Automation for information security using machine learning

Automation for Information Security using Machine Learning

Stefan Martin Thaler

Printed by Gildeprint Drukkerijen, Enschede, The Netherlands
ISBN: 978-90-386-4696-1
A catalogue record is available from the Eindhoven University of Technology Library.

Automation for Information Security using Machine Learning

This research was partially supported by the following grants: The Dutch national program COMMIT under the THeCS project and the Big Data Veracity project; the European Union's Horizon 2020 research and innovation programme under grant agreement No 780495.

Cover: The cover shows an artistic fusion between circuit boards and real neurons. The circuit boards represent artificial neurons and the neurons on the back side are neurons from a mouse virtual context. The original picture[1] was released by Dr. Massimo Scanziani under the Creative Commons Attribution-NonCommercial 2.0 Generic license[2].

[1] https://www.flickr.com/photos/nihgov/3029203242
[2] https://creativecommons.org/licenses/by-nc/2.0/

Automation for Information Security using Machine Learning

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de rector magnificus prof.dr.ir. F.P.T. Baaijens, voor een commissie aangewezen door het College voor Promoties, in het openbaar te verdedigen op woensdag, 20 februari 2019 om 13:30 uur

door

Stefan Martin Thaler

geboren te Rum in Tirol, Oostenrijk

Dit proefschrift is goedgekeurd door de promotoren en de samenstelling van de promotiecommissie is als volgt:

voorzitter:	prof. dr. ir. J.J. van Wijk
1e promotor:	prof. dr. Milan Petković
copromotor:	dr. Vlado Menkovski
leden:	prof. dr. S. Etalle
	prof. dr. M. Pechenizkiy
	prof. dr. ir. R.L. Lagendijk (TU Delft)
	dr. M. Johnstone (Edith Cowan University)

Het onderzoek of ontwerp dat in dit proefschrift wordt beschreven is uitgevoerd in overeenstemming met de TU/e Gedragscode Wetenschapsbeoefening.

Acknowledgements

The process of getting a Ph.D. is often compared to a making a journey, but to me, this process has more in common with hiking a mountain where the top is hidden in the clouds. Let me explain to you why I think this way.

In the beginning, you are fresh and motivated. You get up early in the morning before the sun rises and you embark on a hiking trip to a mountain called the "Ph.D." You don't see the top, because it is hidden behind a thick layer of clouds. You don't mind and so you just start walking. At first, the path is a broad and comfortable road. You feel confident because you have walked on such roads all of your life.

Soon, rough terrain starts and walking gets a bit more exhaustive. After some time, the road gets smaller and steeper, you reach the woods, and you see even less which direction you are actually heading to. Bushes scratch you, and after some time the path gets barely visible. You take some rests, but you start feeling exhausted. Then, you reach the forest line, the trail has vanished completely, and only steep, rocky terrain is left in front of you. The air is getting thinner continually, which adds an extra strain. But you want to reach the Ph.D. top, so you start climbing in a general direction that seems passable. So, you climb, crawl, and move in other strange ways to proceed forward. You move in one direction, and you think you are getting somewhere, but only to realize that you have to go back because what looked like a path in the beginning was actually a dead end. Everything hurts, but instead of giving up you just keep going.

This process feels like it is never going to end, and many things happen on the way. Sometimes, you are asked to pick up a large boulder, and carry it a few kilometers down the mountain, because it looks much better there (I leave it to your imagination to what I am referring to here). Sometimes you have to sprint up the mountain for a few weeks. Sometimes you are seriously doubting your life choices when you see people going up another mountain in a shiny, cozy cabin (people who went to industry). Sometimes – especially on Fridays – you meet in your favorite, local hut (GEWIS) to relax and complain about this difficult endeavor to your fellow mountain climbers. But you keep on going, climbing, crawling and sweating.

And then, you walk around a large boulder, and out of nowhere you are finally able to see the mountaintop, just a few hundred meters ahead. You summon all your remaining strength and finish the last part all the way to the top.

I was not alone on climbing this Ph.D. mountain. During these almost five years, I met many awesome and interesting people, most of which helped me to climb this mountaintop and made this climb a worthwhile one. At this place, I want to thank them for that.

First, I want to thank the head of our research group, Sandro Etalle, who is one of the people that made this Ph.D. possible in the first place. Without the TU/e Security Group there would not have been a mountain for me to climb. Also, he talked some sense into me when I most needed it, which in hindsight I am deeply grateful for.

I want to thank my promotor Milan Petkovic, who guided my research pursuits over the past five years. In this mountain analogy, he would fly around in a helicopter. From there he kept the overview of what I was doing. He always ensured that I could pursue my interests freely but never went astray too much. He also directed me away from major pitfalls. We had many meetings, and I was told it is a privilege to get that much of Milan's time. People passing at the corridor would often see us laughing during these meetings, and may have concluded that we had a lot of fun. That is undoubtedly true, but more importantly, Milan can laugh at stupid things instead of getting angry - and throughout the past years I may have said a stupid thing or two.

I want to express tremendous gratitude to my Doctorate Committee – Mykola Pechenizkiy, Inald Lagendijk, and Mike Johnstone – who spent their valuable time and expertise to improve the quality of my thesis. By providing helpful comments and critical feedback, they ensured that the path I have discovered actually leads to a mountain top.

In the first one and a half years of my Ph.D. I was supervised by Jerry den Hartog. When I asked him if a path was correct, he would analyze it and show me that I have missed at least four other routes. Thank you for disentangling my thoughts and helping me to walk the first part of the climb. The more my research focus shifted towards machine learning, the more I was supervised by Vlado Menkovski. His encouraging nature kept me climbing. He ensured that I reclaimed my curiosity and kept looking under rocks to find interesting bits and pieces. Thanks a lot for the intellectual stimulation, tough questions and motivational pushes.

A special thanks goes to Mahdi, Ömer, and Laura, whom I shared the office throughout the last few years. They had to endure my rants about ontologies, deliverables and other things I deemed worth to complain about. Next, I want to acknowledge Riki Orcboss for creating and maintaining a list of hilarious anagrams of our group members' names. These anagrams literally made me laugh tears. Also, I want to thank Jolande, Mirjam and Anjolijn for creating a warm and friendly atmosphere in the group and helping with all the organizational hassles that we ran across. I also want to acknowledge Daan's shoes, which occasionally shared their timeless wisdom with the rest of the group.

There are of course many more people on this mountain, who made this hiking tour possible and worthwhile. On this hike, I have crossed path with many senior researchers, PhDs, and Master Students. Each would try to climb (or run in case of

Nicola) their own mountain during the day, and then return and meet again at the mountain hut GEWIS to compare blisters, complain about pains, have a few cold ones and play some foosball. A big "thank you" for many debates, lunch breaks, dark jokes, coffees, beers, cakes, sweets, trips, and other social activities to (in no particular order): Gustavo, Francesco, Bouke, Christine (thanks for initiating the social link between the crypto and security group - that entailed a lot of fun), Ale junior, Ale sober, Murtaza, Rien, Lorenz, Milan, Alex, Ömer, Mahdi, Laura, Dion, Davide, Daan, Andy, Daniel, Ali, Dominik, Harm, Leon, Wil, Frank, Manon, Manos, Niels, Jeroen, Rien, Chloe, Taras, Meilof, Matthew, Guillaume, Huyn, Reza, Luca, Nicola, Niek, Alessio, Sowmya (thanks for playing bingo with me), Taras, Sokratis, Serena, Fatih, Umbreen, Estuardo, and Mike (we still have to finish this round of office golf).

Next, I want to thank my parents for laying the base that ultimately enabled me to pursue a Ph.D. They raised me to be inquisitive, and I always knew I could count on them if I ever needed some backup. Thank you!!! I want to thank my brother for intellectually challenging me with countless hours of discussions thereby often revealing biases and flaws in my ways of thinking, and my sister for being my paranymph and enriching my life with many delicious cakes. A big fat thank you is also directed to my friends in Austria, the Netherlands and other parts of Europe, who made my off-time of research enjoyable.

Lastly, I want to thank my girlfriend Pia. If I wrote down all the things I want to thank you for throughout the last few years, I could probably fill a book on its own. But helaas pindakaas, this book is about a different topic, so I'll keep it short. I've really enjoyed our adventures so far, and I am looking forward to upcoming ones. Thanks for all the patience, support, love and friendship I have received from you, especially during the tough phases of my Ph.D.

To conclude I want to mention one of the most striking similarities of doing a Ph.D. and climbing a mountain, which is the marvelous view once you've reached the top. This majestic view humbles you but also fills you with a bit of pride that you were able to conquer it, if only for a little while. And, at the end of the day when you finally reach home, your feet are hurting, but all you can think about is which mountain to climb next ...

Eindhoven, February 2019

Stefan Thaler

View from top of the Wurmtaler Kopf (3225m), my brother and me, 2014.

Abstract

Many tasks in information security require data analysis to ensure the security of information and information systems. The amount of data that a security officer typically needs to analyze is large, which makes the automation of such analysis highly desirable. One way to achieve such automation of security tasks is by using machine learning. Machine learning techniques aim to solve tasks by learning a model from data, thereby drastically reducing the need for human labor. However, to apply machine learning on information security tasks a number of challenges need to be addressed.

Currently, research on information security concerning machine learning focuses mainly on two areas: the application of machine learning to solve particular security tasks, and the design of secure machine learning algorithms. In this work, we focus on three challenges that are crucial for the application of machine learning in information security: the lack of supervision, the integration of domain knowledge and the lack of contextual information. To address them, we select two specific tasks from two use cases, one from forensic log analysis and one from intrusion detection, in particular, data leakage detection.

To address these challenges, we propose the following: a method for signature extraction from forensic logs to verify our assumption that deep learning models are well-suited for capturing the structure of such logs; an unsupervised method for clustering forensic logs to address the lack of supervision; a method for integrating domain knowledge in the machine learning process; and a method that can create believable project decoys in a data-driven way that could be used for obtaining the lacking contextual information.

In this thesis, we empirically show the following: i) deep learning models are well-suited for modeling semi-structured forensic logs, hence can be used for signature extraction better than the state-of-the-art ii) deep learning models can outperform state-of-the-art methods for clustering forensic logs in an unsupervised way, which means that we can address such a problem without using labels iii) we can use domain knowledge to improve the learning of deep learning models for the clustering of forensic logs and iv) Markov Chains can be automated to generate believable project decoys in a data-driven way.

The proposed methods demonstrate ways to address the lack of supervision, the

integration of domain knowledge, and the lack of contextual information. Addressing these challenges leads to the advancement of automation for information security when using machine learning.

Apart from that, we believe that a similar, machine learning based methods can be applied to other data theft scenarios, with different types of decoy objects. Furthermore, we believe that similar DL-based methods as the ones proposed in this thesis could be applied to automate other tasks with similar challenges, for example, malware analysis or the detection of phishing attacks.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Many aspects of our modern life are being digitalized, i.e., transferred from physical space to cyberspace. This digitalization entails that more digital infrastructure is used and a rapidly increasing amount of data is produced. Much of this data that is valuable or sensitive and large parts of this digital infrastructure became essential for our complex societies to operate correctly. While the digitalization of our society made many aspects of our lives more efficient and more convenient, it also created the space for a new type of crime, namely cybercrime. The number of digital crime incidents has increased at the same rapid pace as the digitalization of our society. In 2018, cybercrime is estimated to have produced worldwide damage of more than 450 billion USD [82].

Information security is the discipline that attempts to counter such cybercrime. In particular, information security addresses the protection of information and information systems from unauthorized access, use, disclosure, disruption, modification, or destruction in order to provide confidentiality, integrity, and availability [9]. To illustrate the domain, we consider four examples of core information security tasks: determining whether a piece of software is malicious or not, detecting data leakage, preventing a phishing attack on an employee, and the problem of reconstructing events after a crime has happened on a digital system.

To determine whether a software program is malicious or not, the minimum data analysis that needs to be conducted is to analyze the binary code of the program. However, often, analyzing the program's binary data is not sufficient, as the maliciousness of a program can only be determined after analyzing the recordings of the actions of this software. Typically, such actions are recorded in log files, which often contain entries in the form of semi-structured sequential data. Semi-structured sequential data are sequences of tokens, where some parts of the tokens are fixed according to a schema, and other parts can be filled with variable tokens.

In the next example, the goal is to detect data leakage. For this, let us assume a company with digital intellectual property in the form of patents stored on company servers, customer data in databases and software in data repositories. Patents, software and customer data are valuable assets to the company. Hence it would like to prevent the unintentional distribution of such data to unauthorized third parties. To detect a

data leakage, different data flows from and to the company servers need to be analyzed.

The third example considers the task of protecting employees from a phishing attack. Let us assume an attacker wants to steal the credentials of an employee. In a typical phishing attempt, the attacker crafts a deceptive email that encourages the employee to click on a link and enter his or her credentials on a webpage that is under the attacker's control. To prevent such an attack from happening, data may need to be analyzed at many stages of this process. For example, emails when sent cause data to be produced on a server, the user's computer, and cause a stream of data to be sent via the network. Alternatively, if a user opens a webpage, streams of data will be sent via the network and cached on computer systems along the way. Of course, many benign emails and websites may be accessed at the same time. Thus, the challenge of preventing such a phishing attack is to analyze lots of data in a timely fashion to warn the user of the potentially impending attack, and thereby stop the credential theft.

For the last example, we illustrate the forensic analysis on a personal computer. Whenever such a computer is used, data is stored in many forms. When web pages are visited, their access is stored in the caches and the browser history. When emails are sent, they are stored in the mail client's database. When files are manipulated, metadata about the manipulation is stored in the file system. When software is executed information about the software state, errors and other important information are stored in logs. Imagine a crime has happened, and a computer system was potentially part of that crime. Then the goal of a forensic investigator is to extract relevant information from this computer system to reconstruct the actions that were involved in this crime. To do so, the investigator needs to analyze the data that is stored in multiple locations on such a computer.

A common theme in these four core examples is that often the protection of information systems involves sifting through a significant amount of data, i.e., data analysis plays a central role in solving many information security problems. Manually analyzing data by domain experts is cumbersome for many reasons. First of all, since the amount of data that needs to be analyzed in information security problems is typically large, manual analysis is labor-intensive, and therefore costly. The amount of data that needs to be analyzed is not only large, but it is also expected to grow exponentially in the foreseeable future. Secondly, since analyzing large piles of data usually involves repetitive tasks, it is likely error-prone when executed by humans. Thirdly, in many cases, the analysis needs to be conducted in a timely fashion, which is a problem because humans analyze data comparably slow.

Since manual data analysis scales poorly, security professionals often use software tools, i.e., expert systems, to aid them with their analysis task. Such tools automate repetitive tasks by formulating rules on the data, which allow to extract and aggregate valuable information from large piles of data in an efficient way. However, larger, more complex problems require a large number of rules. Domain experts need to consider exceptions, rules may contradict each other, and rules need to be updated and maintained to reflect the changing environments of information systems. As a consequence, the set of rules may be brittle, expensive to maintain or may not

generalize well. Often, it is simply impractical even for domain experts to devise a good, comprehensive set of rules for a given problem, because it is unknown what good rules are.

One possible way to tackle shortcomings of expert systems on data analysis tasks is the application of machine learning-based methods. Machine learning techniques attempt to learn a solution to a problem from data instead of relying on an expert-defined rules. Machine learning is a sub-field of artificial intelligence which evolved from the field of pattern recognition and computational learning theory.

The promise of obtaining solutions directly from data is compelling for information security since many problems in information security are very complex and finding good rules for expert systems is difficult. This property leads to many machine learning-based methods in information security. Examples such machine learning-based methods are the detection of anomalies from network traffic [81], the recognition of functions from binary code [136], the detection of hidden messages in speech data, i.e., steganalysis [84], the detection of data leakage [92], and the detection of phishing attempts from emails [102].

1.1 — Challenges

Data analysis plays a crucial role in addressing many information security problems. Since the amount of data that needs to be processed in such problems is large, automation is highly desirable. One way to achieve such automation is by using machine learning algorithms. Machine learning algorithms have the potential to solve well-defined problems from data instead of relying on manual, hard-coded rules. However, to apply machine learning methods, a number of challenges have to be addressed. In this thesis, we focus on three main challenges:

Challenge 1 - Lack of supervision. Many problems in information security that use machine learning rely on supervised machine learning. That is, to solve a problem a model is learned using labeled data, i.e., supervised data. For example, to determine whether a piece of software is benign or not, a machine learning model learns to distinguish benign from malign binaries by deriving its parameters from a set of binaries that are labeled benign and a set that is labeled malign. Obtaining labeled data is typically a labor-intensive and therefore costly process. Particularly in information security, labeled data is hard to get by, because data that is analyzed often contains sensitive or private information that should not be shared. An additional challenge in information security is that the context where models can be used continuously evolves. An additional challenge is, that information systems may change over time which could render existing labels obsolete. To overcome a lack of supervision or to increase the efficiency of the supervision, machine learning methods typically fall into four categories: unsupervised methods, semi-supervised methods, transfer learning methods, and n-shot learning methods. In this work, we address this challenge by proposing methods that draw from unsupervised and n-shot learning methods.

Challenge 2 - Usage of domain knowledge. In information security, data that needs to be analyzed comes from human-made processes, which means that often some form of domain knowledge is available. However, machine learning focuses on deriving a solution from the data directly, and it is difficult to use such additional domain knowledge in the learning process. The primary challenge is thus how to integrate such knowledge into the machine learning process. Typically, this challenge is addressed in one of the following ways. Domain knowledge can be added by creating a custom loss function or regularization, hence directly changing the learning problem. It can be added by reflecting the domain knowledge in the model architecture. Alternatively, it can be added by pre-processing the data in a certain way. In each of the cases, the main difficulty lies in the definition of the method. In this work, we address this challenge by defining a custom loss function in combination with metric learning.

Challenge 3 - Lack of contextual information. Machine learning models are powerful pattern detectors. In information security, there is a specific challenge, namely that two patterns may be the same, but one of them is malicious and the other one benign. For example, let us reconsider the previously mentioned example of data leakage detection. In such a scenario, the process of copying a software project from a project repository to an external hard drive can be both benign and malicious. It is benign if the purpose of the copying is to create a backup, but it is malicious if data is copied to the hard drive to steal this data. In such a case, the primary challenge here is the application of machine learning on patterns with ambiguous meaning with lack of contextual information. A typical solution to such problem explores alternative strategies that aim to obtain such contextual information, e.g., via the use of honeypots [143]. Honeypots are deceptive objects that are intrinsically worthless and aim to obtain the contextual information by enticing attackers to interact with them. In this work, we address this challenge by proposing a method for creating such deceptive objects.

Many other challenges need to be addressed when applying machine learning algorithms to information security problems. For example, in information security, errors are usually costly. A false negative can mean that a malicious entity caused harm to systems or data without anybody noticing. False positives on the other hand – especially when high in number – cost time of human experts that need to investigate the false alarm. Another challenge for applying machine learning on information security tasks is the semantic gap between results and the interpretation. Consider an intrusion detection system that forwards alarms of malicious behaviors to human operators. If an alarm is sent to the operator, but the operator fails to grasp the meaning of this alarm, the alarm will be ignored, which can lead damage to systems and data.

While these challenges are undoubtedly important, we focus on the lack of supervision, the usage of domain knowledge and the missing context, because they are more

fundamental to functioning of the machine learning process. Without supervision, many machine learning algorithms cannot solve any problems. Moreover, in case supervised data and additional domain knowledge is available, the domain knowledge cannot readily be incorporated into the learning process. Thus, valuable information gets lost. Furthermore, a lack of contextual information and ambiguous pattern render the learning process challenging.

Currently, these three challenges are typically only marginally addressed in the domain of information security. Instead, research focuses mainly on two areas: the application of machine learning to solve particular security tasks, and the design of secure machine learning algorithms, e.g., how to do privacy preserving machine learning or how to protect machine learning models that contain proprietary information from being pirated.

1.2 — Research Questions

The overarching objective of this thesis is to automate data analysis tasks in information security. Machine learning promises a way forward for achieving such automation, but as indicated in the application of machine learning is not straightforward and comes with many obstacles. In particular, we will focus on three challenges: the lack of supervision and the integration of available other domain knowledge that does not come in the form of labels, and machine learning in scenarios without sufficient contextual information.

