning algorithms in the cloud, offering different services like facial recognition and natural language processing. Some of these publicly accessible tools may contain sensitive data within their model that has to be protected. Fredrikson et al. (2015) show how confidential information of a machine learning model can be extracted by inverting it. The authors are able to reconstruct the images of the training data of a facial recognition system. For each image submitted, the system responds with a list of names together with their confidence value. This allows an attacker to treat it as an optimization problem finding the input that maximizes the confidence of a target class. The time for reconstruction depends on the model and varies between 1.4 s and 21 min. The attack is also applicable if nothing about the model is known (black-box) but takes significantly longer. Tramèr et al. (2016) improve the computation time by starting with stealing the model using prediction APIs and then running the inversion attack on the copy of the model. They further show how decision trees, logistic regression-based classifiers, and neural networks can be stolen by just using the provided interfaces and the rich information returned by MLaaS solutions.

4 Data Science for Security

After having discussed some of the security challenges a data scientist might face when developing and using modern data science technologies, this section deals with the opportunities of using data science to help solve major challenges in information security. In this context, we are looking at three general application areas: (1) anomaly detection, (2) malware detection and classification, and (3) threat detection. In the next chapter, we are going to take a more detailed look at a specific case study where machine learning was applied to detect obfuscated JavaScript code.

4.1 Anomaly Detection

The detection of anomalies is a major challenge in information security and has many applications such as network intrusion detection, credit card fraud detection, insurance claim fraud detection, insider trading detection, and many others. An anomaly describes a single point or a set of data points within a large data set that does not match the normal or usual behavior. In a network intrusion detection system, this could be a large amount of login attempts or an attacker who scans systems for open ports to get information about a targeted infrastructure. In credit card fraud detection, this could be an anomalous transaction over a significantly

larger amount than what is usually spent by the credit card holder. Another example is using the credit card in a different context than usual, for instance in a different country. In credit card fraud detection, this is a serious challenge due to the vastly increased amount of online transactions that are difficult to assign to specific locations. A third example of anomalous credit card usage would be a huge amount of small transactions in a short time.

A broader overview about this topic and the performance of different machine learning algorithms for anomaly detection is given in the survey by Chandola et al. (2009). They show how machine learning can contribute to solve different anomaly detection-based challenges. Their core conclusion is that there is currently no "one size fits all" solution. Nearest neighbor and clustering-based techniques suffer when data is high dimensional, because the distance measures cannot differentiate between normal and abnormal behavior anymore. Classification-based algorithms deliver good results but labeled data is often rare. Mahmud et al. (2016) give an overview of machine learning algorithms and their performance in credit card fraud detection. They achieve a classification accuracy of 98.25%, but the fraud detection success rate is below 50% because the fraction of fraudulent credit card transactions in the data set they used was small. According to the results, the highest detection rate is achieved using RotationForest, KStar, and RandomTree models. Finally, Gulenko et al. (2016) have evaluated machine learning algorithms for anomaly detection in cloud infrastructures. They come to the conclusion that high precision and recall rates can be achieved but the models suffer from aging effects. Therefore, models have to be periodically retrained and updated. Specific answers about the required periodicity are not given, however, and left open as future research.

The class imbalance problem that Mahmud et al. (2016) faced when they developed their credit card fraud detection system is fairly common in anomaly detection: the number of normal items, events, or observations is usually much larger than those of anomalous ones. If this imbalance in the distribution of the normal and the abnormal class(es) is not taken into account, a detector might perform poorly. Two examples where this imbalance tends to be quite strong are credit card fraud detection and network intrusion detection. Pozzolo et al. (2015) work with a data set with credit card transactions from European cardholders in September 2013. This data set has only 492 cases of fraud in the total of 2,84,807 transactions. Shiravi et al. (2012) present a reference data set (the ISCX data set) for validating network intrusion detection systems where, according to Soheily-Khah et al. (2017), attack traffic accounts for only 2% of the overall traffic. While 2% is quite low, it might easily be much lower, for example 0.01%, as in the application layer denial-of-service data set of Viegas et al. (2017).