The work in this thesis addresses these challenges by focusing on two information security use cases, signature extraction from the area of information forensics and the creation of decoy projects for data theft detection from the area of intrusion detection. We chose the forensic use case because it is representative of the challenges that we want to address in this thesis.

In the forensic use case, a forensic investigator has to analyze forensic logs. The volume of data that needs to be analyzed in such a scenario is typically large. Labels are not readily available, as the logs that need to be analyzed change frequently, and depend on the software and operating system where they are installed. Moreover, additional domain knowledge in the form of heuristics such as the length of the log lines or edit distances is readily available. Furthermore, this use case shares technical challenges from the examples mentioned previously in the introduction, which also need to operate on similarly structure data. We chose the data theft use case because it represents a scenario where an action such as backing up a project from one drive to another cannot be distinguished from stealing a project. Thus, in this scenario there is a lack of contextual information, which prohibits further data analysis.

The first use case is concerned with signature extraction from forensic logs and its related problem, clustering of forensic loglines. Typically, on a computer system and program states and events are written to so-called log files. An entry in such a log file consists of a sequence of tokens, some of them are fixed, and some of them are variable. For example, when a user named *johndoe* logs on to a system, a message such as "*New*

session c1 of user johndoe." could be stored in a log file. This log message indicates the system state that a user has logged on to the system, and which user has logged on. The session id, i.e., *c1* in our example and the username are mutable, and the rest of the logline is fixed. The fixed parts and the variable parts are typically defined by the programmer who wrote the software that is reporting its state. A signature is a template that consists of free text fixed parts and placeholders for variable parts that are used to create log messages for a specific program state.

Signature extraction is the process of deriving the signatures from a, and it is part of the event-reconstruction phase of a forensic analysis. In this phase, the forensic investigator attempts to understand the actions that happened in the system under investigation. To achieve such an understanding, the investigator collects traces from different information sources on this system. An essential source for traces is system and applications logs. System and application logs keep track of events that occur in systems and their software. Typically, these traces from all different sources are combined to one large log file, the forensic log.

One way to analyze such forensic logs is to extract or create corresponding log signatures. One common way to define signatures manually is by creating regular expressions. Log signatures allow an investigator to run more sophisticated analysis on the logs. For example, queries such as which users logged on to a system or how frequently did a user login become possible if signatures are available. Without log signatures, determining such information is difficult, because of the potentially high dimensionality of the variable parts and the complex relationships between variable and fixed parts. A log file may contain log lines originating from many hundreds of signatures, and a log line may consist of many variable and fixed parts. Manually defining log signatures is a labor-intensive and error-prone task.

Clustering forensic logs according to their signatures is a related problem to signature extraction. The objective of this task is to cluster the log lines according to the signatures that created them and independent of the variable parts of such a logline. The outcome of such clustering is a cluster id for each log line. The same signature should have produced two log lines with the same cluster id. Such cluster ids can be used for further processing of the logs, or to find more high-level patterns in these logs. If log signatures are already available, the process of log clustering turns to a simple matching problem, i.e., which log line matches which signature.

The second use case is concerned with the discovery of data theft via decoy objects. Digital data theft has become a massive problem in our ever more digitalized society. The total number of data breach incidents as well as the damage caused are rising at an alarming rate. Detecting data theft using machine learning can be difficult since for some patterns it cannot be distinguished whether they are malicious or not. One strategy to overcome this challenge and to detect ongoing digital attacks is baiting data thieves with digital decoys. These decoys are intrinsically valueless, which is why any interaction with them is suspicious and will alert the responsible security offer. One type of decoys is decoy documents. A decoy document is a file which contains seemingly sensitive information. These documents are usually

stored with other sensitive documents and carefully monitored. Interactions with them are reported and stored. However, while decoy documents are useful tools to obtain intelligence about an ongoing attack, most of a company's intellectual property comprises multiple files and folders, grouped in a project folder.

To investigate the challenges of lack of supervision and the incorporation of domain knowledge for our concrete use cases, we ask the following four research questions.

Research Question 1. How can we automate the signature extraction of semi-structured log sequences with labeled data?

We ask Research Question 1 to confirm our assumption, that machine learning methods are sufficiently capable of capturing the complex, high-dimensional semi-structured nature of the log sequences encountered in information forensics. We consider log lines to be semi-structured, due to the combination of fixed parts and variable parts. If a supervised method is not capable of modeling such data, it is unlikely that other, more sophisticated methods will.

In a typical information forensic scenario, labeled data is not readily available. An investigator will collect such logs from a machine without knowing which operating system, software, and services have been installed and produced such logs. Software and operating systems constantly change, so labeling data becomes impractical. To counter a lack of labels, ask the following questions:

Research Question 2. How can we automate the clustering of semi-structured log sequences without supervised data?

Clustering such logs will help a forensic investigator to uncover large patterns of similar logline sequences. In case of forensic logs, approximate domain knowledge in the form of heuristics is readily available. For example, the length of a logline is a reasonable indicator that two log lines are distinct or not - it is improbable that a very long and a very short logline originate from the same signature. However, such knowledge cannot readily be translated into labels and be used for supervision, and it may also not be correct. Hence, we ask the question:

Research Question 3. How can we use approximate domain knowledge to aid the automation of semi-structured log line clustering?

Finally, we turn to the data leakage detection use case. Our main challenge in this use case is that the ambiguity of patterns, i.e., the lack of contextual information. We, therefore, want to study, whether it is possible to use machine learning to automate tasks differently to address this problem. To achieve this, we investigate an alternative strategy, namely the use of decoy objects for detecting data theft. Since manually creating such decoy objects is cumbersome, we want to investigate whether it is possible to automate the creation of such decoy objects in a data-driven way. To be

able to fool an attacker, such decoys have to appear to be realistic, i.e., they need to be believable. Hence, we ask the following question:

Research Question 4. How can we automate the generation of believable decoys objects from structured data?

1.3 — Contributions

Here we detail our contributions to answer the four research questions that were outlined in Section 1.2.

To address **Research Question 1**, we develop a novel deep learning-based method for signature extraction from labeled data. The novelty of this method is that we learn a model based on data instead of manually creating an algorithm to solve the problem. We treat log lines in the forensic use case as semi-structured sequences of tokens. The tokens within these sequences have complex, non-linear dependencies. Furthermore, the variable tokens in such sequences entail high-dimensionality of the data. To address these of scalability, complex relationships and high-dimensionality within these sequences we resort to deep learning techniques.
Deep learning is a sub-field of machine learning, where models are composed from multiple non-linear layers of abstraction. In machine learning, features need to be engineered manually by experts. In contrast, the layered architecture allows deep learning models to learn useful representations for the task that they are deployed to solve. In addition to that deep learning models are trained using variants of the mini-batch stochastic gradient descent. Mini-batch stochastic gradient descent allows the learning of models that scale to millions of training examples and to efficiently learn models with billions of parameters [133].
In addition to that, deep learning methods have been successfully applied to problems in many other domains where vast amounts of complex, high-dimension sequences needed to be processed. Examples for such domains are speech recognition (e.g. [50]) or natural language translation (e.g. [24, 38]). Due to the previously mentioned properties as well as successful demonstration of these capabilities in other domains, we consider deep learning methods suitable for our forensic log analysis use case. Ideally, we want to be able to solve this problem in an unsupervised way, because labeling data involves tedious manual work. However, before we want to tackle the challenge in an unsupervised way, we want to show that it is possible to address the problem in a supervised way. Hence, we propose a method for identifying variable and fixed characters of log messages using a neural language model and demonstrate its efficiency. The work in Chapter 3 has led to the publication in [153].

Then, we address **Research Question 2** by showing that it is also possible to cluster forensic log lines according to their signature in an unsupervised way, i.e., without supervision. We propose a novel deep learning-based method, LSTM-AE+C, that uses an RNN auto-encoder to create a logline embedding space. These representations are subsequently clustered. The novelty of this method is that we use a

learned representation for clustering such logs instead of manually designing them. We empirically show that this method can cluster the forensic loglines according to their signature better than the current state of the art. The work presented in Chapter 4 has been published in [154] and in [155].

We contribute to **Research Question 3** by proposing a novel method for efficient metric learning by using approximate domain knowledge. The proposed method has two novelties: the use of approximate knowledge for learning, and way of combining labeled data with such approximated knowledge. This method learns a metric of forensic log lines. We adopt the triplet network architecture for sequential data by using recurrent neural networks (RNN) and improve the efficiency of the proposed method with regards to the labels by using a proxy distance metric (Jaccard distance) that allows us to learn a high-quality distance metric with a small number of annotations. We show that this method outperforms an RNN model on the same number of labels, which means that using this method we can use machine learning with additional domain knowledge, it can be readily combined with labeled data and the labeled can be used more efficiently. The results of this work have been published in [156].

To advance **Research Question 4**, we propose a novel method for generating believable decoy project folders for detecting data theft, which is based on Markov chains. The novelty of this method is that we learn to create such decoys in a data-driven way instead of manually designing rules to create them. Project folders are folders that contain information about some valuable information, for example, a software project. The main difficulty in this use case is that same patterns of actions may have ambiguous meaning, i.e., they can be malicious or benign. Hence, we follow an alternative strategy, namely to trick attackers into interacting with a decoy project folder that looks like a real project folder. Since we know that these decoys are worthless, any interaction with them is suspicious.

Our proposed method learns to represent folder contents using Markov chains. A Markov chain is a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event [112]. We treat the content of a folder as a sequence of random events, which are then captured by such a Markov chain. A Markov chain then learns the distribution of these events or appearances of files and folders in our case. Also, Markov chains allow sampling sequences of events from the learned distributions by performing a random walk over the Markov chain.

We use Markov chains to represent folders, and then use their sampling capabilities to create random folder contents. By creating multiple of such folder contents, we can create decoy projects. We empirically validated the perceived believability of the generated decoy projects and found that such decoys are hard to distinguish from real project folders. Believability is a crucial requirement for decoy objects in order to be useful for detecting data theft. Parts of the work presented in Chapter 6 have been published in [152].

A list of the author's publications can be found in Appendix D.

1.4 — Structure of the thesis

Here we outline the structure of the thesis. In Chapter 2 we introduce the concepts that are necessary for understanding the remaining chapters. In particular, we introduce a selection of deep learning concepts and ideas that are relevant for Chapters 3, 4 and 5. We also introduce the concepts of Markov chains and Mutual information, which are relevant for understanding Chapter 6. Furthermore, in Section 2.3, we provide an overview of the research focus of deep learning methods in information security to outline the challenges that are currently addressed.

Chapter 3 to 5 deal with the challenges of lack of supervision and integration of domain knowledge. In detail, in Chapter 3 we address Research Question 1 by verifying our assumption that deep learning models are suitable for learning the complex dependencies within forensic loglines. Next, in Chapter 4, we show that forensic logs can be clustered according to their signatures without supervision by proposing a method that can do so.

After we have verified that clustering in an unsupervised way is possible, in Chapter 5, we focus on the challenge of integrating available domain knowledge within the existing learning process. We propose a novel method that can integrate existing approximate domain knowledge via the loss term, and also allows for combining such approximate knowledge with knowledge in the form of labels.

In Chapter 6, we address the challenge of lacking contextual knowledge by describing a novel method that can automate the generation of decoy projects. Such projects can be used for obtaining the lacking contextual knowledge. Furthermore, we study whether the generated decoy projects are perceived to be believable, which is a crucial property for decoy projects to be applicable in a real scenario.

We conclude a with a discussion of the limitations of the presented work in Chapter 7. Furthermore, we motivate and point out further research directions.

Chapter 2

Background

In this chapter, we briefly introduce preliminary concepts that are relevant to understanding the other chapters of this thesis. This chapter consists of three sections.

In Section 2.1, we introduce the deep learning concepts that are relevant for Chapters 3-5. The main purpose of this section is to give a high-level understanding of these main concepts. To this end, we first want to outline the four main components of a typical deep learning solution. Then, in Section 2.1.1 we elaborate how a DL model can be composed, and briefly introduce the main architectures that we have used. After that, detail different types of data organization in Section 2.1.2. Then, in Section 2.1.3, we introduce the general tasks that we have been working on in this thesis. We conclude this section with a more detailed introduction to Long-Short-Term-Memory (LSTMs), as they are core components of the methods that we propose.

In Section 2.2 we briefly introduce Markov Chains and Mutual Information, two preliminary concepts that are necessary for understanding Chapter 6.

In Section 2.3, we provide a brief overview of the research focus of deep learning methods in information security. The purpose of this third section is to demonstrate that currently, the research focuses mainly on two topics: the application of algorithms to concrete problems and the security properties of deep learning algorithms. Furthermore, we want to provide this overview to point out that other challenges such as the lack of supervision, the use of domain knowledge are only sparsely addressed.

2.1 — Deep Learning

Deep Learning (DL) is sub-field of machine learning. Tom Mitchell defined machine learning as: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." [100].

Machine learning typically uses single-layered models to address tasks. Such a model is a parameterized function, and its parameters are derived from data, which serves as experience. For machine learning, data representations are typically designed manually by domain experts. In contrast to that, DL solutions use multi-layered models. These multi-layered models allow a DL solution to learn representations that

are useful to solve a task from data. That is, each layer in a DL model aims to learn a representation that is useful for the following layer in such a model.

Similar to machine learning algorithms, DL-based solutions typically consist of four components: a model, data, an objective, and an optimization procedure [47]. In the next four paragraphs, we will explain these components and how they relate to the definition of machine learning. We first introduce the model, data, and the objective and how they relate to each other. The relationship between these components is also depicted in Figure 2.1. We then sketch the optimization procedure, which uses the other three components perform the actual learning.

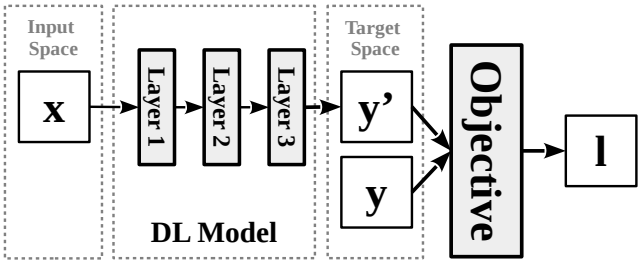

Figure 2.1: Relationship between data, multi-layered DL model and an objective function. The model maps data x from input into target space y'. An objective function and a ground truth y is used to calculate a loss l. The loss measures how wrong the model predicted y given x.

Model A model in the context of deep learning is a parameterized function, which maps input data to target data. An optimization procedure learns the parameters of such a model. Furthermore, when referring to deep neural networks, a model is often composed in different ways, which commonly referred to as architecture. We present different ways on how to construct such composite models in Section 2.1.1.

Data Data is an essential ingredient of DL-solutions as it can be understood as experience from which the model learns. DL-solutions can roughly be divided into two categories: supervised and unsupervised solutions. In supervised solutions, the experience comes in form of labeled data. For example, consider the task of classifying malware into malicious and benign. In a supervised scenario, a domain expert could provide the ground truth in form of labeled data that tell the algorithm whether the prediction y' was correct. In unsupervised solutions, the experience comes from the input data. For example, in an auto-encoder that tries to learn representations from a malware binary, one goal can be to reconstruct the input data, i.e., in this case the

ground truth $y = x$. For both categories, the relationship between the data, objective and the model as depicted in Figure 2.1 stays the same.

DL methods learn useful representations to solve a task from data, the learned model can only be as representative as the data was of the problem that is about to be solved. If the data is not representative, the model most likely is also not. Mostly, data used to train DL-models is high dimensional. One of the fundamental assumptions is that such high dimensional data concentrates around a low dimensional manifold, and the DL algorithm is capable of finding such lower dimensional manifolds.

Objective Many problems that are addressed with DL solutions are optimization tasks, where the goal is to minimize or maximize a certain objective function. This objective function serves as performance measure and determines how the parameters are learned. It tells a model the prediction error towards the ground truth. It is often also referred to as goal-, cost-, or loss-function. An example objective is the Mean Squared Error [40].

Optimization procedure The optimization procedure describes the steps to perform the actual learning. The optimization procedure describes the way how the performance of the algorithm improves with experience. For DL solutions, the optimization procedure is almost always a form of mini-batch stochastic gradient descent [47]. Mini-batch stochastic gradient descent performs the following steps for a subset of the overall data: a forward pass, which calculates the predictions y' given a model and the data x, a backward pass, which calculates the gradients of the objective with respect to the parameters; a parameter update step, which updates the parameters of the model.

Mini-batch stochastic gradient descent has a few compelling properties that motivate the use of it as an optimization procedure in DL context. First, since it is an iterative method, it does not require the whole dataset to be loaded into working memory. Instead, only a mini-batch needs to be loaded in memory, which allows working on huge datasets. Secondly, it can be parallelized, which allows for higher speeds and much better scaling potential than other exact gradient-based methods [21].

In pseudo-code, the three steps of stochastic gradient descent could be written in the following way. Let x be our input data, y our target output, $y' = f(w, x)$ our model that uses parameters w to map input x to a prediction y', and $o(y, y')$ a cost function that defines the *wrongness* of a predicted value y' to the correct value of y, and α be the learning rate. Then, mini-batch stochastic gradient descent in pseudo-code could be written as shown in Algorithm 2.1.

Recent methods use momentum-based forms of mini-batch SGD such as Adam [74] or RMSprop [159], which speed up the learning and reduce the chance of getting stuck at a plateau of the cost function.

2.1.1 – Deep Learning Architectures. Typically, a DL model component is composed of different "building blocks", which are commonly referred to as layers. The

2

Algorithm 2.1 Stochastic Gradient Descent in pseudo code.

while *Not at a local minimum of the objective* **do**
    /* 1.Forward pass */
1   $y' \leftarrow f(w, x)$
    /* 2.Calculate gradient */
2   $g \leftarrow \nabla_w o(y, y')$
    /* 3.Update parameters */
3   $w \leftarrow w + \alpha * g$
end

overall composition of these building blocks is typically called the model architecture. Mathematically speaking, a layer is a function $y = f(x)$ that maps some input x to some output y. Typically, f is a parameterized function, although non-parameterized functions building blocks for neural networks also exist. In this section we introduce common building blocks that have been used to build the architectures in Chapters 3 to 5.

Fully-connected (FC) Layers, Multi-Layer Perceptrons (MLPs) A fully-connected (FC) layer maps each of the input elements to each of the output elements in a non-linear way. That is, for an input vector x, and output vector y, trainable parameters W and biases b, and σ as a non-linear activation function, a fully connected layer maps input to output in the following way: $y = \sigma(x^\top W + b)$.

Fully-Connected layers are a fundamental building block and can be found in many DL model architectures. Multiple, chained fully-connected layers can learn arbitrary function mappings between the input and the output [64]. Fully-connected layers are also often found in other architectures such as convolutional neural networks and recurrent neural networks.

One common architecture that is composed of multiple FC layers is the Multi-layer Perceptrons (MLPs). MLPs are feed-forward neural networks that are composed of multiple, fully-connected layers. This composition allows each layer to use the features of the previous layer to create more abstract features. An example of a two-layer neural network could look like this: the first FC layer outputs a hidden representation h which is defined $h = \sigma(x^\top W_1 + b_1)$, and the second layer outputs y as $y = \sigma(h^\top W_2 + b_2)$.

MLPs can approximate arbitrary, non-linear functions [64], and should not be confused with perceptrons, which can only learn simple, linear relationships between the input and the target data. One significant disadvantage of MLPs is that the model size grows proportionally to the dimension of the input features, which makes it a poor choice for very high dimensional data such as large images or sequential data.

Recurrent Neural Networks (RNN) Often, data is arranged sequentially, i.e., the order and the relationship between elements of that data convey information. Examples of such sequential data are time series, natural language text or audio files. Often, this data is high dimensional, and of varying length. MLPs are unsuitable for such data, as the number of parameters required for analyzing such data would grow very large. Furthermore, MLPs are rigid, which means that only sequential data of a fixed length could be represented. To address these issues of variable sequence length and exploding parameter count, Recurrent Neural Networks (RNNs) have been developed.

RNNs are neural networks that are designed in a way to reuse the outputs of the network in later calculations. The core of an RNN is a cell function, which is a neural network model that maps an input to an output. This cell function is recurrently applied to each element of the input sequence. The parameters of this cell functions are shared for each of the time steps, which allows capturing the dependencies between the elements of the sequence.

RNNs can be used to learn functions in flexible ways. They can be used to learn functions to map sequences to a single output (many-to-one), for example, to classify sequences. They can be used to learn functions that map sequences to other sequences (many-to-many), for example, to tag sequences with specific labels or for natural language translation [24]. Moreover, they can be used to map a single value to multiple outputs (one-to-many), for example, to generate descriptive text from an input image [168].

Autoencoders (AE), Recurrent Autoencoders (RAE) Autoencoders (AE) are composite models that consist of two models: an encoder model e and a decoder model d. The goal of an AE is to learn to reconstruct its original input. That is, given an input x, learn the parameters for its encoding model e and decoding model d in such a way that $h = e(x)$ and $x' = d(h)$, i.e., $x' = d(e((x))$ in such a way that it minimizes the difference between x and x'.

If an AE has sufficient capacity, it will learn two models that will copy the input to the output, which generally not useful. Therefore, the representations that an AE has to learn are typically constrained. Examples for such constraints are forced sparsity or a dimensionality bottleneck on the output of the encoder. Such constraints prevent an AE from correctly copying the data. Hence, the encoder model needs to learn representations that contain potentially useful properties or regularities of the data.

As AE do not rely on labeled data, they are typically used for unsupervised representation learning. Hence, this approach may find a suitable representation from a large, unlabeled dataset. Such a learned representation typically has a reduced dimensionality in comparison to the original data and captures properties of the are useful for reconstructing the data.

The input data type may influence the choice of architecture for the encoder and decoder model. A typical choice for modeling sequences of data is to use RNNs.

Hence, when faced with the challenge of learning a representation for sequences in an unsupervised way, a common choice is to use a Recurrent Autoencoder (RAE). Recurrent Autoencoders are similar to Autoencoders, except that the encoder model and decoder model are variants of RNNs. Such Recurrent Autoencoders can learn dense representations for sequences in such a way that they maintain information about the order and relationship of the elements in that sequence [33].

2.1.2 – Data Types. DL algorithms learn models from data. Data can be organized in many forms, and depending on the data organization the model architecture will vary to solve tasks. Concerning the data, DL-solutions differ from shallow machine learning solutions. In machine learning solutions, the organization of the data plays only a minor role, as the representations that are used in shallow machine learning models are typically hand-crafted. The resulting solutions depend on the features, which depend on the domain. In contrast, DL models consist of multiple layers, each of which learns representations that are useful for the following layer. As a consequence, DL solutions are much more dependent on the data type that is being analyzed, but much less dependent on the domain.

The organization of the data depends on the domain and the task that is solved. There is no one right way to organize, but commonly data are organized in a way that is efficient to process or natural for the domain. Here, we distinguish between five different data types: spatial data (PD), sequential data (QD), and structured data (SD), text data (TD) and multi-modal data (MD). In the next paragraphs, we will describe these five data types in more detail.

Spatial Data Spatial data is data that is organized in such a way that individual data points are addressable by an N-dimensional coordinate system. For certain data types, the spatial location of data points carries much information about the information stored in the data. An example is images, where much of the information that is contained in the image is stored the location of a specific pixel. Consequently, a spatial representation of such data offers the advantage of leveraging the information that is stored in this information.

In InfoSec, the most prevalent type of spatial data which DL is applied to is images. Images occur in naturally in domains such as biometrics, steganography, or steganalysis where images are a data modality of central interest for the domain. However, images also occur in other domains, such as in website security, where researchers attempt to break CAPTCHAs, which are often visual. Apart from that, only a few approaches in InfoSec use DL on spatial data, such as organizing audio data into a spatial arrangement using spectrograms, or binary malware data that gets transformed into a spatial arrangement.

Sequential Data Generally, sequential data are ordered lists of events, where an event can be represented as a symbolic value, a real numerical value, a vector of real or symbolic values or a complex data type [181]. Sequential data can also be

considered as 1D spatial data, that is, locally correlated and each event is referable via a 1D coordinate system. Furthermore, text can also be seen as sequential data, either as a sequence of characters or as a sequence of words, although the relationship between the elements of text is complex. The distinguishing is arbitrary and depends on the data, the treatment of the data and the chosen data analysis techniques. For example, network traffic can be treated as a sequence of packets, but also each packet could be analyzed individually, without regard to the order of the packets.

Many challenges in InfoSec depend on analysis of sequential data. Most prominently there is the analysis of network traffic to discover attacks, the analysis of audio files for steganography and steganalysis, and the analysis of logs.

Structured Data Structured data are data whose creation depends on a fixed data model, for example, a record in a relational database. Semi-structured is a data type that has a structured format, but no fixed data model, for example, graph data or JSON formatted data. Here, we combine structured and semi-structured data, since from a DL perspective they are very similar.

A prominent source for structured data in InfoSec is network traffic. Network traffic is usually parted into packages, each of which contains a fixed set of fields with specific values. Another source of structured data is event logs that are stored in a database, if the data base depends on a fixed data model. We also consider data where a fixed set of high-level features (the stress is on high-level) is extracted from data as structured data because this equals a record in a database where each field of the entry corresponds to a feature.

Text Data Text data are a form of unstructured, sequential data. Commonly, a text is interpreted in two ways: as a sequence of characters, or as a sequence of tokens where tokens can be words, word-parts, punctuation or stopping characters. The elements of a time series are generally related to each other in a linear, temporal fashion, i.e., elements from earlier time steps. This relationship is often also true for text data. However, the relationships between the elements are more complicated, often long-term and high-level.

In InfoSec, text data is mainly found in log files and in analyzing communication such as social media messages or emails. Another source of text may be the source code of software to identify vulnerabilities.

Text data is generally treated as sequential data and analyzed with similar means. One problem of analyzing text is to find suitable representations of the input words. Text data is high dimensional, which renders sparse representations computationally impractical. Hence, commonly dense representations such as Word2Vec [99] and GloVe [117] are used to represent input words.

Multi-Modal Data At times more than one data modality is used for solving a problem, or multiple modalities are derived from the same data type. An example

of multiple modalities are videos that contain both a spatial and a temporal component. Such data can be analyzed by combining models like Convolutional Neural Networks(CNNs) and LSTMs to represent both of the aspects of the data.

2.1.3 – Tasks. Tasks are the problems that should be solved. One of the key challenges to solve a problem with machine learning is how to present the problem in such a way that the machine learning algorithm can solve it. For example, the problem of finding malware can be reduced to a classification problem, i.e., given a binary file, decide whether it is malicious or benign. How to solve a problem and which task to use is often a design decision, since the borders between different tasks are often vague or a problem can be solved in multiple ways. For example, the previously mentioned problem of finding malware can also be addressed by a clustering task. In such a clustering task, binaries would be grouped based on features and a distance metric. Then, if a binary that needs to be tested lands in a group with known malware, it could be flagged as malicious.

Here we briefly introduce the tasks that we address in this thesis: classification, which is used in Chapter 3, representation learning, and clustering which are used in Chapter 4, and metric learning, which is used in Chapter 5. The tasks that we are describing here are not unique to DL but are described from a machine learning point of view.

Classification Classification is the task of mapping an input sample to a discrete category of output classes. Many problems can be formulated as a classification task, and typically there are multiple ways of doing so. For example, biometric fingerprint matching can be framed as a classification task in two ways. One could frame it as binary classification task of given two fingerprints, are they from the same individual. Alternatively, one could ask: given a fingerprint, to which individually does it belong. Regression tasks are closely related to classification tasks, but instead of discrete output categories, the goal is to predict a numerical value.

Clustering Clustering (CL) is the task of finding groups and group membership in a set of objects. Objects that are similar to a certain metric and attributes should be grouped, and objects that are dissimilar should not be grouped. Clustering is related to classification in the sense that it assigns membership to a certain group. However, the main focus of clustering is to find such groups of similar objects. Deciding the number of groups is a currently unsolved problem, but several heuristics exist [146, 158].

In InfoSec, clustering techniques have the appealing property that they can discover previously unknown groups from data. For example, clustering techniques can be used to identify previously unknown attacks, if such attacks have similar patterns. In InfoSec, DL-techniques are typically not used for clustering directly. Instead, they are used to learn representations that can be clustered in a second step.

Representation learning Consider the problem of deciding whether an email is spam or not. One strategy of addressing this problem is to train a binary classifier that addresses this challenge on labeled examples of the email's body text. When opting for this strategy, the expert has to decide how to represent the body text data in a way that the classifier can use it. Often, such representations need to fit computational and mathematical constraints. For example, many classification algorithms perform poorly on high dimensional or sparse input data, which makes such representations less suitable.

Representation learning (RL) is the task of deriving representations from data that are suitable for solving another task. A typical output of such a task is a function that maps the raw data from input space to a more suitable feature space. For example, in the spam classification example, a representation learning task could yield a low-dimensional, dense representation of email body texts that capture relevant properties for deciding whether the email is spam or not.

In InfoSec, representation learning based on DL-techniques is often used to reduce the dimensions while capturing relevant properties of the input data, e.g., [87] or to transfer domain knowledge from a similar task, e.g., [172].

Metric learning Consider the challenge of image-based authentication. In such a case, an authentication system would need to identify a person given a stored image and a current image. If such images are represented as pixels, this is a challenging task because of the complexity and the high dimensionality of the input data. The information stored in the pixels is only loosely correlated with the location in the image, and other factors such as lighting or perspective may distort the pixels of the image. On the other hand, if a good measure of similarity between those two images were available, the task would become trivial. In such a case, one would calculate the similarity and then decide on a threshold.

Many problems such as clustering depend on a suitable distance metric to be solved or become trivial if a suitable distance metric is available. A distance metric is a function that defines a distance between each pair of elements in a set of elements. Standard distance metrics such as the Euclidean Distance are readily available. However, such distance metrics often fail to calculate a meaningful distance between two data-points.

Distance metric learning is the task that is concerned with finding a meaningful distance metric [180]. The main idea of metric learning is to find a projection function that maps data points from their input space to a target space, in which similar points are close to each other, and dissimilar points are further away from each other with respect to a chosen standard metric such as the Euclidean distance.

2.1.4 – Long-Short-Term-Memory Networks (LSTM). RNNs (see Section 2.1.1) can theoretically learn to capture sequences of arbitrary length. To train RNNs on sequences of data, the recurrent cell function of the RNN has to be applied recurrently as many times as the length of the sequence. Hence, for long sequences, a RNN

essentially becomes a very deep neural network. One consequence learning such very deep networks is that the gradient signal vanishes, i.e., the signal that is required to learn meaningful connections in the network tends towards zero. This vanishing gradient leads to a loss of information about long-term dependencies between the elements of the sequence [11].

Long-Short-Term-Memory Networks(LSTMs) are a form of recurrent neural networks (RNN) that have been introduced by Schmidhuber and Hochreiter to address the problem of vanishing gradients [62]. The core idea of LSTMs is to introduce a cell state that stores long-term information about the current state of the sequence and is passed on with modification between each call to the cell function. Hence, the output for each input element of the sequence depends on the cell state and the current input element. Keeping a cell state allows an LSTM to address the problem learning long-term dependencies. Conceptually, the idea of the cell state is related to the residual connections [57], which were introduced to counter the vanishing gradient problem in very deep convolutional neural networks.

On a high level, the cell function in an LSTM is implemented in the following way. Similar to an RNN, this cell function is applied to each element of the sequential data. Here, we refer to the index of an element in the sequence as a time step. For a single time step, the cell function maps an input-tuple (x_t, c_{t-1}) that consists of the current input and the previous cell state to an output tuple (h_t, c_t), that consist of the output data for that element and an updated cell state. Here, x_t is the input sequence at time step t, c_t is the cell state and h_t is the cell output.

The flow of information in this cell function is controlled by so-called gates, each of which is a neural network. There are three gates in an LSTM cell function, an input gate, an output gate, a forget gate. The input gate controls which parts of the input are added to the state, the forget gate controls which parts of the state should be forgotten, and the output gate controls which parts of the state should be used to produce the output.

In more detail, the cell function is composed of five functions: a forget gate function (see Equation 2.1), an input gate function (see Equation 2.2), an output gate function (see Equation 2.4), a cell update function (see Equation 2.3) and an output function (see Equation 2.5). In Equations (2.1) to (2.3), σ denotes the sigmoid function, $\odot$ represents the element wise product, and all W represent learnable parameters. The relationships of these functions are as follows.

First, the forget gate f_t decides based on the current input x_t and the previous output h_t which information of the cell state to keep, and which one to discard. A zero at the forget gate means that the value should be forgotten, and one means that the value should be kept.

$$f_t = \sigma(W_{fx}x_t + W_{fh}h_{t-1} + b_f) \tag{2.1}$$

Then, the input gate i_t controls what parts of the input are relevant for updating

the cell state in a similar way to the forget gate.

$$i_t = \sigma(W_{ix}x_t + W_{ih}h_{t-1} + b_i) \tag{2.2}$$

The cell state update function decides which information of the previous state and the current input should be used to update the cell state.

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_{cx} + W_{ch}h_{t-1}) \tag{2.3}$$

Next, the $o_t$decides which parts of the previous output and the current input are relevant for the output of the current time step. The output function combines the updated cell state with the relevant output parts.

$$o_t = \sigma(W_{ox}x_t + W_{oh}h_{t-1} + b_o) \tag{2.4}$$

Finally, the output function defines the output of the current time step. It is defined by the output of the output gate with the updated cell state.

$$h_t = o_t \odot c_t \tag{2.5}$$

Training LSTMs Since LSTMs are an RNN variant, they are commonly trained with stochastic gradient descent using backpropagation through time (BPTT) [174]. BPTT is a variant of back propagation, and it consists of the same three steps as the backpropagation algorithms (see Algorithm 2.1): a forward-pass step, a backward pass, and a parameter update step. However, since the parameters of the cell functions are shared, the LSTM will be unfolded for a certain number of time steps. That is, all parameters of a cell function are replicated for the number of timesteps. The forward pass is calculated for all unfolded cells. Then, the gradient of the cost function with respect to the parameters is calculated. Finally, in the update step, an aggregate of the gradients such as the average is applied to the parameters of the cell function.

Addressing exploding gradients LSTMs are designed to be capable of learning long-term dependencies in the data. The cell state can be viewed as a form of residual connection, which passes through long-term relationships in the data to later cells [57]. However, when an LSTM is unfolded for many time-steps, gradients may become prohibitively large for learning. This phenomenon is called the exploding gradients problem. One simple but effective way to address this problem was proposed by Pascanau et al. in the form of gradient norm clipping [116]. Their solution involved clipping the gradients if they became larger than a certain threshold.

Addressing overfitting One common problem when training deep neural network is the problem of overfitting. The problem of overfitting is an error that arises from sensitivity to small variations of the data [45]. DL models often have a large number of parameters and layers. This large capacity allows the model to memorize the training

data, which is called overfitting. As a consequence, overfit DL models perform well on the training task, but generalize poorly to new data points. DL models that cannot generalize well to new data points have little use for solving a task.

In this thesis, we used two techniques to address the problem of overfitting, dropout [78, 144] and batch normalization [68].

Dropouts [144] is a training-time regularization technique that aims to avoid overfitting by stochastically setting the outputs of a previous layer to zero at a given probability of p, with p being a hyper parameter. This mechanism should prevent units of previous layers from co-adapting too much, i.e., it helps to prevent layers to rely too much on local pattern variations. Dropout essentially turns one network into an ensemble of networks, thereby increasing the generalization capability of the network. Apart from that, it has been empirically shown that dropout increases the training speed of DL models [144].

Batch normalization [68] is a regularization technique that aims to improve the stability of learning process of DL models by reducing the internal co-variance shift. The internal co-variance shift is the change of the input distribution of each layer of the DL model over the time of the training. Such a co-variance shift can slow down the convergence of the training of a DL model, as the input could end up in the saturating part of the non-linear activation function. Batch normalization reduces this co-variance shift by normalizing the means and variances of the layer inputs. It achieves this normalization by the following four steps. First the mean μ and variance σ^2 of a batch of input data x is calculated. Then, each batch element of this normalized by this mean and variance. Finally, each such normalized element of this batch $\hat{x_i}$ is shifted and scaled by two learnable parameters γ and β.

In detail, the four steps to calculate batch normalization BN for a batch of input data $x_1 \dots x_m$ with m elements are defined as:

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x_i, \sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu)^2, \hat{x_i} = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}, BN(x_i) = \gamma\hat{x_i} + \beta \quad (2.6)$$

ϵ in this equation is a small stabilizing constant that is added to the variance to prevent division by zero in the normalization step.

Batch normalization has empirically shown to speed up and stabilize the training procedures of DL, and in cases also reduce the overfitting of the network such that it provides an alternative to Dropout [68] .

Alternatives to LSTM LSTMs are a form of gated RNNs. An alternative that we considered in our thesis was the gated recurrent unit (GRU) architecture that was proposed by Cho et al. [24]. The main difference between a GRU and an LSTM is that a single gating unit controls which parts of the state to forget, and which parts of the input to keep. As a consequence, GRUs have fewer parameters than LSTMs, thus

theoretically should be faster to train while it has been claimed that they have similar performance to LSTMs [24].

Empirical studies (e.g., [53,71]) aimed to evaluate different variants of gated recurrent neural networks. However, none of these variants was found to be consistently better than others.

In our experiments, LSTMs consistently outperformed GRUs regarding accuracy, while not yielding any significant performance increase regarding training speed. Hence, for our experiments we used LSTMSs and variants of LSTMs that had minor improvements.

2.2 — Markov Chains and Mutual Information

In Chapter 6, we propose a method for automating the process of creating decoy project folders. To achieve this, we want to learn a probability distribution from real project folders. This distribution should capture files, folders and their structure. Once we have obtained such a probability distribution, we could sample from this distribution to create new, realistic looking project folders.

Ideally, one would learn a probability distribution over of all files and folders co-occurring within each other. That is model the probability distributions of files and folders that occur together. The co-occurrence determines whether a folder will be perceived realistic. However, calculating all paired probabilities for n files and folders requires $n * 2^n$ operations, which is computationally unfeasible in practice.

A simple solution would be to capture the distribution of folder contents of folders with the same name. However, such a probability distribution captures only the overall distribution of the folder's content, but not the co-occurrence of the files. Hence, we need to find a way to approximate the probability distribution over the co-occurrence of files and folders.

A Markov chain is a stochastic process that models sequences of random events. To predict an upcoming event given previous events, one would need to model the conditional probability of this event given all the previous. Markov chains approximate this conditional probability distribution by assuming the Markov property, i.e., the next event only depends on the current event.

In this work, we treat folders' contents as a sequence of random events, which allows us to approximate files' co-occurrence with Markov chains.

In this thesis we will use first-order, finite-state, discrete-time, absorbing Markov chains. For brevity, we will refer to them simply as Markov chains. We will adhere to the notation of Kemeny and Snell [73]. First order Markov chains only depend on the current event to predict the next event. Finite-state means that the amount of states that a Markov chain can have is finite, in contrast to being infinite. Discrete-time means, that transition from one state to another happens at discrete instances of time, i.e., each time step is an integer.

Let $S = \{s_1, s_2, ...s_n\}$ be a finite set of n states and P an $\mathbb{R}^{(n \times n)}$ matrix. Then a Markov chain is defined as follows:

2

Definition 2.1. *A* Markov chain *(*S, P*) comprises a set of states* $S = \{s_1, s_2, ...s_n\}$ *and a state transition matrix* $P \in \mathbb{R}^{(n \times n)}$ *with each element of the transition matrix* P_{ij} *describing the probability of transitioning from state* i *to state* j*. A state* s_i *is called* absorbing *when* P_{ii}=1.

Parameter Estimation A common way of estimating parameters for Markov chains is via maximum likelihood estimation. The goal of maximum likelihood estimation is to maximize the probability that generating a sequence of events is similar to observed sequences of events.