Fortunately, many techniques exist to handle such imbalanced class distributions. One way to address the problem is to resample the training data to turn it into a more balanced data set. In the example with the credit card transaction data mentioned before, Pozzolo et al. (2015) performed a study on the impact of undersampling on classification accuracy and probability calibration. They found that randomly selecting and removing legitimate transactions to get a more balanced data set can indeed increase classification accuracy. However, for some of the other data sets they

used, this was not the case. An overview of this very active area of research—mainly with focus on binary classification—can be found in Branco et al. (2016).

Another way to approach the problem is to make use of machine learning algorithms that can cope better with (strongly) imbalanced class distributions. Traditional methods like support vector machines or decision trees have a bias toward the majority class since "... rules that correctly predict those instances are positively weighted in favor of the accuracy metric, whereas specific rules that predict examples from the minority class are usually ignored (treating them as noise), because more general rules are preferred. In such a way, minority class instances are more often misclassified than those from the other classes" (Galar et al. 2012). Or in other words, if a credit card fraud detector would classify all transactions as not fraudulent, the classifier could achieve 99% accuracy for a data set where 1% of the transactions are fraudulent.

According to Krawczyk (2016), the most popular branch of machine learning algorithms that aims at addressing this problem is cost-sensitive approaches where learners are modified to incorporate penalties for (some) classes. "This way by assigning a higher cost to less represented set of objects we boost its importance during the learning process (which should aim at minimizing the global cost associated with mistakes)" (Krawczyk 2016). However, for most of these approaches, profound theoretical insights into why and how well they perform with arbitrary imbalanced data sets is lacking. An overview over related work on this topic can be found in Branco et al. (2016) or Galar et al. (2012).

The most important takeaway from this discussion is that one should be aware of the imbalance problem when developing anomaly detection solutions.

A good starting point for a more in-depth study is Branco et al. (2016) and/or Krawczyk (2016). Furthermore, another takeaway is that retraining is an overall important task in anomaly detection as the normal behavior defined in the past will usually not sufficiently represent the future. This question is also addressed in general in novelty detection, which is the task of classifying data that differ in some respect from the data that are available during training [Pimentel et al. (2014)].

4.2 Malware Detection and Classification

In the past few years, hundreds of millions of new and unique malware samples have been found every year. However, most of these samples are very similar and belong to a few thousand malware families only (Check Point 2016). One of the reasons for this is that today, most malware authors modify and/or obfuscate their malware on a per victim basis. This way, they can evade simple signature-based antivirus scanners. To mitigate this problem, samples from known families should be recognized and filtered, and only new ones or samples that are "different enough" should have to be analyzed by malware analysts (if at all). Machine learning and big data seem to be capable solutions to handle such a large amount of continuously evolving data and to perform highly accurate classification tasks on it. For example, the next-generation

antivirus software from Cylance makes use of "... data-mining techniques to identify the broadest possible set of characteristics of a file. These characteristics can be as basic as the file size or the compiler used and as complex as a review of the first logic leap in the binary" (Cylance 2017). They claim to extract the uniquely atomic characteristics of the file depending on its type (*.exe*, *.dll*, *.com*, *.pdf*, *.doc*, etc.).

The importance of this task for the research and anti-malware industry was stressed by the fact that in 2015, Microsoft (2015) launched a contest to get new inputs on how to do the classification of malware samples into malware families from the community. The contestants were given a labeled training and a test data set, each consisting of 10,000 samples from nine different malware families. The results of this contest suggested that this task can be solved with very low multiclass loss (around 0.003). However, to achieve this, the contestants had data such as the assembly code of the binaries, which is difficult to extract without using dynamic code analysis. Furthermore, modern malware hides its true nature and unpacks or decrypts its malicious code only when run outside of an analysis environment. This and scalability problems when having to run all suspicious binaries make approaches based on dynamic code analysis less attractive than those based on static analysis.

Static code analysis describes all information about an application that can be gained without running it. On Android systems, this is usually the *apk* file, where security-relevant information such as API calls and even the source code itself can easily be accessed. This is good news since G DATA (2016) reported an average of 9468 new malicious applications for Android per day during the first half of 2016. It seems that due to their increased usage for mobile payment, mobile ticketing, and many other business cases, mobile devices became a very attractive target for cyber criminals.

Tam et al. (2017) provide a comprehensive overview of the challenges encountered when trying to detect and classify malicious Android applications. The authors find that in 2012, popular antivirus systems had a detection rate from around 20–79.6%. In all cases, complex malware was not detected. In particular, the systems often failed when the malware was obfuscated or when malicious Java code was executed after it was dynamically loaded during runtime. They show that new approaches from the data science domain can (easily) surpass traditional ones. This is confirmed by Arp et al. (2014), where the proposed DREBIN method achieves a detection rate of 97% with a low false-positive rate by combining statistical analysis with support vector machines.