In Markov chains, the transition probability from one state to the other only depends on the current state. Hence, the maximum likelihood can be estimated in the following way. Let n_t be the total number of observed transitions from a state s_i to any other state. Furthermore, let n_j be the number of observed transitions from state s_i to state s_j. Then the transition probability P_{ij} can be estimated by:

$$P_{ij} = \frac{n_j}{n_t} \tag{2.7}$$

Example of a Markov Chain Here we give a simple example of a Markov chain with an absorbing state. Let us assume a scenario with four states. Then S is the set $\{s_1, s_2, s_3, s_4\}$. Further, let's assume the transition matrix P is given by:

$$P = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \\ s_4 \end{array} \begin{array}{c} \begin{array}{cccc} s_1 & s_2 & s_3 & s_4 \end{array} \\ \begin{pmatrix} 0 & 0.3 & 0.7 & 0 \\ 0 & 0 & 0 & 1.0 \\ 0 & 0.6 & 0 & 0.4 \\ 0 & 0 & 0 & 1.0 \end{pmatrix} \end{array} \tag{2.8}$$

We can visualize such a Markov chain also as a graph where the states are the nodes and the transition probability are the edges. The graph of our Markov Chain (S, P) is depicted in Figure 2.2. Suppose we were in state s_1. Then, we see that the probability to transition to state s_3 is 0.7, and the probability to transition to state s_2 is 0.3. We can also see that s_4 is absorbing, since we cannot leave s_4 to any other state.

2.2.1 – Mutual Information. In Chapter 6 we will learn a decoy model from project folders. In order to maximize the information of our learned decoy model, we use a heuristic based on the mutual information [30].

Mutual information measures the dependency between two random variables X and Y. In this context, the two random variables are whether two files are present within a set of folders with the same name or not. Furthermore, we use discrete probabilities to represent the co-occurrence of files.

Let X and Y be discrete random variables, $P(X)$ and $P(Y)$ probability distributions over X and Y, and $P(X, Y)$ be a joint probability distribution over X and Y. Further-

more, for brevity, we write $P(x)$ instead of $P(X = x)$. Then the mutual information is defined as follows:

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} P(x,y) \log \frac{P(x,y)}{P(x)P(y)} \tag{2.9}$$

2.3 — Deep Learning in Information Security

In this section we systematically survey the state of the art of deep learning research in information security. We describe the methodology which we used to conduct this survey in Appendix C. The purpose of this survey is to provide an overview of which challenges are currently addressed by the research community.

Deep learning is a sub-field of machine learning. We focus on deep learning techniques in this survey, as in many fields such as computer vision, natural language processing and speech recognition the methods with leading performance rely deep learning techniques.

On a high level, we group the surveyed approaches into two parts. The first part consists of Sections 2.3.1 to 2.3.5 and presents approaches that apply DL algorithms to address security and privacy issues. This first section is structured along the five data types that we introduced in Section 2.1, i.e., sequential-, spatial-, structured-, text- and multi-modal data. For each of the data types, we group the surveyed approaches based on the state security task of that approach. For each of the tasks, we compare the different architectural decisions to solve the tasks and draw connections to advancements from in the DL community. We chose this data-type drive perspective on the use of DL in InfoSec because it reflects on the fact that the successful application of DL methods is data dependent, but not domain dependent. We believe that this perspective will help researches identify challenges and potential solutions to data of their domain more easily.

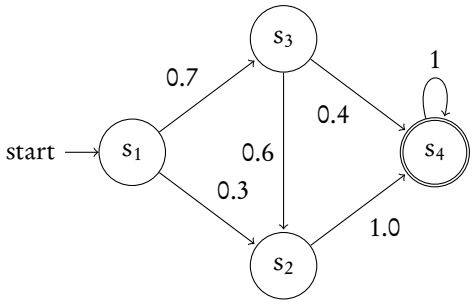

Figure 2.2: An example of a Markov Chain.

The second part – Section 2.3.6 presents approaches that address security and privacy issues of DL algorithms rather than using DL algorithms to solve security problems.

2.3.1 – Sequential Data.

Function recognition Binary data or bytecode analysis is an important tool for malware analysis. One challenge of binary data analysis is function recognition, i.e., finding the start and end positions of software functions in a piece of code. Shin et al. tackle the challenge of function recognition from binary code by framing it as a binary classification task [136]. They treat the binary code as a sequence of one-hot-encoded bytes and predict for each byte whether it is a start or end byte of a function. They evaluate a variety of sequential models such as RNNs, LSTMs [62], GRUs [28], and bi-directional RNNs [130], which are all well-suited for finding regularities in complex sequences. They train all their models using RMSProp [159], which is a momentum-based back-propagation through time variant. They validate their approach on multiple datasets, and their best-performing model, a bi-directional RNN, achieves an F1-score between 98% and 99%, which means an average improvement of 4% to the then state-of-the-art, and the training and the evaluation are an order of magnitude faster than the state-of-the-art.

Instead of predicting start and stop bytes, Chua et al. aim to identify the function type given x86/x64 machine code [26]. To this end, they treat the machine code as a sequence of instructions, which are embedded using neural word embeddings [10]. These embedded sequence instructions are then modeled using a 3-layer stacked GRU-RNN since the sequences are variable length and since the stateful nature of the recurrent neural network aids modeling sequence of instructions. The approach, which is named EKLAVYA, is evaluated on the machine code of two commonly used compilers: cc and clang and on multiple classification tasks. EKLAVYA achieves an average of up to 0.9748 accuracy for unoptimized binaries, and up to 0.839 accuracy for optimized code.

Traffic Classification Chen et al. propose a method for classification of mobile app traffic classification [23]. They capture the traffic, transform the headers into a 1-hot encoding and treat a sequence of packages as a sequence of vectors. They then train a 6-Layer custom Convolutional Neural Network(CNN) model to analyze the traffic. They validate their approach on a dataset captured from 20 apps with 355,235 requests and can achieve an average accuracy of 0.98.

Encrypted Video Classification The MPEG-DASH streaming video standard allows to stream in an encrypted way. Schuster et al. show that it is possible to classify the videos based on the packet burst patterns in the video even though the videos are streamed in an encrypted way [131]. A dash time series is created from captured TCP flows by aggregating the series into 0.25-second chunks. Then, they use a 1D CNN

architecture to capture the locally correlated patterns that the encrypted sequences of burst sizes contain. The model uses three convolution layers to avoid an information bottleneck in the beginning. The convolutional layers are followed by a max-pooling layer and two dense layers. Dropout is used to prevent overfitting. The attack is validated on video datasets from different platforms, and in their best setting they achieve a recall of 0.988 with zero false positives.

Physical Side Channels Industrial control systems are controlled by programmable logic controllers (PLC). Han et al. propose ZEUS, a DL-based method for monitoring the integrity of the control flow of such a PLC [54]. PLCs, when operate emit electromagnetic waves. Han et al. learn a program behavior model from spectrum sequences that are derived from execution traces. The execution traces are collected unobtrusively via a side channel by an inexpensive electromagnetic sensor. The spectrum sequences are modeled with stacked-LSTMs, which are well suited for modeling the sequential nature of the data. For validation, ZEUS is implemented on a commercial Allen Bradley PLC. ZEUS is capable of distinguishing between legitimate and malicious executions with an accuracy of 98.9% while causing zero overhead to the PLC.

Mobile User Authentication Sun et al. use sequences of characters and the accelerator information to identify users on a mobile phone [147]. They call their approach DeepService. To model the sequences they use a GRU [27], which is a variant of the LSTM that uses fewer parameters, which is important for a mobile phone setting. DeepService is evaluated on a dataset that recorded 40 users, which was collected by Sun et al. On this dataset, DeepService achieves san F1-score of 0.93.

Steganalysis of Speech Data Steganography is the discipline of hiding data in other data. Steganalysis attempts to discover and recover such hidden data. Lin et al. propose an RNN-based method to detect hidden data in raw audio data [84]. They segment speech data into small audio clips and frame the steganalysis problem as binary classification problem to differentiate between stego and cover. They train an LSTM-variant, called RNN-SM to classify on the raw data of the audio clips. Due to their stateful nature, LSTMs are well suited for modeling such data. RNN-SM is evaluated on two self-constructed datasets. One dataset contains 41 hours of Chinese speech data and the other contains 72 hours of English speech data. The accuracy depends on the language of the audio clips. In the best performing setting RNN-SM achieves an accuracy of up to 99.5%.

Audio tampering detection Adaptive multi-rate audio is a codec for compression of speech data that is commonly used in GSM networks. One way to manipulate such data is to decompress it to raw wave, modify the wave and re-compress it again. Luo et al. propose to extract features from compressed audio with a stacked under-complete

AE SAE to identify whether a speech file has been recompressed or not [87]. To detect double-compression, the extracted feature sequences are classified with a universal background model, namely a Gaussian Mixture Model. The features are extracted by first splitting the audio file to frames, then normalizing them, then compress them via a SAE and finally fine-tune them with a binary classification layer. Two SAE models are trained, one for single compressed files and one for double compressed files. The under-completeness of the AE motivates the extraction of the most salient features, thereby enables to differ between the two compression types. Luo et al. validate their approach on the TIMIT database, where they achieve a 98% accuracy.

Speech forensics Li et al. combine DL-based representation learning and spectral clustering to cluster mobile devices [83]. They use an under-complete AE to compress speech files that originate from different mobile devices. This under-complete AE compresses the speech files and thereby extracts representations that capture the artifacts of each type of mobile device. The AE was pre-trained with restricted Boltzmann machines. Pre-trained AE have been showed to work well for modeling audio data [59]. The representations that are learned by the AE are then used to construct Gaussian super-vectors. Finally, these Gaussian super-vectors are clustered via a spectral clustering algorithm to determine the phone type of the speech recording. Li et al. evaluate their method on three datasets, MOBIPHONE, T-L-PHONE, and SCUTPHONE and in their best configuration their method achieves a maximum classification accuracy of 94.1% among all possible label permutations.

2.3.2 – Spatial Data.

Breaking CAPTCHAs CAPTCHAs are a method for discriminating between automated and human agents [169]. CAPTCHAS achieve this discrimination by posing an agent a challenge that is easy to solve for humans but hard for computers. An example of such a challenge is to ask whether an image contains a store or not, which is easy to solve for humans but not for computers. From their inception, CAPTCHAs have been continually evolving in an arms race with smarter bots that attempted to break them. DL algorithms proved to be very successful in analyzing images, especially distorted and incomplete images, which eventually led to attacks on CAPTCHAs using DL methods.

Gao et al. attack hollow CAPTCHAs using a DL-based approach [43]. Hollow CAPTCHAs use contour lines to form connected characters. Their attack consists of two phases, a pre-processing phase, and a recognition phase. In the pre-processing phase, they repair the contour lines using Lee's algorithm, fill the contour lines using a flooding algorithm, remove the noise component, and finally remove the contour line. The pre-processed data is then used to train a CNN-based model, which is trained to classify the characters on the picture by numbering and classifying the strokes and stroke combinations in the CAPTCHA. The final characters are determined by a

depth-first tree search of a sequence that scored the highest predicted values. They selected a fully-connected version of LeNet-5's architecture [139], which provides partial invariance to translation, rotation, scale, and deformation and has shown to work well on handwritten digits. They validate their attack on multiple datasets and their success-rate for attacks on hollow CPATHCA's ranges from 36%-to 66%.

Algwil et al. use a similar CNN architecture and method to break Chinese CAPTCHAs [5]. Chinese CAPTCHAs consist of distorted Chinese symbols which need to be recognized by the agent. Algwil et al. address this challenge with a multi-column CNN [29]. A multi-column CNN is an ensemble of CNNs, whose predictions are averaged. The structure of each CNN is inspired by LeNet5, but it uses more and larger filter maps. Another difference is that the multi-column CNN uses max-pooling operations instead of trainable sub-sampling layers. This architecture also profits from the robustness of the CNNs to distortion, scaling, rotation and transformations. To attack the CAPTCHAs, they split each character into a set of radicals, i.e., composite elements, and classify input character to their radicals. Their best performing network is able to correctly solving a Chinese CAPTCHA challenge with 3755 categories with a test error rate of only 0.002%.

reCAPTCHA is an image-based CAPTCHA challenge developed by Google. Agents get a text-based assignment, and need to click all images that correspond to the assignment, for example: "Select all images with wine on it." Sivakorn et al. propose an automated reCAPTCHA breaker system [141]. Their approach uses reverse image search to derive labels for images, word vectors [98] to derive similarity between assignment and a web-based on service that is based on deconvolutional neural networks [194] to classify the images of the reCAPTCHA challenge. Deconvolutional neural networks use a form of sparse coding and latent switch variables that are computed for each image. These switch variables locally adapt the model's filters to learn a hierarchy of features for the input images, which enhance the classification performance on natural images. Their system is able to pass the reCAPTCHA challenge in 61.2% of the attacks.

Finally, Osadachy et al. propose a CAPTCHA generation scheme, that is robust to DL-based CAPTCHA attacks [111]. Their proposed CAPTCHA task is: given an adversarial image of one class, identify all images of the same class from a set of other images. An adversarial image is an image that has been modified with adversarial noise [150]. Such adversarial noise prevents DL architectures from correctly classifying images, but the images appear to be visually similar for a human. They generated such adversarial images using a CNN-F architecture [22]. CNN-F is a variant of AlexNet [78], which was the winning architecture on the ImageNet LSVRC-2012 challenge. AlexNet has popularized three main techniques for image analysis: ReLUs as activation functions, Dropout instead of regularization to prevent overfitting and overlap pooling to reduce the network size. CNN-F is a stripped-down version of Alex net that reduces the number of convolutional and fully-connected layers.

Biometrics from 2D Spatial Data Biometrics are biological traits of a person which can be used to derive keys for identification in digital systems. Many biometrics are based on 2D spatial data, for example, images of fingerprints, faces or irises. One of the key challenges of creating biometrics from 2D spatial data is to be able to distinguish between subtle elements of the raw data that allow the identification.

Faces are a standard way to identify people. One of the main challenges is that in real-life situations, images from faces will have distortions, different angles, and shades. Sun et al. proposed a DL-architecture that is well suited for such a task [148]. This architecture describes a CNN that consists of 4 convolutional layers, three max-pool layers and one fully connected layer, which yields an image representation. Additionally, they use local feature sharing in the third convolutional layer to encourage more diverse high-level features in different regions. Finally, the outputs of the last max-pool layer and the output of the last convolutional layer are shared inputs to the fully connected layer. This connection prevents an information bottleneck caused by the last convolutional layer and is essential for learning good representations. Their approach manages to achieve a 99.15% accuracy on the LFW dataset with 5749 different identities when their network was augmented with additional, external training data. Zhong et al. use a similar approach for the task of face-attribute classification [201]. They compare VGGnet [140] with FaceNet [129] and conclude that both are suitable to extract facial attributes. Instead of Iris as a biometric, Zhao et al. focus on the region around the eyes – the periocular region – for authentication [199]. They use AlexNet to classify the eyes and derive semantic information such as age, or gender using additional CNNs that are trained on the surrounding regions of the eye. Goswami et al. use a combination of an SDAE and a DBM to extract features from multiple face images which they extract from video clips and then use an MLP to authenticate people via their faces [49]. They pre-filter the frames with a high information content as input for the classification network. At a false-accept rate of 0.01, they achieve up to 97% accuracy when validated against the point and shoot challenge database, and 95% against the Youtube faces dataset.

Instead of directly classifying faces with a CNN for biometrics, Liu et al. propose to learn a metric space for faces [86]. Their approach is inspired by triplet networks of Schroff et al. [129]. Triplet networks learn a metric space by solving a ranking problem between three images that are represented by the same CNN. The parameters of such a model are learned by minimizing a triplet loss. Liu et al. pre-train their model on the CASIA Web-Face Dataset and evaluate CASIA NIR-VIS 2.0 face dataset, where they achieve a rank one accuracy of 95.74%. The second best approach achieved an accuracy of 86.16%.

Fingerprints are another standard way to identify people. One important feature to match fingerprint images is the local orientation field. Cao et al. frame the latent orientation field extraction as a DL-based classification task [19]. Given a fingerprint image, they predict the pixel probabilities of this image that a pixel belongs to a local orientation field. They base their CNN model on LeNet-5, but adjust the filter size and extend the model by dropout and local response normalization [78]. In addition

to that, they augment their training data with texture noise. They evaluate their method on the NIST SD27 dataset, where they achieve a pixel-class root mean square deviation of 13.51, which presents an improvement for all latent orientation fields of 7.36%.

Segmentation is the computer vision task of splitting an image into multiple parts so that they can be processed better. In biometrics, iris segmentation aims to separate the parts of an image that belongs to an iris from the remaining images, so that the iris can be used for biometric identification. Liu et al. propose a DL-based approach for biometric iris segmentation [85]. They use an ensemble of pre-trained VGG19-net [140], which is fine-tuned on two iris datasets (UBIRIS.v2, CASIA.v4), to perform a pixel-based, binary classification task. VGG-net [140] advanced the state-of-the-art by two novelties: they increased the depth of the network by using many layers of smaller, 3x3 filters and they used 1x1 convolution, to increase the non-linearity of the decision function without reducing the size of the receptive fields. Simonyan et al. evaluate their approach on the UBRIRIS.v2 and the CASIA.v4 datasets and achieved a pixel segmentation error rate of less than 1%. Qin and Yacoubi use a similar approach for finger-vein segmentation. However, they use a custom CNN instead of a pre-trained VGG19-net [120]. Their approach is validated on two finger-vein databases and achieves an equal-error-rate of 1.69 in the best setting.

Peoples' appearances are a form of soft-biometrics that can be used to identify people in surveillance videos. Zhu et al. uses multiple AlexNet-based CNNs to classify properties of people from surveillance images to identify them [203]. They use an AlexNet for a binary classification task for each of the desired attributes. They validate their approach on the GRID and VIPeR databases, and their method outperformed the baseline, an SVM based classifier, by between 6% and 10% percent of accuracy on average. Hand and Chellappa also classify attributes of faces. They frame the problem as multi-class approach and design a particular architecture for the task: MCNN-AUX [55]. The MCNN-AUX consists of two shared convolution-max pool layers for all attributes and one CO layer followed by two FC layers for each task. Additionally, the data of all attributes are used for classification. Gonzalez-Sosa et al. compare a VGG-net variant with commercial off the shelf face attribute estimators, and find that soft biometrics improve the verification accuracy of hard biometric matchers [46]. Another soft biometric is gait, i.e., how people walk. Shiraga et al. use a CNNs called GEINet to classify peoples gait [137]. The input to GEINet is gait energy images (GEI), a particular type of images that are extracted from multiple frames of images that show people's walk. GEINet is based in AlexNet [78]. Shirage et al. reduce the depth of the network because of the different nature of the task, that is to classify GEI the network needs to focus on subtle inter-subject images. During the evaluation, GEINet consistently manages to outperform baseline approaches by a significant margin; In their best experiment settings, GEINet achieves a rank-1 identification rate of 94.6%. Age, gender, and ethnicity are other common soft-biometrics. Narang et al. empirically validate that VGGnet is suitable for age, gender, and ethnicity classification [104]. Azimpourkivi et al. propose another soft-biometric

for authentication for mobile-phone pictures of objects [8]. They use an InceptionV3 CNN [8] to derive features from pictures a mobile phone user takes. The Inception architecture introduced inception modules, which consist of multiple convolutional filters of different sizes. These filters are concatenated, and allow the network to decided which filter size it needs at a given layer in the network to solve the task at hand. To construct images hashes that are not sensitive to subtle changes, they use a particular version of locality sensitive hashing which allows them to group similar image features next to each other. Azimpourkivi et al. demonstrate that their approach is robust to real image-, synthetic image-, synthetic credential- and object guessing attacks, and show, that the Shannon entropy of their soft biometric is higher than that of fingerprints. Additionally, unlike real biometrics, soft biometrics that are derived from object pictures can be easily exchanged.

People's voices are also biometric data that can be used to identify a person. Generally, audio data is presented in waveform, which is a form of sequential data. However, a common pre-processing step in speech recognition is to calculate the Mel-frequency cepstral coefficients (MFCC), which represents the speech waves as 2D spatial data. Uzan and Wolf combines MFCCs and an AlexNet-based CNN [78] to identify speakers [161]. They validate their approach on their own dataset with 101 persons and 4584 utterances and their method achieves an utterance classification accuracy of 63.5%.

Another challenge in biometrics is to determine whether a biometric is authentic or not, i.e., to detect it has been tampered with or not. Menotti et al. propose a DL-based framework for detecting spoofed biometrics [97]. At the heart of their framework is a CNN that performs a binary classification task on images such as iris scans, fingerprint scans or face images. The goal of this task is to distinguish between fake and real biometric images. They evaluate two different strategies, architecture optimization, and filter optimization. In their architecture optimization strategy, they leave the parameters random but change the architecture to fit the problem. In their filter optimization, they fix the architecture but try to find the optimal hyper parameter settings. To do so, they create spoofnet. Spoofnet is a CNN, which is an architecture that is based on Cuda-net [77]. Spoofnet is designed to handle subtle changes in data better because of the modified filter sizes. In contrast to their architecture optimization strategy, they train the parameters of spoofnet with SGD. They validate their approach on seven different databases and achieved excellent results compared to the state-of-the-art. Nougeria et al. use pre-trained VGG-19, which is fine-tuned on fingerprints, for distinguishing between fake and live fingerprints [107]. Their best performing model achieves a 97.1% test-accuracy. Instead of CNNs, Bharati et al. use supervised deep Boltzmann Machines [14] to detect the spoofing of face images. Their method outperforms an SVM on their custom datasets. Conceptually very similar to spoofing detection is biometric liveness detection.

Detecting Privacy Infringing Material Pictures may contain privacy-sensitive information, such as driver's licenses or other confidential material. Tran et al. propose a CNN-based method to detect such material [160]. They combine a modified version of AlexNet [78] with a CNN for sentiment analysis [189] to detect such material. They name this model PCNH. They use 200 classes from the ImageNet challenge [126] to pre-train PCNH, and then train PCNH on categories of sensitive material. They validate their approach on their own and a public dataset with an F1-score of 0.85 and 0.89, respectively. Yu et al. tackle a similar challenge with a hierarchical CNN [190]. They combine deep CNN for object detection with a tree-based classifier over the visual tree. The object identification focuses on detecting infringing privacy object, and the tree-based classifier combines them.

Steganalysis and Steganography Steganalysis is the process of analyzing data to find out whether secrets have been hidden in the data or not. One way to hide information is hiding it in images. One of the key challenges of detecting such hidden data requires to identify subtle changes in the data. Ye et al. propose a DL-based method for image steganalysis, where they use a deep neural network to detect whether information has been hidden or not [188]. Their proposed CNN architecture has an adaption to suit the data for the task at hand. First, the first three layers are convolutional layers only, which is important not to lose any information. Also, no Dropout or normalization is used. Furthermore, instead of Max-Pooling layers, they use mean pooling layers. In total, their network is ten layers deep. Apart from that, they use a truncated linear unit (TLU) activation function in the first layer, which reflects that most steganographic hidden data is between -1 and 1. The weights in the first convolutional layer are initialized with high-pass filters, which helps the CNN to ignore the image content and instead focuses on the hidden information. For training their network, Ye et al. adopt a curriculum learning scheme, which selects the order of the images that are used for training. This ordering is essential for detecting information that has been hidden with a low bit-per-pixel (BPP) rate. They evaluate their method on multiple steganographic schemes and BPP rates, and their approach consistently outperforms the baseline approaches. Their detection error rate ranged from 0.46 to 0.14. Zeng et al. propose to use the residuals and an ensemble of CNNs to detect steganography in images [195]. As input to their network, they use quantized, truncated residuals, with three different parameters for the quantization and truncation. The three different input sets are used to train three different CNNs. The CNN architecture is based on insights of the steganalysis network of Xu et al. [182], but they use ReLUs instead of TanH. Zeng et al. validate their approach on ImageNet and three steganographic algorithms, and their best performing model achieves a detection accuracy of 74.5%.

Authenticity of Goods and Origin Determination Sharma et al. applied AlexNet on the problem of separating real from counterfeited products [132]. To detect faked

material such as fake leather, they use microscopical images to build a model of the texture of the objects. Then, they use AlexNet to classify different types of material. Depending on the material, the proposed method reaches a test accuracy of up to 98.3%. Similarly, Ferreira proposes a DL-based method to determine the device that a printed document was printed with [39]. Printed documents differ from each other in very subtle ways, depending on the manufacture of that printer. To find these subtle differences, Ferreira et al. use a CNN architecture with two convolutions and two sub-sampling layers. They use three different represent the input in three different ways, raw pixel, median residual filter and average residual filter. These different input representations were either combined with an early or a late fusion strategy. Early fusion combines multiple representations of the same data into one input data vector, and late fusion combines the output of multiple CNNs. Ferreira et al. evaluated their approach on a dataset which consists of 120 documents and ten printers. Using the best overall settings, their method was able to attribute printers with an accuracy of 97.33%.

Forensic Image Analysis A problem law enforcement faces when searching suspects computer for illegal pornographic material is the vast amount of pictures that can be stored on a computer, many of which are irrelevant. In their work, Mayer et al. evaluate DL-based image classification services on their capabilities to identify and pornographic material from non-pornographic material [93]. They evaluate Yahoo's service, Illustration2Vec, and Clarifai, NudeDetect and Sightengine. They find, that when such services are used to rank such images on their not safe for work character, that, on average, relevant images are discovered on positions 8 and nine instead of position 1,463.

Watermarking Watermarks address the problem of data attribution. For example, watermarks are added to copyright protected material to identify the owner this material. Kandi et al. propose to use a convolutional neural AE to add an invisible watermark to image [72]. A CNN AE is an AE that has CNN as encoding and decoding networks.

Detecting Polymorphic Malware Nguyen et al. propose a method for detecting polymorphic malware from binary files via a CNN [106]. To be able to use CNNs, they first extract the control flow graph from the binary. They transform the graph into an adjacency matrix, where a vertex of the control flow graph is considered as a state, and three values describe each state: register, flag, and memory. The register, flag, and memory are mapped to the color of red, green and blue pixels. This representation allows describing similarity in a program state. Even if two variants of the same malware have different execution codes, the core malicious actions retain in the same area of such an image. The corresponding image will differ, and CNNs are highly suitable for detecting similarities between such images with losing spatial

correlation. Nguyen et al. evaluate different architectures on a set of polymorphic malware, and they find that the YOLO-architecture [121] is the best performing. The YOLO-architecture can detect polymorphic malware with an accuracy of up to 97.69%.

Detecting Website De-facement Website defacements are unauthorized changes to a website that can lead to loss of reputation or financial gain. Borgolte et al. use deep neural networks for detecting web site-defacements from website screenshots [15]. They train their model on screenshots from web-pages, which they automatically collected using a web-crawler. Since the screenshots are comparably large, they extract a window uniformly sampled from the center of the webpage. This extracted window is then compressed with a stacked denoising AE., and finally is classified to normal or defaced using an AlexNet-like network. They validate their approach on their own, large-scale defacement dataset and their method achieves a true positive rate of up to 98.81%.

Forensic face matching Forensic face-sketches are pictures of faces of crime suspects that are hand-drawn by an artist with help from instructions of eye-witnesses. Mittal et al. propose a DL-based approach to match forensic face-sketches with images of people [101]. Their approach consists of two phases. First, they learn representations from face images in an unsupervised way. Their model for representation learning is a combination of stacked denoising AE [166] and deep belief networks [60]. They first train this model on real people's faces and fine-tune it on the face sketches. Secondly, they train an MLP in a directed way to predict a match-score between pairs of images. The input to this MLP is the concatenated learned representation for the real image and the sketch image. They evaluate their approach on multiple databases achieved an accuracy score of 60.2%, which is a significant improvement to the second-best approach, which matched only 50.7% face-sketch pairs correctly. Galea and Farrugia tackle the same task of matching with a combination of VGGnet and a triplet network, which is called DEEPS [42]. They validate DEEPS on the UoM-SGFS 3D sketch database and achieved a Rank-1 matching rate of 52.17%.

Wang et al. use the representation-learning capabilities of neural networks to increase the performance of Commercial Off-the-Shelf (COTS) forensic face matchers [172]. They use a pre-trained AlexNet [78] to derive a vector representation of face images of the last fully connected layer before the classification layer. They use cosine similarity as a distance metric between faces and rank the images using probabilistic hypergraph ranking [65]. Then, they select the top k images as an input to the COTS PittPatt matcher, where k is a hyperparameter. They validate their approach on a combination of three databases, the Labeled Faces in the Wild (LFW), the Weakly Labeled Faces (WLFDB) and the Pinellas County Sheriff's Office (PSCO) database and their approach consistently scores a higher mean average precision than the PittPatt matcher alone.

Spoofing Detection Czajka compares hand-crafted features to learned features for suitability on the task of detecting a spoofing attack with rotated biometrics [31]. The final classification is conducted with an SVM in both cases. They evaluated their approach on multiple datasets from different sensors and found that the hand-crafted features performed better in the cross-sensor test and the learned features performed better in the same-sensor test.

Mobile Iris Verification Zhang et al. explore the suitability of learned representations for biometric iris verification on a mobile phone [197]. They use a model that is based on the ideas of Zagorykyo et al., who have proposed to compare two image patches by a 2-channel CNN that takes as input a pair of images for each channel [192]. This model has one output and is trained on a hinge-loss to regress on the similarity between the two patches. Zhang et al. combine two types of representations for their iris-verification approach: the one derived by a 2-channel CNN and representations derived by optimized ordinal measures. They demonstrate that the FER rate for iris verification can be significantly decreased by combining these two features.

Face Verification Gao et al. address the task of face identification by using a supervised deep, denoising stacked AE. This model learns face representations that are robust to differences such as illumination, expression or occlusion [44]. Their architecture consists of a two-layer denoising AE. The input was a canonical face image of a person, i.e., a frontal face image with a neutral expression and normal illumination, and the "noisy" input to reconstruct were images of the same person. They use their AE to extract face features, and sparse representation-based classification for the face verification task. On the LFW dataset, they achieve a mean classification accuracy of 85.48%. Bharadwaj et al. applied a similar approach on baby faces [13]. They use stacked denoising AE to learn robust representations of baby-faces, and a one-shot, single-class support vector machine to classify the baby faces. Their system achieves a verification accuracy of 63.4% with a false accept rate of 0.1%. Noroozi et al. propose to learn representations for biometrics with an architecture called SEVEN [108]. SEVEN combines a convolutional AE [91] with a Siamese network. The combination of these components allows learning general salient and discriminative features of the data. Also, it enables the network to be trained in an end-to-end fashion.

2.3.3 – Structured Data.

Authorship identification Alsulami et al. propose to identify the authors of source code by features that are extracted by an LSTM from the source code [6]. To achieve this, they first generate the abstract syntax tree (AST) from the source code. Then, this AST is traversed depth-first to create a sequence of nodes. Each node is embedded via an embedding layer, and a model is learned from such sequences of nodes via a bi-directional LSTM. Alsulami at al. evaluate their method via code obtained from

public source code repositories, and achieve an author classification accuracy of up to 96%.

DL-based PUF Verification Physically Unclonable Functions (PUFs) can be used in an authentication system in a challenge-response based protocol. Yashiro et al. propose DAPUF, a strong, arbiter PUF-based system for authentication of chips, that can be used to authenticate them. They show that their system is resistant to DL based attacks [187]. To carry out these attacks, they use a stacked denoising AE followed by a FC layer to differentiate between fake and genuine PUFs in a binary classification task.

Network Intrusion Detection Network intrusion detection is the task of analyzing network traffic data or data from other components of a network to identify malign actions. Osada et al. propose a network intrusion detection method that uses latent representations of network traffic to identify malign actions [110]. Osada et al. propose a semi-supervised version of VAEs, the forked variational AE to address this task. The forked VAE learns a representation from the traffic, and subsequently the mean of the latent space defined by the VAE is used as input to a classifier that , and predicts whether an example is benign or not. The error is back-propagated and combined with the VAE reconstruction error. Details on feature extraction and how to overcome problems of training a discrete VAE are omitted. Osada et al. evaluate their approach on the NSL-KDD and Kyoto2006+ datasets. Adding 100 labeled examples increased the absolute recall-rate by 4.4% points the total false-positive rate by 0.019%. Similarly, Aminanto et al. propose to learn features from network data to detect impersonation attacks [7]. They extract the features using a sparse deep AE from the existing network data features, i.e., they compress the already complex features. They use this learned frame representation to learn a feed-forward MLP. They evaluate their approach on the Aegean Wireless Network Intrusion Detection Dataset (AWID) and achieve a per-frame classification accuracy of 99.91% and a false positive rate of 0.012.

Drive-by Attack Detection Drive-by attacks are attacks that infect clients that visit a web page by exploiting client-side vulnerabilities. Shibanara et al. propose a method for detecting such attacks by classifying the sequences of URLs into benign and malicious sequences [135]. They extract 17 features from each of the URLs that are loaded when a client visits a web page, and classify these sequences using a CNN. Shibanara et al. propose an Event Denoising CNN (EDCNN) to suit the data at hand. This EDCNN has an allocation layer, which rearranges the values of the input layer to convolve over two URLs whose order is similar. Additionally, they use a spatial pooling layer to summarize sequences of different length [56]. They evaluate their approach on a data set that they collected using a honey client and which consists of 17,877 malicious and 41,127 benign URL sequences. On this dataset, an EDCNN

achieves a false-positive rate of 0.148 and an F1-score of 0.72, which outperforms a regular CNN, which achieves a false-positive rate of 0.276 and an F1-score of 0.59.

Cross-Platform Binary Code Similarity Detection Cross-Platform binary code similarity detection aims to identify whether two pieces of binary function code from different platforms describe the same function. To address this problem, Xu et al. propose to learn a metric space for functions [185]. They do so by combining a Siamese network [18] with a Structure2Vec model [34]. That is, they treat the function code as a graph, hence chose Structure2Vec as graph embedding network in a Siamese learning setting. Xu et al. demonstrate on four datasets that their approach outperforms state-of-the-art approaches by large margins. Additionally, they show that the embedding was learned 20x faster and that the embedding maps binary codes of the same function close to each other, and functions from different functions further apart.

2.3.4 – Text Data. Here we survey applications of DL solutions on text data.

Password guessing attacks Melicher et al. propose DL-based password guessing attacks, which they use to measure the strength of passwords [94]. Their idea is inspired by works of Sutskever et al., who have successfully demonstrated that RNNs could be used to build language models for text, which in turn could also be used for generating text. Melicher et al. use fine-tuned LSTMs [71] to learn a model for the characters of passwords. This model can be used to predict the likelihood of a given password, and given the likelihood, they calculate the strength. They train their model on a large list of publicly available passwords and show, that a DL-based password model performs better at predicting than Markov chains or Context-Free Grammars.

Creating and Detecting Fake Reviews E-commerce sites sell products which partly derive their reputation via user reviews. Yao et al. use a DL-based language model to create fake reviews as well as a defensive model [186]. Their fake review language model is a word-based language model based on LSTMs [62]. LSTMs have been shown to model statistical properties of texts well [24, 38, 149]. They use a large text corpus, the Yelp review database for training, and fine-tune the generated reviews by a simple noun replacement strategy. This noun replacement strategy refines the review to the context of the review. Yao et al. experimentally show, that about 96% of the fake reviews pass automated plagiarism detection, and human recall and precision are 0.160 and 0.406. To defend against such automated attacks, Yao et al. propose to use a character-based language model, as they find that generated texts have a statistically detectable difference in character distribution to text written by real people. Their defense achieves a prevision of 0.98 and a recall of 0.97.

Log analysis System logs keep track of activities happening on a computing system. In case of system failures or attacks, system logs can be used to debug such failures or reveal knowledge about an attack. Du et al. propose a neural sequence model-based approach for anomaly detection in system logs called DeepLog [36]. Their neural sequence model consists of a stacked LSTM [62]. To build the sequence model, they manually parse lines to a specific event IDs to construct sequences of events. Such sequences describe workflows on the analyzed system. The normal workflow model is built from a set of workflows. The model continuously tries to predict next event IDs, and if they predicted event ID is above a certain mean squared error for the actual error of the event, the system raises an alarm. Du et al. integrate user feedback to update the model if a false positive has been detected. DeepLog is evaluated on two large system logs, and it achieves an F1-score of up to 96%.

Thaler et al. propose an unsupervised method for signature extraction from forensic logs [157]. This approach groups log lines based on the print statement that originated their log lines. Their proposed method combines a Neural language model [38] and with a clustering method. They train the neural language model as an RNN-autoencoder to learn a representation of the loglines, which captures the complex dependency between mutable and immutable parts of a logline. These representations are then clustered. They evaluate their approach on three datasets, one self-created and two public system logs with 11,023, 474,796 and 716,577 log lines. Their method clusters log lines with a V-measure of 0.930 to 0.948.

2.3.5 – Multi-Modal Data.

Cyber bullying detection Zhong et al. propose a DL-based method to detect bullying in cyber space [200]. To do so, they build a model from images and comments of Instagram. For classification of the images, they use a pre-trained version of AlexNet, which they fine-tuned using their dataset. For representing text, they used word2vec [99] and other, shallow representations such as bag-of-words. Zhong et al. experimented with different configurations and combinations and conclude, that the title of the post is one of the strongest predictors of cyber bullying.

ECG Biometrics With cheap mobile sensors available, ECG can be used as a modality for biometrics. Da Silva Luz et al. propose a method for ECG-based biometrics, which treats the ECG data as both, raw sequential data and spatial data [32]. They derive the 2D spatial data from the raw ECG data by calculating a spectrogram. They build two models from these two data modalities, a 1D, and a 2D CNN. The outputs of these two networks are merged using either a sum, a multiplication or a mean rule. To capture a whole ECG cycle, both CNN models have large first layer convolutions. The method is evaluated on multiple datasets. The fusion of the two model results consistently increases the accuracy of the model.

2.3.6 – Security Properties of DL Algorithms.

Privacy of DL Algorithms Although DL algorithms are successful in many domains, in some areas such as healthcare their uptake has been limited due to privacy and confidentiality concerns of the data owners. Shokri and Shmatikov proposed a system that enables multiple parties to jointly learn and use a DL model in a privacy-preserving way [138]. The core of their idea is to devise a novel training procedure: distributed selective stochastic gradient descent (DSSGD). In this procedure, each participant downloads the global model parameters to their local system. On their local system, they compute the gradients using stochastic gradient descent and submit a selection of the gradients to the server. Either the largest k gradients select the submitted gradients, or randomly subsample k gradients that are above a certain threshold t, where k and t are hyper-parameters. Their approach is not geared towards a particular DL architecture, but they validate their approach on two datasets MNIST and SVHN, with two models, an MLP and CNN. They demonstrate that the models learned by their approach guarantee differential privacy for the data owners without sacrificing predictive performance. Instead of calculating the privacy-loss per parameter, Abadi et al. calculate the privacy-loss per model to select the parameter updates and ensure differential privacy for stochastic gradient descent [2]. The calculation of privacy-loss per model is beneficial in scenarios where a whole model will be copied to a target device, for example in the context of mobile phones. The main components of their approach are a differentially private version of the SGD algorithm, the moments accountant, and hyperparameter tuning. Abadi et al. experimentally show that differential privacy only has a modest impact on the accuracy of their baseline. Phan et al. propose a privacy-preserving version of a deep AE that enforces ϵ-privacy by modifying the objective function [118]. They achieve ϵ-privacy by replacing the objective with a functional mechanism that inserts noise [196]. Additionally, Phan et al. conduct a sensitivity analysis. Hitaj at al. develop an attack on distributed, decentralized DL and show that data of honest participants is leaked [61]. Their attack uses generative adversarial networks (GANs) [48] and assumes an insider at the victim's system. The insider uses a GAN to learn to create similar objects for a particular of the victim's classes and injects these images into the learning process under a different class. As a result, the victim has to reveal more gradients about the original class, thereby leaking information about the objects. Phong et al. demonstrate that the scheme proposed by Shokri and Shmaktikov [138] can leak data to an honest-but-curious server [119]. Also, they propose to address this by combining asynchronous DL with additively applied homomorphic encryption. In their evaluation setting, the computational overhead for adding homomorphic encryption to asynchronous DL is around 3.3 times the computational power of not using encryption. However, no information is leaked to the server.