On platforms where the source code is not easily available, static analysis gets more difficult. Narayanan et al. (2016) assess the performance of different machine learning and pattern recognition algorithms on imaginary representations of malware binaries. They find that samples from the same families result in a similar image texture. With this approach, it was possible to achieve results that were nearly as good as those of the winners of the Microsoft contest, but without having to extract the assembly code of the malware.

4.3 Threat Detection

Security information and event management (SIEM) (Zuech et al. 2015) technology supports threat detection and security incident response through the (real-time) collection and historical analysis of security events from a wide variety of events and contextual data sources. Such events might include failed and successful authentication attempts, the number of packets dropped by a firewall, or a report by an antivirus program about the identification and blocking of a malicious file.

In 2016, the security operations center of IBM recorded more than 20 billion security events per day (IBM 2016). This is still quite moderate when compared to the numbers from fortune 100 telecommunication providers, which can face up to one million events per second and up to 85 billion events per day (IBM 2013). Traditional SIEM solutions relying on structured databases and (mostly) manual definition of what is normal and malicious and/or abnormal behavior have difficulties scaling up to these large amounts of data.

The use of big data solutions and machine learning is therefore the next logical step in the evolution of such systems. Technologies such as Apache Hadoop and Apache Spark offer fast and scalable methods to analyze vast amount of data. According to Dark Reading (2012),

> in an environment where its security systems generate 3 terabytes of data a week, just loading the previous day's logs into the system can [. . .] take a full day

and

> searching among a month's load of logs could take anywhere between 20 minutes to an hour [. . .]. In our environment within HIVE, it has been more like a minute to get the same deal.

This is why companies such as HP and IBM put a lot of effort into the development of systems using new data science technologies (IBM 2013). However, determining which events are related to activities that are harmless, for example because they stem from an attack that failed, and which are related to real threats, is a challenging problem. In a large-scale experiment from HP, which had the goal to identify malicious domains and infected hosts, more than 3 billion HTTP and DNS Requests where collected from 900 enterprises around the world. They showed that high true-positive rates are possible using decision trees and support vector machines with a very limited amount of labeled data by simultaneously keeping the false-positive rates low (Cloud Security Alliance 2013a). Another work demonstrates the usage of a system called Beehive, which analyzed around 1 billion log messages in an hour and successfully detected violations and infections that were otherwise not noticed (Yen et al. 2013).

5 Case Study: Detecting Obfuscated JavaScripts

To demonstrate the potential and the challenges of applying machine learning to detect malware, this section describes in more detail the results of a recent analysis that was done by Tellenbach et al. (2016).

JavaScript is a common attack vector to probe for known vulnerabilities and subsequently to select a fitting exploit or to manipulate the Document Object Model (DOM) of a web page in a harmful way. The JavaScripts used in such attacks are often obfuscated to make them hard to detect using signature-based approaches. On the other hand, since the only legitimate reason to obfuscate a script is to protect intellectual property, there are not many scripts that are both benign and obfuscated. A detector that can reliably detect obfuscated JavaScripts would therefore be a valuable tool in fighting JavaScript-based attacks.

To evaluate the feasibility and accuracy of distinguishing between different classes of JavaScript code, a classic machine learning approach was used. In the first step, a data set was collected that contains JavaScripts and correct labels (obfuscated or non-obfuscated). Next, 45 features were selected and extracted from the JavaScripts in the data set. These features capture various aspects such as frequency of certain keywords, number of lines, characters per line, number of functions, entropy, and more. Based on this, several classification algorithms were trained and evaluated. The following sample code was used to make visitors of a hacked website connect to a server hosting the CrimePack exploit kit (Krebs 2017). The script is obfuscated to hide the fact that it injects an iframe and to obfuscate the URL it connects to:

```
tmssqrcaizo = "WYTUHYjE3cWYTUHYjE69WYTUHYjE66";
var makvvxmaqgh = "WYTUHYjE72";
var nlsysoyxklj =
"WYTUHYjE61WYTUHYjE6dWYTUHYjE65WYTUHYjE20WYTUHYjE6eWYTUHYjE61WYTU
HYjE6dWYT
UHYjE65WYTUHYjE3dWYTUHYjE22";
var zezugacgoqg =
"WYTUHYjE6eWYTUHYjE6fWYTUHYjE6aWYTUHYjE72WYTUHYjE73WYTUHYjE65WYTU
HYjE72WYT
UHYjE66WYTUHYjE6cWYTUHYjE72WYTUHYjE6f";
var nmcwycmeknp =
"WYTUHYjE22WYTUHYjE20WYTUHYjE77WYTUHYjE69WYTUHYjE64WYTUHYjE74WYTU
HYjE68WYT

(not shown)

var vbvvhagnggg = new Array();
vbvvhagnggg[0] = new Array(
tmssqrcaizo +
makvvxmaqgh +
nlsysoyxklj +

(not shown)
```

```
xmzvkbtpiof);
document[
  "WYTUHYjEwWYTUHYjErWYTUHYjEiWYTUHYjEtWYTUHYjEeWYTUHYjE".replace(
      /WYTUHYjE/g,
      ""
  )
](
  window[
"WYTUHYjEuWYTUHYjEnWYTUHYjEeWYTUHYjEsWYTUHYjEcWYTUHYjEaWYTUHYjEpW
YTUHYjEeWYTUHYjE".
replace(
  /WYTUHYjE/g,
  ""
  )
  ](vbvvhagnggg.toString().replace(/WYTUHYjE/g, "%"))
);
```

The script below is the unobfuscated version of the above script (URL is not the original one). The de-obfuscated code is significantly easier for a security researcher or any programmer to analyze:

```
document.write(
'<iframe name="nojrserflro" width="1" height="0"
src="http://localhost/index.php" marginwidth="1" marginheight="0"
title="nojrserflro" scrolling="no" border="0" frameborder="0"></
iframe>'
);
```

In general, there are many different ways how a script can be made hard to read and understand.

To collect the non-obfuscated samples in the data set, JavaScripts were extracted from the jsDelivr[3] content delivery network, which contains many JavaScript libraries) and the Alexa[4] Top 5000 websites. This resulted in 25,837 non-obfuscated samples, which includes both regular JavaScripts (as they have been written by the developers) and minified JavaScripts (where whitespace have been removed and function- and variable names have been shortened to reduce the overall size). To collect the obfuscated samples in the data set, two different strategies were used. First, a set of truly malicious (and obfuscated) JavaScript samples was received from the Swiss Reporting and Analysis Centre for Information Assurance MELANI.[5] However, this consisted of only 2706 samples. Therefore, additional obfuscated samples were synthetically generated by obfuscating the non-minified JavaScripts from the collected non-obfuscated samples. For this, six different, publicly available

[3] https://www.jsdelivr.com
[4] http://www.alexa.com
[5] http://www.melani.admin.ch

obfuscators were used, which delivered an additional 73,431 samples. Overall, this resulted in 76,137 obfuscated samples.

Based on this data set, the best classification performance could be achieved with a boosted decision tree classifier. With this classifier, only 1.03% of the non-obfuscated scripts were classified as obfuscated (false positives) and only 0.32% obfuscated scripts were classified as non-obfuscated (false negatives). Overall, boosted decision tree was the only classifier that achieved F1-scores above 99% for both classifying obfuscated and non-obfuscated JavaScripts, demonstrating that machine learning works well on this task.

Next, it was analyzed how well classification works to detect obfuscated JavaScripts if the corresponding obfuscator is not used for any of the JavaScripts that are used to train the classifier. The purpose of this analysis was to get an understanding about how well the classifier can "learn about obfuscation in general." The results of this analysis varied greatly depending on the specific obfuscator left out from the training set. For one obfuscator, the F1-score remained almost the same. For the other obfuscators, the F1-score was impacted by a few percent up to almost 100%. Finally, it was analyzed how well the truly malicious JavaScripts can be detected if the training set only includes the non-obfuscated and the synthetically generated obfuscated JavaScripts. In this case, less than 50% of the malicious JavaScripts were classified correctly as obfuscated.