Adversarial Attacks on DL Algorithms In their work, Szegedy et al. describe that is easy to construct images that appear to be visually similar to a human observer, but will cause a DL model to misclassify [150]. This approach requires knowledge about the parameters and architecture of the model. As an extension to that, Papernot et al. experimentally show that it is also possible to create adversarial attacks with only knowing the labels given an input [115]. Their black-box attack has an adversarial success rate of up to 96.19%. Further, Papernot et al. introduce a class of algorithms that craft adversarial images [113], i.e., they propose an algorithm that poses a threat to the integrity of the classification. Their algorithm used the forward derivative and the saliency map to craft adversarial images and evaluated on feedforward deep neural networks and at test time they achieve an adversarial success rate of 97%. In addition to that, Papernot et al. propose a defensive mechanism against adversarial attacks = defensive distillation [114]. Defensive distillation is a more robust variant of stochastic gradient descent, where additional knowledge about training points is extracted and fed back into the training algorithm. They experimentally validate that defensive distillation can reduce the effectiveness of adversarial attacks to 0.5%. Shen et al. show that poisoning attacks are possible in a collaborative DL setting with a success rate of 99% [134]. As a defense, Shen et al. propose a collaborative system – AUROR – that detects malicious users and corrects inputs. Malicious updates are detected by gradients that follow an abnormal probability distribution. AUROR reduces the poisoning attacks success rate to 3%. Meng et al. propose MagNet, another defense system against gray-box adversarial attacks that is independent of the target classifier or the adversarial example generation process [96]. MagNet consists of two components, a detector network, and a reformer network. The detector network is an AE that attempts to distinguish between real images and fake images, and the reformer network, which is also an AE, attempts to reconstruct the test input. Both, the detector and the reformer network are trained with Gaussian noise. Examples with larger reconstruction error of the detector network are rejected, and if they are not rejected, they are passed through the reformer to be moved closer to the original manifold to disturb their adversarial capability. During the evaluation, Magnet shows a minor reduction in classification accuracy due to the loss that is caused by the reformer network, but it is very effective to prevent adversarial attacks.

CHAPTER 3

Neural Signature Extraction

The first two challenges that we want to address in this thesis is the lack of supervision and the integration of domain knowledge. To this end, we selected the concrete problem from forensic log analysis that is representative of these two challenges - the problem of signature extraction and its related problem of log clustering.

Signature extraction is a critical preprocessing step in forensic log analysis because it enables sophisticated analysis techniques to be applied to logs. These logs are typically large, and their structure varies a lot. Currently, most signature extraction frameworks either use rule-based approaches or hand-crafted algorithms. Rule-based systems are error-prone and require high maintenance effort. Hand-crafted algorithms use heuristics and tend to work well only for specialized use cases. Machine learning methods bear the potential automate this process. However, supervision in the form of labeled data is typically not available.

Hence, before address the problem of lack of supervision, we want to verify our assumption that machine learning models can learn to capture the complex dependencies of log lines. We consequently ask the research question of how to automate signature extraction of forensic logs with labeled data (see Research Question 1).

In this chapter, we present a novel method that is based on a neural language model to extract signatures from forensic logs. This language model learns to classify mutable and non-mutable parts in a log message. We use this information to extract signatures. Neural language models have shown to work exceptionally well for learning complex relationships in natural language text. The remainder of the chapter is structured as follows: In Section 3.1 we will introduce the problem of signature extraction and explain its relationship to log clustering. In Section 3.2 we proceed by proposing our method, which uses a neural language model to identify the mutable and non-mutable parts of a log message. We then provide empirical evidence that this method can be used to solve the task (Section 3.3), and that it outperforms baseline approaches (Section 3.3).

3.1 — Introduction

Log signature extraction is the problem of finding the set of signatures from a given set of loglines. The extracted signatures can be used for matching signatures to loglines,

which is closely related to log clustering [151]. The key difference to clustering is that signature extraction is concerned with finding accurate cluster descriptions, i.e., signatures, as opposed to finding good clusters according to a given distance metric. In this section, we formalize the concepts of log signature matching and log signature extraction. We first define the concepts of a log message, a signature, signature matching and signature extraction. We then provide an example how signatures can be used for matching.

Let $|s|$ denote the number of characters or length of an arbitrary, finite string of characters s. A log message X is a finite string with length l_x. X consists of n_x parts P_x=$\{p_1 \dots p_{nx}\}$, where $0 \leqslant n_x \leqslant l_x$, $|p_{xi}| > 0 \; \forall p_{xi} \in P_x$ and $\sum_n |p_n| = l_x$. p_{xi} denotes the $i-th$ part of P_x.

Similarly, a signature S is a finite string with length l_s, with n_s parts P_s=$\{p_1 \dots p_{ns}\}$.

Parts can be either mutable or non-mutable. A mutable part consists only of mutable characters, and a non-mutable part consists only of non-mutable characters. Furthermore, mutable parts are always neighbored by non-mutable parts, unless they are at the beginning or the end of a log message or signature. To describe mutability of parts, we define a function m that maps mutable parts to 1 and non-mutable parts to 0.

$$m(p_{ns}) = \begin{cases} 0 & \text{if } p \text{ is non-mutable} \\ 1 & \text{if } p \text{ is mutable} \end{cases} \tag{3.1}$$

Next, we define the condition when a log X matches a signature S. Given a log message X, its parts P_x, a signature S, its parts P_s and a mapping function m. Then X matches S, if all non-mutable parts at the same position are equal. More formally, a match is defined, when for any $P_{xi} \in P_x$, $P_{sj} \in P_s$ with $i = j$ the following two conditions hold:

1. $m(P_{xi}) = m(P_{sj})$, i.e., each part at the same position has the same mutability;
2. $P_{xi} = P_{si}$ when $m(P_{xi}) = 0$ or $m(P_{sj}) = 0$, i.e., the part at the same position is equal if not mutable.

Using this match definition, we can define a simple scoring function for matches.

$$match\_score(X, S) = \begin{cases} 1 & \text{if } X \text{ matches } S \\ 0 & otherwise \end{cases} \tag{3.2}$$

Finally, we define the signature extraction problem as follows. Given a set of log messages $\mathcal{D}$, find k message signatures $S = \{S_1, ...S_k\}$ and a k-partition $C_1, \dots, C_k$ of $\mathcal{D}$, each of which contains a subset of $\mathcal{D}$ and indexed by j, to maximize the objective function $J(S, D)$:

$$J(S, D) = \sum_{i=1}^{k} \sum_{X_j \in C_i} match\_score(X_j, S_i). \tag{3.3}$$

Table 3.1: Example signature matching. To match, all non-mutable parts should be the same.

X	"New session c1 of user lightdm."				
Source	"New session %s of user %s."				
Correct signature parts					
P_s	"New session "	φ	" of user "	φ	"."
$m(P_s)$	0	1	0	1	0
A match.					
P_{1x}	"New session "	"c1"	" of user "	"lightdm"	"."
$m(P_{1x})$	0	1	0	1	0
A mismatch.					
P_{2x}	"New "	"session c1"	" of user "	"lightdm"	"."
$m(P_{2x})$	0	1	0	1	0

In Table 3.1 we present a simple example of signature matching. In this example, X represents a log message of a system log, Source describes the definition of this log message in the original source code, and P_s describes the parts of the correct signature S. We omitted S, as for the matching problem we only need to see whether the non-mutable parts to be at the same position. Furthermore, "%s" in Source denotes a placeholder for a string, and φ represents a place holder character in the signature. The function m describes whether the parts are mutable or not.

P_{1x} describes the parts of a message X that, in combination with m match signature S, because all parts that are non-mutable are equal and at the same position.

P_{2x} describes an example of logline parts that do not match signature S, because the first part of P_{2x}, "New" does not match the characters of the first part of P_s, "New session".

The signature extraction problem in this example is given X, find the parts of the signatures P_s and define which ones are mutable and which ones are not.

Given as set of signatures, a forensic investigator may analyze forensic logs more efficiently. For example, having such signatures would allow the investigator to extract certain variable from all log lines that originated from that particular signature. Or, an investigator could label a sequence of log lines, and obtain all similar sequences

Extracting signatures is challenging, because the variable parts may be of varying lengths, often hundreds of tokens, and may have different contents, a potentially large alphabet, and forensic logs may contain many million entries. If we treat the signature extraction as the problem of finding the Longest Common Subsequence, then for two sequences with an alphabet size of two the problem is considered to be NP-complete, and the general case is estimated to be intractable [88]. Since the signature extraction involves more than just finding the Longest Common Subsequence and log lines have a potentially very large alphabet, we argue that the signature extraction problem is also at least NP-complete.

3.2 — Method

We propose to use a neural and probabilistic framework to address the signature extraction problem. In particular, given a set of logs, we want to find the set of signatures that maximizes our goal objective (see Equation 3.3). Since in our case we know the set of correct signatures, we want to create a method that finds the most correct signatures. In this section, we will fist motivate our choice for an approach based on a neural language model. Then we describe two approaches that elaborate how signature extraction can be achieved using a neural language model. After that, we introduce the architecture of this model.

Our primary assumption is that log messages are partially structured natural language. This structured language follows specific complex, probabilistic rules that help us to detect mutable and non-mutable parts. Furthermore, we assume that a neural language model can capture these rules. We motivate our choice for a probabilistic neural framework by the recent advances and successful applications of neural networks in the NLP domain. Neural networks were able to capture the hidden structure of natural language well enough to deliver superior performance over rule-based approaches for natural language-related tasks. The hidden structure of natural language is arguably more difficult to capture than in partially structured logs. Therefore, we assume that neural models are well-suited for understanding the hidden structure of logs files given enough data.

We describe two similar approaches that address the signature extraction problem. Both use a character-based neural language model with the same architecture. However, they use this model differently. On a high level, both our signature extraction approaches consist of three steps: predict, correct / merge and squash. We depict an overview of these steps in Figure 3.1. The approaches differ by the type of prediction, i.e., Approach I tries to reconstruct the original character, and Approach II tries to estimate whether it is a mutable or non-mutable character.

During the prediction step, Approach I uses the character-based language model to determine the mutable and non-mutable characters of a log message. Mutable characters are replaced with φ, and non-mutable characters are kept the same. Then, in the correct step, we replace wrongly predicted, non-mutable characters using the characters of the original log message. We can correct wrong predictions because we know that a φ character should either replace the predicted characters or be identical to the original character. Finally, in the squash-step, we merge multiple φ following each other to a single φ because we assume that no two mutable parts are allowed (see Section 3.1). All characters between two φ are considered a non-mutable part.

To illustrate Approach I, consider an example log message with the characters "*a97b*". First, the model predicts the output for each character $a\varphi\varphi a$. This prediction means that the model *believes* that the characters "a" and "b" are non-mutable and the characters "9" and "7" are mutable. Next, since the model mispredicted the last character, we replace it with the original value "b". The result of this operation is "$a\varphi\varphi b$". In the final step, we merge the two mutable characters φφ to φ. This

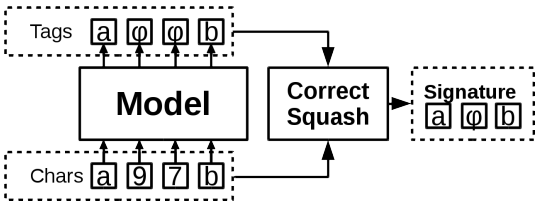

Figure 3.1: Overview Approach. A model predicts whether characters of a logline belong to a mutable part. These predictions are used to extract a signature.

operation results in the signature for this logline: "$a\varphi b$."

In Approach II, the model attempts to predict what type each character is, namely 1 if it belongs to a mutable part and 0 otherwise. The output of the first step is a sequence of tags, one for each character. Next, we merge the characters of with their tags. That is, we replace each character with φ if the tag was 1 and leave the character unchanged otherwise. The final squashing step is the same as in Approach I.

To illustrate Approach II, consider an example log message with the characters "$a97b$". First, the model predicts the mutable and non-mutable characters. The result of this operation is four tags "0110", which means that the model "believes" that characters "a" and "b" are non-mutable and characters "9" and "7" are mutable. Next, we merge the characters from the original log message with the tags. That is, we replace a character of the original string with a special φ if it was tagged mutable and do not change it otherwise. The merge operation results in "$a\varphi\varphi b$". The final squashing step is similar to the squashing step in Approach I.

Figure 3.2 depicts an overview of the architecture of our character-based neural language model. The objective of this model is to directly maximize the log-likelihood of a sequence of tags T given a sequence of characters from a logline X. Maximizing the log likelihood of a sequence of tags can be achieved by minimizing the Kullback–Leibler (KL) divergence between the predicted tag distribution and the correct tag distribution, which in our case is equal to minimizing the cross-entropy between the two distributions because the entropy of our correct tag distribution is zero.

We construct the architecture of our model using five layers: an embedding layer, a bi-directional long short-term memory (bi-LSTM) layer, a dropout layer, feedforward layer, and softmax layer. This model is suitable for the classification task at hand, as it respects the sequential nature of the input data and allows to model the dependencies in both directions. We detail the hyper parameters for this architecture in Section 3.3.

The embedding layers transform the categorical character input to a vector form. That is, it maps each character to a row of an embedding matrix. The length of the embedding row vector can be chosen as a hyperparameter, the number of rows depends on the number of different input characters. We chose to use an embedding matrix because it allows us to create a dense representation of our characters. We chose this embedding layer since we do not know the number of characters that will

occur in a log file in advance. For a new character, we have to add another row in the embedding layer. If we were to use one-hot representations for the input characters, we would need to fix the number of input characters up front or change the model every time that we encounter a new character.

The embedded characters are fed into a bi-directional LSTM [52]. A bi-directional LSTM consists of two independent LSTMs, one that processes the inputs in "forward" order one in reverse or "backward" order. Both LSTMs output sequences of vectors. The output of both networks is concatenated. Bi-LSTM neural networks allow each prediction to be conditioned on the whole past and future context of a sequence. The capability to use past and future context have proven to be useful in different NLP tasks. Bi-directional LSTMs are composed and trained in a similar way as regular LSTMs (see Chapter 2, Section

Then, during the training of our model, we apply dropout to the concatenated results of the bi-LSTM layer to prevent overfitting [144]. The dropout layer stochstically sets outputs of the previous layer to zero. These stochastic output drops prevent the units from co-adapting too much. The fraction of inputs that should be multiplied with zero can be chosen as hyperparameter.

Finally, we use a fully connected, feedforward neural network layer with a softmax activation function to predict our tags [47]. The number of neurons in the feedforward layer is set by the size of the concatenated output of the bi-LSTM layer. Adding a feedforward neural network adds more predictive capacity to our model. This additional capacity allows capturing more complex relationships between the characters of our input characters and output tags.

We optimize the categorical cross-entropy loss on the characters and annotations using *rmsprop*. *rmsprop*is a variant of stochastic gradient descent where the gradients are divided by a running average of their recent magnitude [51]. This adjustment of the parameter updates speeds up the mini-batch learning and has recently been successfully applied to other works on sequential data e.g., [183].

3.3 — Experiment Setup

In this section, we report on the experiments that we have conducted to assess the performance of our approaches. The source code and the datasets of the experiments are available on GitHub[1].

For our neural language model based approaches we have evaluated the character based tagging performance using 5-fold cross-validation. This evaluation does not make sense for our baseline approaches since they operate on a word basis.

Furthermore, for all the approaches we evaluated two measures: i) the signature extraction performance and ii) the clustering performance. The signature extraction evaluation assesses the correctness of the extracted signatures the number of missing signatures. The clustering performance measures the correct grouping of log lines using the extracted signatures.

[1] https://github.com/stefanthaler/2017-fnlm-experiments-supervised

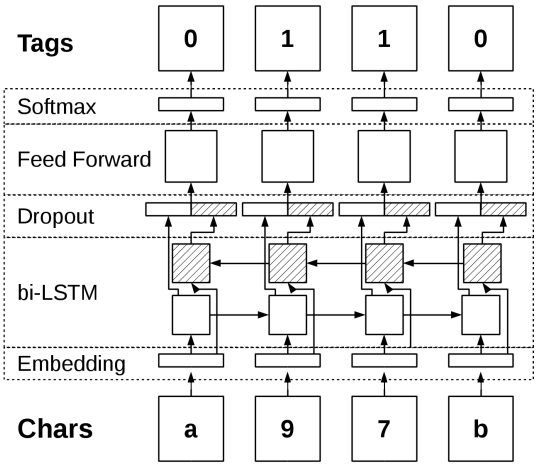

Figure 3.2: Model architecture. Sequences of characters are used as input. The mutability for each character is predicted.

Dataset Our framework attempts to solve the signature extraction task as a supervised learning problem. Therefore, we created a dataset of annotated log lines. To create a forensic log, we set up a virtual machine and installed a Linux operating system. On this virtual machine, we conducted a set of actions. We used the open source log2timeline tool[2] to extract the forensic super log. Next, we extracted the plain text log messages of this forensic log and removed duplicate lines. We then annotated each character of these cleaned log messages as either mutable or non-mutable. We ensured that our tagging was correct by analyzing the source code of the software that emitted this logline. In most cases, the source code reveals the mutable and non-mutable parts of a log message unambiguously.

In total, our training dataset consists of 950 log lines with an average of 77 characters, which means our training dataset consists of 73540 annotated characters for training. We used this training dataset for measuring the character-based tagging performance. We also use this training dataset to train the models that produce the signatures for measuring the signature extraction and clustering performance. Our signature extraction and clustering dataset consist of 200 log lines with an average of 73 characters per line. The test dataset contains 81 signatures.

Baseline We use LogCluster [163] and IPLoM [89] to derive our baseline. For LogCluster, we used the authors' implementation[3]. For IPLoM we used the imple-

[2]https://github.com/log2timeline/plaso/wiki/Using-log2timeline
[3]http://ristov.github.io/logcluster/logcluster-0.08.tar.gz

mentation[4] that was published by He et al. [58] in their comparative study of log parsers.

We chose these approaches because SLTC [162] and IPLoM scored highest in a direct comparison [58] and LogCluster is an improved implementation of SLTC. Moreover, both approaches can find the number of signatures as well as the signatures. This capability is crucial since we assume that the number of signatures is not known in advance. For IPLoM we use the default settings of the implementation. These settings scored the highest in the performance comparison by He et al. [58]. For LogCluster we empirically determined a support threshold of three and a word frequency of "0.9". processing

Model training We derived training sequences by creating sliding windows over log-character sequences. The sliding windows have the same length as the LSTM layer. We post-padded training sequences that were shorter than the sliding window size and ignored padding via masking during the training and prediction phase.

For Approach I, the best performing model was a bi-LSTM unrolled for 25 time steps, a cell size of 512 and a character embedding size of 128. For Approach II, the best performing model was a bi-LSTM unrolled for 20 time steps, a cell size of 128 a character embedding size of 64.

We trained the models of both approaches for 15 epochs and selected the best model based on validation data. The mini-batch size was set to the number of training examples divided by 200. For all experiments, we used *rmsprop* to calculate gradient updates, a dropout of 0.7 to prevent overfitting. Furthermore, we used a learning-rate of 0.0005, ρ of 0.9, ϵ of $1e^{-}0.8$ and a learning-rate decay of 0.05.

We determined all hyperparameters for training empirically using random search, which is a common strategy for neural network-based approaches [12]. Also, we only report the best-performing model. However, here we discuss empirical lessons that we learned from applying different architecture choices. First, we found that the LSTM consistently performed slightly better regarding classification accuracy than a GRU [24], without having any significant performance increase regarding training speed, which motivated our choice for the LSTM. Next, we found that a bi-directional LSTM outperformed a single LSTM with similar capacity. That is, a bi-directional LSTM with two LSTMs of size 128 would out-perform a single layer of 256. Also, we found that the bi-directional LSTM outperformed a two-layer LSTM of the same capacity, i.e., a bi-directional LSTM with two LSTMs of size 128 would. This finding was consistent for both approaches, although the classification task for Approach I was much more challenging than the one for Approach II, thus a network with larger capacity was needed in general.

Adding more layers of LSTMs did not yield any accuracy increases, and neither did add multiple layers of bi-directional LSTMs. Hence, we stayed with a single bi-

[4] https://github.com/cuhk-cse/logparser/tree/20882dabb01aa6e1e7241d4ff239121ec978a2fe

directional LSTM layer. Adding more capacity to the LSTM layers generally increased the overfitting, i.e., results on the training data were better than the results on the hold-out set. However, we found that having a model with slightly larger capacity and then adding dropout would

The learning-rate decay increased the accuracy performance in the later stage of the training. Increasing the learning-rate decay by a factor of 10 slowed down the convergence speed, and reducing the learning-rate decay by a factor of 10 caused a reduction of the performance gains at the later stage.

As an alternative to *rmsprop*, we evaluated Adam [74] and vanilla SGD and SGD with momentum. However, for our scenario *rmsprop*converged faster than the alternatives. Our best performing learning rate was related to the mini-batch size. If we trained with a larger mini-batch size, we had to reduce the training rate, and vice versa. Using smaller mini-batches slowed down the training considerable due to the overhead of loading the examples to the GPU, whereas large mini-batches would case a memory-overflow of our training hardware.