This case study exhibits several interesting results and provides some lessons learned when using machine learning to detect malware or malicious activity in general:

- In general, classifying obfuscated and non-obfuscated JavaScripts works well, provided that the obfuscators used for the obfuscated JavaScripts are also represented in the data set used to train the classifier.
- Detecting obfuscated JavaScripts that use obfuscators not represented in the training set is more difficult. While this might be improved somewhat by using better-suited features, it clearly demonstrates that it is paramount to include samples that use a wide range of obfuscators in the data set so the classifier can learn a wide range of properties employed by different obfuscators. Generalizing this to other scenarios indicates that it is important to use representative malicious samples, whether it is actual malware or malicious activity in general.
- This directly leads to another challenge: It is difficult to get a large number of truly malicious samples. This is not only the case for malicious JavaScripts, but in general for "samples" that capture malicious behavior (e.g., network or system intrusions). While creating synthetic malicious samples may help to a certain degree, this has its limitations as it can usually not capture the full range of truly malicious samples, as demonstrated by the final analysis described above.

6 Conclusions and Lessons Learned

With respect to security, data science is a double-edged sword. On the one side, it offers many new opportunities and a lot of potential to significantly improve traditional security algorithms and solutions. Recent advances in challenging domains such as anomaly detection, malware detection, and threat detection underline the tremendous potential of security data science.

On the other side, it comes with many challenges. Most of them, including questions related to infrastructure and software security, can be addressed with the following practices:

- Protect your information system with suitable security controls. Get an idea of the complexity of the topic by checking out guides like the CIS top 20 security controls (Center for Internet Security 2017) and consult with or hire experts to protect your infrastructure.
- Implement an SSDLC to make sure that the software and services you develop are as secure as possible and that they remain secure.
- Check out the top security problems related to a specific technology or service. For example, the OWASP top 10 (OWASP 2017b) for web applications and services.
- Study the default configuration and all of the configuration options of a component to avoid insecure configurations.
- Keep in mind that anonymization is not perfect; whenever data privacy is critical, one has to choose the anonymization method with care and balance the privacy of the individuals and the required precision of the analyses.
- Check whether your system is susceptible to any of the various ways attackers might try to exploit data-driven applications (data poisoning, model extraction, etc.).

Nevertheless, recent incidents show that these practices are not widely used yet. One of the reasons is that today's security measures for data science heavily rely on security by afterthought, which is not acceptable as security aspects have to be considered during all steps of the development and product and data life cycle.

Other challenges require more research before they can be widely adopted, including questions related to perform computations on encrypted or anonymized data.

References

Amazon Web Services. (2017). *Artificial intelligence on AWS*. Retrieved from https://aws.amazon.com/amazon-ai/

Arp, D., Spreitzenbarth, M., Gascon, H., & Rieck, K. (2014). DREBIN: Effective and explainable detection of android malware in your pocket. *Presented at the 21st Annual Network and*

Distributed System Security Symposium (NDSS). Retrieved from http://dblp.uni-trier.de/db/conf/ndss/ndss2014.html#ArpSHGR14

Bösch, C. T., Hartel, P. H., Jonker, W., & Peter, A. (2014). A survey of provably secure searchable encryption. *ACM Computing Surveys, 47*(2), 18:1–18:51.

Branco, P., Torgo, L., & Ribeiro, R. P. (2016). A survey of predictive modeling on imbalanced domains. *ACM Computing Surveys, 49*(2), 31:1–31:50. https://doi.org/10.1145/2907070.

Brown, B. (2017). *How to make fully homomorphic encryption "practical and usable"*. NETWORKWORLD. Retrieved from https://www.networkworld.com/article/3196121/security/how-to-make-fully-homomorphic-encryption-practical-and-usable.html

Burgees, M., & Temperton, J. (2016). The security flaws at the heart of the Panama Papers. *WIRED Magazine*. Retrieved from http://www.wired.co.uk/article/panama-papers-mossack-fonseca-website-security-problems

Center for Internet Security. (2017). *CIS 20 security controls*. Retrieved from https://www.cisecurity.org/controls

Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing Surveys, 41*(3), 15:1–15:58.

Check Point. (2016). *Check Point research shows surge in active malware families during first half of 2016*. Retrieved from https://www.checkpoint.com/press/2016/check-point-research-shows-surge-active-malware-families-first-half-2016/

Cloud Security Alliance. (2013a). *Big data analytics for security intelligence*. Retrieved from https://downloads.cloudsecurityalliance.org/initiatives/bdwg/Big_Data_Analytics_for_Security_Intelligence.pdf

Cloud Security Alliance. (2013b). *Expanded top ten big data security and privacy challenges*. Retrieved from https://downloads.cloudsecurityalliance.org/initiatives/bdwg/Expanded_Top_Ten_Big_Data_Security_and_Privacy_Challenges.pdf

Cloud Security Alliance. (2016). *Security and privacy handbook: 100 best practices in big data security and privacy*. Retrieved from https://downloads.cloudsecurityalliance.org/assets/research/big-data/BigData_Security_and_Privacy_Handbook.pdf

Cox, L. H. (1980). Suppression methodology and statistical disclosure control. *Journal of the American Statistical Association, 75*(370), 377–385.

Cylance. (2017). *Math vs. Malware (white paper)*. Retrieved from https://www.cylance.com/content/dam/cylance/pdfs/white_papers/MathvsMalware.pdf

Dark Reading. (2012). *A case study in security big data analysis*. Retrieved from https://www.darkreading.com/analytics/security-monitoring/a-case-study-in-security-big-data-analysis/d/d-id/1137299

Dwork, C. (2006). Differential privacy. In *Proceedings of the 33rd International Conference on Automata, Languages and Programming – Volume Part II* (pp. 1–12). Berlin: Springer.

Easttom, W., II. (2016). *Computer security fundamentals*. Pearson Education.

Estopace, E. (2016). *Massive data breach exposes all Philippines voters*. Retrieved from https://www.telecomasia.net/content/massive-data-breach-exposes-all-philippines-voters

Foster, P. (2013). "Bogus" AP tweet about explosion at the White House wipes billions off US markets. *The Telegraph*. Retrieved from http://www.telegraph.co.uk/finance/markets/10013768/Bogus-AP-tweet-about-explosion-at-the-White-House-wipes-billions-off-US-markets.html

Fredrikson, M., Jha, S., & Ristenpart, T. (2015). Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security* (pp. 1322–1333). New York, NY: ACM.

Fung, B. C. M., Wang, K., Chen, R., & Yu, P. S. (2010). Privacy-preserving data publishing: A survey of recent developments. *ACM Computing Surveys, 42*(4), 14:1–14:53.

G DATA. (2016). *New ransomware threatens Android devices*. Retrieved from https://www.gdatasoftware.com/news/2016/07/28925-new-ransomware-threatens-android-devices

Galar, M., Fernandez, A., Barrenechea, E., Bustince, H., & Herrera, F. (2012). A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches.

IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 42 (4), 463–484.

Gentry, C. (2009). Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing* (pp. 169–178). New York, NY: ACM.

Gentry, C., Halevi, S., & Smart, N. P. (2012). Homomorphic evaluation of the AES circuit. In R. Safavi-Naini, & R. Canetti (Eds.), *Advances in Cryptology – CRYPTO 2012: 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2012. Proceedings* (pp. 850–867). Berlin: Springer.

Google. (2017). *Google cloud machine learning engine*. Retrieved from https://cloud.google.com/ml-engine/

Gulenko, A., Wallschläger, M., Schmidt, F., Kao, O., & Liu, F. (2016). Evaluating machine learning algorithms for anomaly detection in clouds. In *2016 IEEE International Conference on Big Data (Big Data)* (pp. 2716–2721). doi:https://doi.org/10.1109/BigData.2016.7840917

Halevi, S. (2017). *HElib: An implementation of homomorphic encryption*. Retrieved from https://github.com/shaih/HElib

Holmes, A. (2007). *Your guide to good-enough compliance*. CIO. Retrieved from https://www.cio.com/article/2439324/risk-management/your-guide-to-good-enough-compliance.html

IBM. (2013). *Extending security intelligence with big data solutions*. IBM. Retrieved from https://www-01.ibm.com/common/ssi/cgi-bin/ssialias?htmlfid=WGW03020USEN

IBM. (2016). *IBM announces new national cyber security centre in Canberra*. Retrieved from http://www-03.ibm.com/press/au/en/pressrelease/50069.wss

Kirilenko, A., Kyle, A. S., Samadi, M., & Tuzun, T. (2017). The Flash Crash: High-frequency trading in an electronic market. *The Journal of Finance, 72*(3), 967–998.

Krawczyk, B. (2016). Learning from imbalanced data: Open challenges and future directions. *Progress in Artificial Intelligence, 5*(4), 221–232.