Evaluation metrics We evaluated the per-character tagging performance by calculating the accuracy of correct tags using 5-fold cross-validation on our training dataset. K-fold cross-validation is a common measure to estimate the test-error of a machine learning problem.

For evaluation purposes, we treat the signature extracting problem as information retrieval problem. Standard metrics for evaluating information extraction problems are precision, recall and F1 score. Precision measures the fraction of extracted signatures that are correct. Recall measures the fraction of correct signatures that are extracted, and the F1 score is a harmonic mean between precision and recall.

Finally, we evaluated the clustering performance for our approaches and the baseline. To compare two clusters, we consider them to be sets of log lines. Let G be the set of log lines that were clustered by a signature of the ground truth, and E the set of log lines that were clustered by a signature of the evaluated approaches. Then we consider them to be equal if $G \cap E = E$. We consider them to be a subcluster if $E \subset G$. We consider errors to be log lines in clusters where they should not be, i.e., if $G \cap E \neq E$ and $G \cap E \neq \emptyset$, then E/G are counted as errors.

Results The best performing model for Approach I scored a mean accuracy of 80.5% and a standard deviation of $\pm$4.6% for the character-based prediction task. For Approach II, the best performing model scored a mean accuracy of 86.4% and a standard deviation of $\pm$3.4% for the character-based prediction task.

Furthermore, we report the precision, recall and F1 score of signatures that have been extracted. IPLoM extracted 11 signatures, and none of them were entirely correct. LogCluster extracted 59 signatures, none of them were entirely correct. Approach I extracted 126 signatures. Again, none of them were correct. Approach II

Table 3.2: Result signature clustering (in %)

	Equal	Subset	Error
Ground Truth	100.0	0.0	0.0
IPLoM [89]	0.23	0.50	99.27
LogCluster [163]	**67.21**	31.15	1.64
Approach I	44.93	54.25	0.82
Approach II	52.53	47.47	0.0

extracted 110 signatures of which 19 were correct. The precision of Approach II was 17.27%, the recall 23.75% and the F1 score 20.0%.

In Table 3.2 we list the results of the clustering performance evaluation. IPLoM can cluster 0.73% of the test logs correctly, LogCluster manages to cluster 99.27% correctly, Approach I 99.18% and Approach II 100%. The signatures of LogCluster finds the most similar clusters. However, LogCluster is not capable of extracting any signatures and mistakenly groups 1.64% of the logs together that should not be in the same cluster.

3.4 — Discussion

When evaluating the character-based tagging performance, Approach II consistently performs better than Approach I. This can be explained by the considerably more difficult classification task. Instead of having to choose between only two tags in Approach II, Approach I has to choose between all possible characters and a special character for variable parts. This difficulty can also be observed when the uncorrected, tagged log lines are analyzed (see Section 3.2). In these lines, characters that should not be changed are frequently mis-classified and confused to other characters.

None of the evaluated approaches performed well on the log signature extraction task. We therefore analyzed the extracted signatures to get more insights. LogCluster and IPLoM are not able to extract correct signatures mainly for the following two reasons. First, both approaches are word-frequency based approaches. This means that signatures are mainly composed by large differences in word frequencies. The words in our dataset are fairly uniformly distributed, since we have 200 lines and 81 signatures. Secondly, both approaches use words as their base unit for extraction. This means that when confronted with a log "*[0.205212]*", both baseline approaches will create a signature "*[* φ", because "*0.205212]*" will be considered one word. According preprocessing such as adding white-spaces before brackets could improve the accuracy. Approach II was capable of extracting 19 correct signatures. The other signatures were mainly correct, except for a few characters that were mistakenly marked as mutable or vice versa. To illustrate the type of mistakes we list three examples of extracted signatures in Table 3.3. Signature S_1 has been extracted for log message X_1, signature S_2 and S_3 for X_2. Signature S_1 and S_2 are correct and the mutable parts have been replaced by a φ. Signature S_3 demonstrates an example of a frequent mistake that

Table 3.3: Example extracted signatures

X_1	"[kernel] : [0.603674] Linux agpgart interface v0.103"
S_1	"[kernel] : [φ] Linux agpgart interface vφ"
X_2	"[kernel] : [0.207326] pnp: PnP ACPI: found 2 devices"
S_2	"[kernel] : [φ] pnp: PnP ACPI: found φ devices"
S_3	"[kernel] : [φ] pnp: PnP ACPI: founφ φdevices"

the neural language model produces. It has mistakenly identified the last character of "found" as being mutable. These mistakes arise because we do not have a sufficient amount of trainings data for the neural language model to be able to generalize well. It therefore does not "know", that it is very unlikely that a word such as "found" last character mutable.

Both of our approaches perform well on the clustering task. Signatures that are extracted by Approach I cluster 99.18% of the test logs correctly and Signatures that are extracted by Approach II cluster 100% of the logs correctly. IPLoM only manages to cluster 0.73% of the logs correctly. IPLoM extracts 11 very generic clusters. These generic clusters match many log lines, which leads to large clusters with many errors. One reason for that is that IPLoM was designed for large logs with many messages and few signatures, which is not the case in our test dataset. In contrast to that, LogCluster can correctly cluster 98.36% of the logs, even though no exact signature has been extracted. A reason for this is that LogCluster extracts more "specialized" signatures that have less mutable parts, which will fit the logs of the test datasets but not correctly identify mutable parts.

Apart from that, the models that we learned in this chapter are trained on a comparably small log dataset. While they perform well on the presented tasks, this limits the conclusions one can draw about the ability of these models to generalize.

3.5 — Related Work

Over the past, log signature extraction has been studied with different goals such as anomaly and fault detection [41, 162], pattern detection [4, 90, 162], profile building [163] or compression [90, 151] of logs in mind. The main motivation for extracting signatures was the fast growing number of system logs in recent years [4, 41, 90, 151, 162, 163].

Different approaches can be used to cluster log lines and to extract signatures. Vaarandi et al. propose a word-density based approach for clustering loglines [162]. This approach had several shortcomings such as not being able to detect wildcards after the last word in a line pattern. Varaandi et al. addressed these shortcomings in a new version of their algorithm called LogCluster [163]. Makanju et al. propose an approach that relies on the word frequency as well as the position of words [89].

Aahron et al. propose a method to identify log signatures based on word frequencies, word order, and word entropy [4]. Instead of frequent words, Tang et al. base their signature extraction approach [151] on word bi-grams. These word bi-grams are constructed from log lines and use to for clustering. Finally, Fu et al. propose to extract log signatures by using word counts, word positions and a modified version of the edit distance [41].

He et al. provide a comparative study [58] on log clustering algorithms. In this study they compare four log clustering algorithms [41, 90, 151, 162] on five datasets. They find that all four can perform well with F1 scores ranging from 0.17 to 1.00. Moreover, they find that preprocessing the logs improves the accuracy of all four algorithms significantly.

Until recently, rule-based approaches dominated natural language processing (NLP) tasks such as part-of-speech (POS) tagging or named-entity recognition (NER). With increasingly available data and processing capabilities, neural network-based approaches have successfully replaced these rule-based approaches both in academic works and in practical applications.

Collobert et al. were one of the first to propose to use neural networks for various NLP tasks [175]. They used a convolutional neural network to achieve better than the then state-of-the-art results for POS tagging, chunking, named entity recognition and semantic relation extraction. After that, many approaches proposed to use a variety of neural architectures for NLP-related tasks, e.g., [37, 79, 165].

3.6 — Chapter Summary

In this chapter, we presented a signature extraction approach for forensic logs based on neural language models. We demonstrated that a character-based neural language model is capable of classifying mutable and non-mutable parts of log lines with an accuracy of 86.4%. We used this information to extract signatures from the logs and apply these signatures to a log clustering task. All logs were grouped correctly, either as equal (52.53%) or as subclusters (47.47%) of the desired clusters.

Our proposed method is only the first step towards automation because it relies on the availability of labeled data. While there is still some manual labor involved in labeling the data for this method, it bears the promise that it generalizes well to other data sets, and thereby reducing the amount of labor that is involved in designing a new, rule-based dataset for each scenario. In particular, the work presented in this chapter contributed to answering Research Question 1 in the following ways:

- We presented a novel method for signature extraction that uses a neural language model to classify the mutable and non-mutable parts. Specifically, the novelty is that we address the signature extraction problem a machine learning point of view without hand-crafting the algorithm.
- We empirically demonstrated that this method is capable of identifying the mutable and non-mutable parts of log lines with an accuracy of 86.4%.

- We showed that this method can be used for log clustering, which is a related problem to signature extraction. Furthermore, we empirically showed that it performs better than the state-of-the-art approaches.

Using supervised approaches is semi-optimal because they require labeled data to work. In this chapter, we mainly empirically verified our assumption that complex dependencies between elements of sequences can be modeled in a supervised way. However, as outlined in the introduction, typically supervision in the form of labeled data is not readily available. That is why in the next chapter, Chapter 4, we explore the use of an unsupervised method for the process of log clustering.

CHAPTER 4

Unsupervised Log Clustering

In Chapter 3 we introduced the problem of signature extraction and its related problem, log clustering. We also empirically verified our assumption that deep neural networks are capable of modeling log lines in a supervised way. As indicated in the introduction, one major problem is that in many cases supervised data is not readily available.

In this chapter, we want to address the challenge of lack of supervision. Hence, we are investigating how we can address the concrete problem of log clustering without labeled data (see Research Question 4). To achieve this, we propose a novel method that uses an RNN auto-encoder to create an embedding of the log lines. Loglines embedded in such a way can then be clustered. Clustering of unstructured text data remains challenging due to the high dimensionality and the discrete nature of the data. Our method learns a dense representation in continuous space of that data, which is beneficial for clustering since a distance between such representations can be computed efficiently.

The remaining chapter is structured as follows. First, we introduce the context and the problem in Section 4.1. The work in this chapter differs from the work in Chapter 3 directly work on the problem of clustering logs to their signatures, instead of first extracting the signatures and then use them to cluster the logs. In Section 4.2 we present our unsupervised method for clustering forensic logs. Then, in Section 4.3 we illustrate our experiment setup, which we use to show that our method can be used for log clustering. In Section 4.3.4, we list and discuss the results of the experiments, which show that our method beats the baseline and that the clusters found with our method have a strong Silhouette that indicates well-formed clusters.

4.1 — Introduction

An essential step of a forensic investigation is log analysis. Logs contain valuable information for reconstructing incidents that have happened on a computer system. In Chapter 3, we propose a method for signature extraction that can be used for forensic log analysis. We determined the signatures using a supervised deep learning model. In this Chapter, we are addressing a related problem, namely clustering log lines to their signatures, as typically in a forensic context labeled data that we have

used in Chapter 3 is not available. Instead of obtaining the signatures directly, here we obtain cluster IDs. Such cluster IDs represent a signature that is matched to a logline.

State-of-the-art approaches identify log signatures based on the position and frequency of tokens [4, 90, 163]. These approaches typically assume that frequent words define the fixed parts of the signature. This assumption holds if the ratio of log lines per signature in the analyzed log is high. Such a skewed ratio can be the case for many application logs where the tokens of fixed signature parts are repeated with a high frequency. However, in information forensics, logs commonly have many signatures but a few log lines per signature. In this case, the number of occurrence of tokens of variable parts may be higher than fixed tokens, which provide for a weak signal to discriminate which tokens are fixed and which ones are variable. Confusing fixed tokens with variable ones leads to signatures that match too few log lines, and mixing variable tokens with fixed ones will result in signatures that will match too few log lines.

To address the challenge of signature extraction from forensic logs, we propose to use a method that takes contextual information about the tokens into account. Our approach is inspired by recent advances in the NLP domain, where sequence-to-sequence models have been successfully used to capture natural language [38, 70]. The proposed method treats the signature extraction problem as a clustering problem, i.e., the problem of clustering log lines according to their signatures [151]. In this context, we treat a logline as a sequence of tokens that gives information about the state of a process that created this logline. The tokens of each log lines are partially natural language and partially structured data. Tokens may be words, numbers, variables or punctuation characters such as brackets, colons or dots. This method treats the loglines differently from the method that was proposed in Chapter 3, where viewed the log lines as sequences of characters.

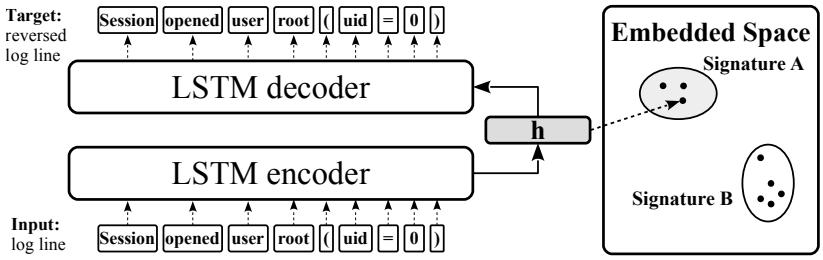

Figure 4.1: We first embed forensic log lines using an RNN auto-encoder. We then cluster the embedded log lines and assign them to a signature.

Typically, sequence-to-sequence models consist of two recurrent neural networks (RNNs), an encoder network and a decoder network. The encoder network learns to

represent an input sequence, and the decoder learns to construct the output sequence from this representation. We use such a model to learn an encoding function that encodes the logline, and a decoding function that learns to reconstruct the reversed input logline from this representation. Figure 4.1 depicts this idea. Based on the findings in the NLP domain, we assume that this embedding function takes contextual information into account, and embeds similar log lines close to each other in the embedded space. We then cluster the embedded log lines and use the clusters as signature assignment.

4.2 — Method

Our method LSTM-AE+C for signature extraction of forensic logs can be divided into two steps. First, we train a sequence-to-sequence auto-encoder network to learn an embedded space for log lines. Sutskever et al. have introduced Sequence-to-sequence neural networks for natural language translation [149], which have been widely applied on natural language problems since then. We use a similar model, however, instead of using it in a sequence-to-sequence manner, we use it as auto-encoder that reconstructs the input sequence. Secondly, we cluster the embedded log lines to extract the signatures.

We depict a schematic overview of our model in Figure 4.2. To learn an embedding, we train the LSTM auto-encoder to reconstruct each input logline. To do that, the encoder part of the auto-encoder needs to encode the logline into a fixed size vector that is fed into the decoder. The fixed size of the vector and the finite precision of its elements limits the capacity of the auto-encoder and provides for a regularization that restricts the auto-encoder from learning an identity function. We use that representation as embedding for the log lines. In the remaining section we will first detail the components of our model and their relationships to each other, then detail the learning objective and finally describe how we extract signatures.

4.2.1 – Model. The input to our model is log lines. We treat a log line as a sequence of tokens of length n, where a token can be a word, variable part or delimiter. The set of unique tokens is our input vocabulary, where each token in the vocabulary gets a unique id.

Since the number of such tokens in a log can be potentially very large, we learn a dense representation for the tokens of our log lines. To get these dense representations, we use a token embedding matrix $E^{(v \times u)}$, where v is the unique number of tokens that we have in our token vocabulary and u is the number of hidden units of the encoder network. The index of each row of E is also the position of v in the vocabulary. We denote a token that is embedded E as w.

Next, we want to learn the log line embedding. To do so, we learn an encoder function ENC using an LSTM [62], which is a variant of a recurrent neural network. We chose an LSTM for both the encoder and decoder, because it addresses the vanishing gradient problem. $h_e\,(t)$ is the hidden encoder state at time step t, and $y_e^{(t)}$ is the encoder output at time step t. $w_e^{(t)}$ is the embedded input word for time step t.

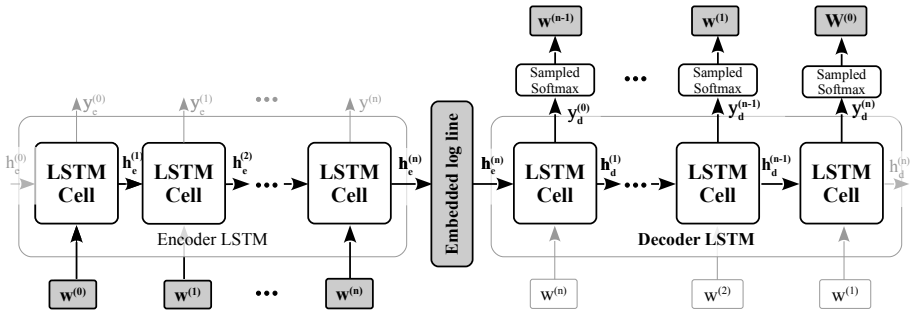

Figure 4.2: We use a sequence-to-sequence LSTM auto-encoder to learn embeddings for our log lines.

We use the encoding state and input word at each time step to calculate the next state and the next output. We use the final hidden state $h_e^{(n)}$ to embed our log lines. $h_e^{(n)}$ also serves as initial hidden state for our decoder network.

$$(y_e^{(t)}, h_e^{(t+1)} = \mathrm{ENC}(w_e^{(t)}, h_e^{(t)})$$

Our decoder function DEC is trained to learn to reconstruct the reverse sequence S' given the last hidden state $h_e^{(n)}$ of our encoding network. The structure of the network is identical to the encoder, except we feed the network the reverse sequence of embedded tokens as input.

$$(y_d^{(t)}, h_d^{(t+1)}) = \mathrm{DEC}(w_d^{(t)}, h_d^{(t)})$$

From the decoder outputs $y_d^{(t)}$ we predict the reverse sequence of tokens S'. Calculating a softmax function for a large token vocabulary is computationally very expensive because the softmax function is calculated in the denominator as the sum over all possible classes. Therefore, we predict the output tokens of our decoder sequence using sampled softmax [69]. Sampled softmax is a candidate training method that approximates the desired softmax function by solving a task that does not require predicting the correct token from all token. Instead, the task sampled softmax solves is to identify the correct token from a randomly drawn sample of tokens.

4.2.2 – Objective. To embed our log lines in an embedded space, the model needs to maximize the probability of predicting a reversed sequence of tokens S' given a sequence of tokens S. In other words, we want to train the encoding network to learn an embedding that contains enough information for the decoder to reconstruct it. However, as an effect of the regularization, we expect the model to use the structure

to use the structure of the log lines to create a more efficient representation, that in turn allows us to extract the signatures.

$$\theta^* = \arg\max_{\theta} \sum_{(S,S')} \log p(S'|S;\theta)$$

θ are the parameters of our model, S represents a logline, and S' represents a reversed logline.

S is a sequence of tokens of arbitrary length. To model the joint probability over $S'_0, \ldots, S'_{t-1}$ given S and θ, it is common to use the chain rule for probabilities.

$$\log p(S'|S,\theta) = \sum_{t=0}^{n} \log\, p(S'_t|S,\theta,S'_0,\ldots,S'_{t-1})$$

When training the network, S and S' are the inputs and the targets of one training example. We calculate the sum of equation 4.2.2 per batch using RMSProp [159]. We detail the hyperparameters of the training process in section 4.3.3.

4.2.3 – Extracting Signatures. After the training of our auto-encoder model is complete, we use the encoding network to generate the embedded vector. We expect that due to the regularization structurally similar log lines will be embedded close to each other, which enables us to use a clustering algorithm to group log lines which belong to the same signature.

Since forensic logs may be very large, we cluster the embedded log lines using the BIRCH algorithm [198]. BIRCH is an iterative algorithm that dynamically builds a height-balanced cluster feature tree. The algorithm has an almost linear runtime complexity on the number of training examples, and it does not require the whole dataset to be stored in memory. These two properties make the algorithm well suited for applications on large datasets.

4.3 — Experiments

We compare our method to LogCluster [163] and IPLoM [89] on their clustering performance. Many algorithms have been designed to be applied to a particular type of application log, where the number of signatures is known up front. However, in an information forensic context the forensic logs that are being analyzed stem from an unknown system, which means that the number of signatures is not known up front. Therefore, it is essential that IPLoM and LogCluster do not require a fixed number of clusters as a hyperparameter. Furthermore, in a study by He et al [58], IPLoM and SLCT were amongst the best-performing signature extraction algorithms. LogCluster is the improved version of SLTC that addresses multiple shortcomings of SLCT. We thus assume the LogCluster would have outperformed SLCT in He's evaluation. For IPLoM, we use the implementation provided by [58]. For LogCluster

we use the implementation provided by the author online[1]. We implemented our own method LSTM-AE+C in Tensorflow version 1.0.1. Our experiments are available on GitHub[2].