Krebs, B. (2009). *Payment processor breach may be largest ever*. Retrieved from http://voices.washingtonpost.com/securityfix/2009/01/payment_processor_breach_may_b.html?hpid=topnews

Krebs, B. (2012). *Global Payments breach window expands*. Retrieved from https://krebsonsecurity.com/2012/05/global-payments-breach-window-expands/

Krebs, B. (2017). *Crimepack: Packed with hard lessons*. Retrieved from https://krebsonsecurity.com/2010/08/crimepack-packed-with-hard-lessons/

Li, X., & Xue, Y. (2014). A survey on server-side approaches to securing web applications. *ACM Computing Surveys, 46*(4), 54:1–54:29.

Mahmud, M. S., Meesad, P., & Sodsee, S. (2016). An evaluation of computational intelligence in credit card fraud detection. In *2016 International Computer Science and Engineering Conference (ICSEC)* (pp. 1–6). doi:https://doi.org/10.1109/ICSEC.2016.7859947

Matherly, J. (2015). *It's the data, stupid!* Retrieved from https://blog.shodan.io/its-the-data-stupid/

Microsoft. (2015). *Microsoft malware classification challenge*. Retrieved from https://www.kaggle.com/c/malware-classification

Microsoft. (2017a). *Azure machine learning studio*. Retrieved from https://azure.microsoft.com/en-us/services/machine-learning-studio/

Microsoft. (2017b). *Microsoft security development lifecycle*. Retrieved from https://www.microsoft.com/en-us/sdl/

Moreno, J., Serrano, M. A., & Fernández-Medina, E. (2016). Main issues in big data security. *Future Internet, 8*(3), 44.

Mozaffari-Kermani, M., Sur-Kolay, S., Raghunathan, A., & Jha, N. K. (2015). Systematic poisoning attacks on and defenses for machine learning in healthcare. *IEEE Journal of Biomedical and Health Informatics, 19*(6), 1893–1905.

Narayanan, A., & Shmatikov, V. (2008). Robust de-anonymization of large sparse datasets. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy* (pp. 111–125). Washington, DC: IEEE Computer Society.

Narayanan, B. N., Djaneye-Boundjou, O., & Kebede, T. M. (2016). Performance analysis of machine learning and pattern recognition algorithms for malware classification. In *2016 IEEE National Aerospace and Electronics Conference (NAECON) and Ohio Innovation Summit (OIS)* (pp. 338–342). doi:https://doi.org/10.1109/NAECON.2016.7856826

National Institute of Standards and Technology. (2017). *NIST Special Publication Series SP 800 and SP 1800*.

Nelson, B., Barreno, M., Chi, F.J., Joseph, A.D., Rubinstein, B.I.P., Saini, U., Sutton, C., Tygar, J. D., Xia, K. (2008). Exploiting machine learning to subvert your spam filter. In: *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats, LEET'08* (pp. 7:1–7:9). USENIX Association, Berkeley, CA.

OWASP. (2017a). *OWASP SAMM project*. Retrieved from https://www.owasp.org/index.php/OWASP_SAMM_Project

OWASP. (2017b). *OWASP Top Ten project*. Retrieved from https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

Pauli, D. (2017a). *MongoDB hackers now sacking Elasticsearch*. The Register. Retrieved from https://www.theregister.co.uk/2017/01/13/elasticsearch_mongodb/

Pauli, D. (2017b). *MongoDB ransom attacks soar, body count hits 27,000 in hours*. The Register. Retrieved from http://www.theregister.co.uk/2017/01/09/mongodb/

Pimentel, M. A. F., Clifton, D. A., Clifton, L., & Tarassenko, L. (2014). A review of novelty detection. *Signal Process, 99*, 215–249. https://doi.org/10.1016/j.sigpro.2013.12.026.

Popa, R. A., Redfield, C. M. S., Zeldovich, N., & Balakrishnan, H. (2011). CryptDB: Protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles* (pp. 85–100). New York, NY: ACM. https://doi.org/10.1145/2043556.2043566.

Pozzolo, A. D., Caelen, O., Johnson, R. A., & Bontempi, G. (2015). Calibrating probability with undersampling for unbalanced classification. In *2015 IEEE Symposium Series on Computational Intelligence* (pp. 159–166). https://doi.org/10.1109/SSCI.2015.33.

Prasanna, B. T., & Akki, C. B. (2015). *A comparative study of homomorphic and searchable encryption schemes for cloud computing*. CoRR, abs/1505.03263. Retrieved from http://arxiv.org/abs/1505.03263

Risk Based Security. (2014). *A breakdown and analysis of the December, 2014 Sony Hack*. Retrieved from https://www.riskbasedsecurity.com/2014/12/a-breakdown-and-analysis-of-the-december-2014-sony-hack

Ron, A., Shulman-Peleg, A., & Puzanov, A. (2016). Analysis and mitigation of NoSQL injections. *IEEE Security and Privacy, 14*, 30–39.

Sahafizadeh, E., & Nematbakhsh, M. A. (2015). A survey on security issues in big data and NoSQL. *Advances in Computer Science: An International Journal, 4*(4), 68–72.

Samarati, P., & Sweeney, L. (1998). *Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression*. Computer Science Laboratory, SRI International. Retrieved from http://www.csl.sri.com/papers/sritr-98-04/

Schreier, J. (2011). Sony estimates $171 million loss from PSN hack. *WIRED Magazine*. Retrieved from https://www.wired.com/2011/05/sony-psn-hack-losses/

Selvi, U., & Pushpa, S. (2015). A review of big data and anonymization algorithms. *International Journal of Applied Engineering Research, 10*(17).

Shiravi, A., Shiravi, H., Tavallaee, M., & Ghorbani, A. A. (2012). Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers and Security, 31*(3), 357–374.

Software Testing Help. (2017). *37 most powerful penetration testing tools (security testing tools)*. Retrieved from http://www.softwaretestinghelp.com/penetration-testing-tools/

Soheily-Khah, S., Marteau, P.-F., & Béchet, N. (2017). *Intrusion detection in network systems through hybrid supervised and unsupervised mining process – A detailed case study on the ISCX benchmark dataset*. Retrieved from https://hal.archives-ouvertes.fr/hal-01521007

Song, D. X., Wagner, D., & Perrig, A. (2000). Practical techniques for searches on encrypted data. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy* (pp. 44–). Washington, DC: IEEE Computer Society. Retrieved from http://dl.acm.org/citation.cfm?id=882494.884426

Sweeney, L. (1997). Guaranteeing anonymity when sharing medical data, the Datafly System. *Proceedings: A Conference of the American Medical Informatics Association. AMIA Fall Symposium* (pp. 51–55).

Sweeney, L., Abu, A., & Winn, J. (2013). *Identifying participants in the Personal Genome project by name (a re-identification experiment)*. CoRR, abs/1304.7605. Retrieved from http://arxiv.org/abs/1304.7605

Tam, K., Feizollah, A., Anuar, N. B., Salleh, R., & Cavallaro, L. (2017). The evolution of Android malware and Android analysis techniques. *ACM Computing Surveys, 49*(4), 76:1–76:41. https://doi.org/10.1145/3017427.

Tellenbach, B., Paganoni, S., & Rennhard, M. (2016). Detecting obfuscated JavaScripts from known and unknown obfuscators using machine learning. *International Journal on Advances in Security, 9*(3&4). Retrieved from https://www.thinkmind.org/download.php?articleid=sec_v9_n34_2016_10.

Tramèr, F., Zhang, F., Juels, A., Reiter, M. K., & Ristenpart, T. (2016). *Stealing machine learning models via prediction APIs*. CoRR, abs/1609.02943. Retrieved from http://arxiv.org/abs/1609.02943

Viegas, E. K., Santin, A. O., & Oliveira, L. S. (2017). Toward a reliable anomaly-based intrusion detection in real-world environments. *Computer Networks, 127*(Suppl. C), 200–216. https://doi.org/10.1016/j.comnet.2017.08.013.

Yen, T.-F., Oprea, A., Onarlioglu, K., Leetham, T., Robertson, W., Juels, A., & Kirda, E. (2013). Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks. In *Proceedings of the 29th Annual Computer Security Applications Conference* (pp. 199–208). New York, NY: ACM. https://doi.org/10.1145/2523649.2523670.

Zuech, R., Khoshgoftaar, T. M., & Wald, R. (2015). Intrusion detection and big heterogeneous data: A survey. *Journal of Big Data, 2*(1), 3.