4.3.1 – Evaluation Metrics. To assess our approach, we treat the log signature extraction problem as log clustering problem because log clustering and log signature extraction are related problems [151]. The key difference between clustering and signature extraction is, that the goal of a clustering approach is the find the best clusters according to some metric, whereas the goal of signature extraction is to find the right set of signatures. This set of signatures does not have to be the best set of clusters.

We evaluate the quality of the retrieved clusters of all our evaluated approaches with two metrics: the V-Measure [123] and the adjusted mutual information [167]. The V-Measure is the harmonic mean between the homogeneity and the completeness of clusters. It is based on the conditional entropy of the clusters. The adjusted mutual information describes the mutual information of different cluster labels, adjusted for chance. It is normalized to the size of the clusters. Both approaches are independent of permutations on the true and predicted labels. The values of the V-Measure and the adjusted mutual information can range from 0.0 to 1.0. In both cases, 1.0 means perfect clusters and 0.0 means random label assignment.

Additionally, we assess the cluster quality for clusters retrieved with LSTM-AE+C using the Silhouette score. The Silhouette score measures the tightness and separation of clusters and only depends on the partition of the data [124]. It ranges between -1.0 and 1.0, where a negative score means many wrong cluster assignments and 1.0 means perfect clustering.

We validate the stability of our approaches using 10-fold cross-validation , randomly sub-sampled 10000 log lines [80]. In Section 4.3.4, we report the average scores for our metrics and their standard deviation.

4.3.2 – Datasets. We use three logs files to evaluate and compare our method: a forensic log that we extracted from a virtual machine hard drive and the system logs of two high-performance cluster computers, BlueGene/L(BGL) and Spirit [109]. An overview over the log statistics is presented in Table 4.1.

We created our forensic log by extracting it from an Ubuntu 16.04 system image disk using the open source log2timeline tool[3]. We manually created the signatures for this dataset by looking at the Ubuntu source code. The difference between a forensic log and a system log is that a forensic log contains information from multiple log files on the examined system, whereas a system log only contains the logs that were reported by the system daemon. The system log is part of the forensic log, but it also contains other logs, which typically leads to more complexity in such log files.

[1] https://ristov.github.io/logcluster/
[2] https://github.com/stefanthaler/2017-ecml-forensic-unsupervised
[3] https://github.com/log2timeline/

Table 4.1: The log file statistics are as follows:

Log Name	Lines	Signatures	Unique tokens
Forensic	11.023	852	4.114
BlueGene/L	474.796	355	114.495
Spirit2	716.577	691	59.972

BlueGene/L(BGL) was a high-performance cluster that was installed in the Lawrence Livermore National Labs. The publicly available system log was recorded during March 6th and April 1st in 2006. It consists of 4.747.963 log lines in total. In our experiments, we use a stratified sample which has 474.796 log lines. We chose to work on a stratified sample of the BlueGene/L and Spirit log due to hardware constraints. We manually extracted the signatures, and used Unix kernel sources as reference for the log messages.

Spirit was a high-performance cluster that was installed in the Sandia National Labs. The publicly available system log was recorded during January 1st and the 11th of July in 2006. It consists of 272.298.969 log lines in total. In our experiments, we use a stratified sample which has 716.577 lines. We also extracted the signatures of this log similarly to the BGL log. We chose to work on a stratified sample of the BlueGene/L and Spirit log due to hardware constraints.

The BlueGene/L and the Spirit logs are publicly available and can be downloaded from the Usenix webpage[4]. We publish our dataset on GitHub[5].

For all three log files, we removed fixed position data such as timestamps or dates at the beginning of each log message. In the case of our forensic log we completely removed these columns. In the case of the other two logs, we replaced the fixed elements with special token, such as TIME_STAMP. We added this preprocessing because it reduces the sequence complexity, but it does not reduce the quality of the extracted signatures.

4.3.3 – Hyper Parameters and Training Details. IPLoM supports the following parameters: File support threshold (FST), which controls the number of clusters found; partition support threshold (PST), which limits the backtracking of the algorithm; upper bound (UB) and lower bound(LB) which control when to split a cluster and cluster goodness threshold (CGT) [89]. We evaluate IPLoM by performing a grid search on the following parameter ranges: FST between 1 and 20 in 1 steps, PST of 0.05, UB between 0.5 and 0.9 in 0.1 steps, LB, between 0.1-0.5 in 0.1 steps and CGT between 0.3 and 0.6 in 0.1 steps. We chose the parameters according to the guidelines of the original paper.

[4]https://www.usenix.org/cfdr-data

[5]https://github.com/stefanthaler/2017-ecml-forensic-unsupervised

LogCluster supports two main parameters: support threshold (ST), which controls the minimum number of patterns and the word frequency (WF), which sets the frequency of words within a log line. We evaluate LogCluster by performing grid search using the following parameter ranges: ST between 1 and 3000 in and WF of 0.3, 0.6 and 0.9.

We generate each input token sequence by splitting a log line at each special character. Furthermore, we add a special token at the beginning and the end of the sequence that marks the beginning and the end of a sequence. Within a batch, sequences are zero-padded to the longest sequence in this batch, and zero inputs are ignored during training.

All embeddings and LSTM cells had 256 units. Both encoder and decoder network had a 1-layer LSTM. We trained all our LSTM auto-encoders for ten epochs using RMSProp [159]. We used a learning rate of 0.02 and decayed the learning rate by 0.95 after every epoch. Each training batch had 200 examples and the maximum length number of steps to unroll the LSTM auto-encoder was 200. We used 500 samples to calculate the sampled softmax loss. We used dropout on the decoder network outputs [144] to prevent overfitting and to regularize our network to learn independent representations. Finally, we clip the gradients of our LSTM encoder and LSTM decoder at 0.5 to avoid exploding gradients [116].

The hyper parameters and the architecture of our model were empirically determined. We tried LSTMs with attention mechanism [38], batch normalization, multiple layers of LSTMs, and more units. However, these measures had little effect on the quality of the clusters; therefore, we chose the simplest possible architecture. We used the same architecture and hyper parameters for all our experiments.

The second step in our method is to cluster the embedded log lines to find signatures. We cluster the embedded log lines using the BIRCH cluster algorithm [198]. We performed the clustering using grid search on distance thresholds between 1 and 50 in 0.5 steps, and a branching factor of either 15, 30 or 50.

4.3.4 – Results. We report the results of our experiments in Table 4.2. Each value reports the best performing hyper parameter settings. Each score is the average of 10-fold random sub-sampling followed by the standard deviation of this average. We do not report on the Silhouette score for LogCluster and IPLoM because both algorithms do not provide a means to calculate the distance between different log lines.

4.3.5 – Discussion of Results. As can be seen from Table 4.2, our approach significantly outperforms the two word-frequency based baseline approaches on the three datasets, both regarding V-Measure and Adjusted Mutual Information. The standard deviation is below 0.005 in all reported experiments, which indicates that clustering is consistently stable over the datasets.

For all three log files (see Table 4.3), we obtain a Silhouette score of greater than 0.70, which indicates that the clustering algorithm has found a strong structure in the embedded log lines. The weakest structure has been found in the Forensic log.

Table 4.2: Log clustering evaluation, best averages and standard deviation.

Log file	Approach	V-Measure	Adj. Mut. Inf.
Forensic	LogCluster [163]	0.904 ±0.000	0.581 ±0.000
	IPLoM [89]	0.825 ±0.001	0.609 ±0.001
	LSTM-AE+C	**0.935** ±0.002	**0.864** ±0.004
BlueGene/L	LogCluster [163]	0.592 ±0.004	0.225 ±0.005
	IPLoM [89]	0.828 ±0.003	0.760 ±0.005
	LSTM-AE+C	**0.948** ±0.005	**0.900** ±0.001
Spirit	LogCluster [163]	0.829 ±0.002	0.677 ±0.004
	IPLoM [89]	0.920 ±0.004	0.895 ±0.003
	LSTM-AE+C	**0.930** ±0.010	**0.902** ±0.008

Table 4.3: Clustering Silhouettes LSTM-AE+C, best averages and standard deviation.

Log file	Silhouette
Forensic	0.705 ±0.001
BlueGene/L	0.827 ±0.002
Spirit	0.815 ±0.004

We hypothesize that the high signature-to-log-line ratio in this log causes the lower Silhouette score.

Finding the optimal number of clusters for a clustering or signature extraction approach is a well-known problem. We do not address the topic of finding the optimal number of signatures in this chapter, but it is a fundamental research topic in many methods for finding the optimal number of clusters have been proposed, for example [146, 158].

4.4 — Related Work

Log signature extraction has been studied to achieve a variety of goals such as anomaly and fault detection in logs [41], pattern detection [4, 90, 163], profile building [163], or compression of logs [90, 151].

Most of the approaches use word-position or word-frequency based heuristics to extract signatures from logs. Tang et al. propose to use frequent word-bigrams to obtain signatures [151]. Fu et al. propose to use a weighted word-edit distance function to extract signatures [41]. Makanju et al. use the logline length as well as word frequencies to extract signatures [89]. Vaarandi et al. use word frequencies and word correlation scores to determine the fixed parts of log lines and thereby the

signatures [163]. Xu et al. propose a method that is not based on statistical features of the log lines. Instead, they propose to create to extract the signatures from the source code [184]. These approaches rely on manually defined features or hand-crafted algorithm. In contrast, our proposed method uses recurrent neural auto-encoders to learn representations that are suitable for clustering logs according to their signatures.

Recently, RNN sequence-to-sequence models have been successfully applied for neural language modeling and statistical machine translation tasks [24,38,149]. Apart from that, Johnson et al. demonstrated on a large scale that sequence-to-sequence models can be used to allow translation between languages even if explicit training data from source to target language is not available [70]. Auto-encoders have been successfully applied to clustering tasks, such as clustering text and images [179]. In contrast to [179], our proposed method uses a model architecture that is directly optimized for operating on sequences instead of using a generic neural network architecture.

4.5 — Chapter Summary

In this chapter, we presented the LSTM-AE+C an unsupervised method for clustering forensic logs according to their log signatures. Knowing that log lines belong to the same signature enables a forensic investigator to run more sophisticated analysis on a forensic log, for example, to reconstruct security incidents. Our method uses two components: an LSTM encoder and a hierarchical clustering algorithm. The LSTM encoder is trained as part of an auto-encoder on a log in an unsupervised fashion, and then the clustering algorithm assigns embedded log lines to their signature.

Experiments on three different datasets showed that this method outperformed two state-of-the-art algorithms on clustering log lines based on their signatures both in V-Measure and adjusted mutual information. Moreover, we found that the Silhouette score of all found clusters by our method is higher than 0.70, which indicates strongly structured clusters.

In detail, the main contributions toward an answer for Research Question 2 are:

- We proposed a novel, unsupervised method, LSTM-AE+C, that uses an RNN auto-encoder to create a logline embedding space. We then clustered the log lines in the embedding space to determine the signature assignment. The novelty is to propose an unsupervised method that does not rely on a hand-crafted algorithm.
- We demonstrated on our own and two public datasets that LSTM-AE+C outperforms two state-of-the-art approaches to the problem of clustering along their signatures.

Unsupervised ML methods directly discover structures in the data and therefore do not require labeled data. The absence of a need for labels saves much manual effort and is a considerable step towards automation. However, in some instances, a large amount of unlabeled data and only a few labeled examples exist. In such a scenario, it is desirable to combine the labeled data with the unlabeled data. LSTM-AE+C does

not readily include mechanisms for combining labeled with unlabeled data. We will explore a way to do so in the next Chapter 5.

CHAPTER 5

Using Approximate Information

In Chapter 3 we introduced the problem of signature extraction and proposed a method using supervised deep learning to verify the assumption that deep learning models can capture forensic loglines. In Chapter 4, we proposed an unsupervised method for the related problem of log clustering as a contribution to the challenge of lack of supervision.

Supervised learning offers higher precision, but requires labeled data. Unsupervised learning is more difficult but does not require labels. Here, we will consider the problem of log clustering in a scenario where we have approximate domain knowledge in the form of heuristics. Machine learning algorithms are generally designed to learn their parameters from labels and have limited ways of using such domain knowledge in the learning process.

In this chapter, we address the challenge of the usage of domain knowledge by proposing a novel method for clustering forensic logs. The proposed method allows learning a model for clustering forensic logs using approximate domain knowledge, which is available in the form of a heuristic. To achieve this, we address the forensic log-clustering problem as a metric-learning problem. This perspective allows us to treat the assignment of labels to a logline as a ranking problem. Ranking problems are easier to solve than exact assignments, which allow us to use a heuristic for learning as well as combine such a heuristic with labeled information.

This chapter starts with a brief introduction on how metric learning can be achieved using deep neural networks (see Section 5.1). In Section 5.2.4 we propose a method for using heuristic knowledge (see Section 5.2.3) that is based on a custom loss function and sketch how this method can be readily combined with labeled examples. In Section 5.3 and 5.4 we provide empirical evidence that the proposed method improves on the heuristic, and that adding few labeled examples increases the performance even further. In Section 5.5 we provide a brief discussion of related work.

5.1 — Introduction

Metric learning methods enable learning of a distance metric that allows projecting data in an embedded space. These methods offer significant flexibility since the data is

not mapped to a particular value such as in supervised classification or regression, but are projected to an embedding space based on their relationship to other data points in the dataset. Different kinds of semantical labels can then be assigned, interpretation of the model decision can be grounded in the training data and techniques such as one shot [76] and zero shot [142] learning can be performed. Furthermore, in many scenarios relative differences between data points can be obtained "cheaper" than direct labels.

In this chapter, we examine metric learning on sequence data. We are presented with sequences of symbols (tokens) that are assigned a class or a label. However, our label space is complex and not well defined. There is a large number of classes, and many of those appear infrequently. Certain classes may be considered "out-of-vocabulary", since they may not appear in the training data, but can appear during inference. In this setting, utilizing all available data requires to collect supervision on the relative distance [103] or ranking [173] between the data points. To tackle this, we propose to use a proxy distance metric that is weakly correlated to the target metric and can be used in the ranking task. Learning in such a way can be combined with a supervision signal that allows for high-quality embedding at a fraction of the needed supervision.

5.2 — Deep Metric Learning Using Approximate Information

Our goal is to learn an embedding function that maps symbolic token sequences from feature space to a metric space such that semantically similar sequences are mapped close to each other, and non-similar ones are farther apart. The general idea behind this is called metric learning [180]. Recently, deep learning methods based on Triplet networks [173] have been proposed to learn such embedding functions. Triplet networks learn a metric by optimizing a ranking problem on input triplets. A triplet consists of an anchor sequence, a positive sequence, and a negative example. These three examples related to each other by a similarity relationship, i.e., the positive example should be more similar to the anchor example than the negative example. The triplet network is trained to learn a function that embeds examples into a metric space, in which positive sequences are closer to the anchor example than negatives ones.

To the best of our knowledge, triplet networks so far only have been trained using information from labels. That is, a training example is positive concerning an anchor example if both have the same label, and negative otherwise. Labels are often not available, or labor-intensive to obtain.
The main idea presented in this chapter is to use a proxy distance metric in combination with a few labeled examples to determine the similarity relationship between triplets of examples. This proxy distance metric gives some indication about the ranking but is not very precise otherwise. If needed, the learned metric space can be improved by adding a few labeled examples. Thus, in other words, we conduct a form of weakly supervised learning [202] to learn a distance metric using triplet networks.

This method offers two advantages over just using a distance metric on the pairwise training examples. First, it scales with the number of training examples, and second, it allows to learn a higher quality, domain-specific metric. It also offers a generic way to combine existing metrics on the input data with labeled examples.

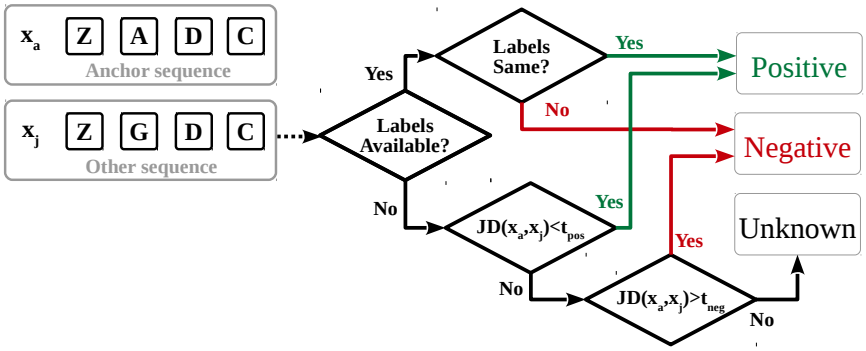

Figure 5.1: Here we show the relationship defining processes of a sequence x_j to the anchor sequence x_a. Our main idea is to learn a distance metric in a weakly supervised way by defining the similarity relationships of input data using a proxy distance metric and a few labeled examples. In combination with triplet networks, these similarity relationships allow us to learn a domain-specific metric for our input sequences.

5.2.1 – Triplet LSTM. Here we describe the model that we use to learn a distance metric for sequences. We base our model architecture on the Triplet network that was introduced by Wang et al. [173]. Such a model consists of a deep neural network for embedding, followed by a L_2 normalization layer. Each input of the triplet is embedded using the same network, and the normalized output is used for calculating the triplet loss. Instead of a deep convolutional neural network, we use an LSTM [62] for embedding the sequences. LSTMs are well-suited for modeling sequences and have been applied to many sequential learning problems, e.g., [95]. We refer to our embedding network including the normalization layer as f, x is a sequence and z is an embedded sequence $z = f(x)$. Figure 5.2 depicts our triplet network architecture schematically.

5.2.2 – Objective. Here we describe our objective for learning the distance metric for sequences. We use the triplet loss that was introduced by Wang et al. [173]. This objective penalizes triplets where the distance between the embedded anchor and the embedded positive example is larger than the distance between the embedded anchor and the negative example.

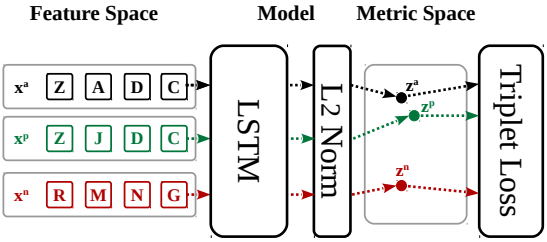

Figure 5.2: We use a triplet-LSTM followed by a L_2 normalization layer to learn a metric embedding space for sequences.

5

Given a triplet of an embedded anchor example z_i^a, an embedded positive example z_i^p and an embedded negative example z_i^n, this objective minimizes the difference in distances between the anchor and the positive and the anchor and the negative example and a given margin α. α is a hyperparameter.

Let T be the set of all possible triplets in a mini-batch, and let the term $[l]_+ = max(0, l)$ denote the standard hinge loss.

Then our learning objective is to minimize the loss L

$$L = \sum_i^N \left[\|z_i^a - z_i^p\|_2^2 - \|z_i^a - z_i^n\|_2^2 + \alpha \right]_+ \tag{5.1}$$

for the set of all triplets that violate the following constraint:

$$\|z_i^a, z_i^p\|_2^2 + \alpha < \|z_i^a, z_i^n\|_2^2 \; \forall (x_i^a, x_i^p, x_i^n) \in T \tag{5.2}$$

The triplet loss L will be small if the distance between the positive example and the anchor is small and the distance between the negative and the anchor is high and large otherwise. The parameter α ensures that a margin between examples of the same class is allowed.

5.2.3 – Jaccard Distance. We use the Jaccard distance as a proxy metric to determine the similarity relationship between two input sequences. The Jaccard distance (JD) is a distance measure between two sets x_1 and x_2. It is defined as:

$$JD(x_1, x_2) = 1 - \frac{|x_1 \cap x_2|}{|x_1 \cup x_2|} \tag{5.3}$$

with $\cap$ being the intersection of the two sets and $\cup$ being the union of the two sets. To calculate the Jaccard distance between two token sequences, we treat the sequences as sets of tokens. The Jaccard distance tells us about the diversity of two sequences,

but it ignores informative properties of the sequences such as the order of the tokens. We hypothesize that using the Jaccard distance provides sufficient information to rank triplets based on their dissimilarity.

5.2.4 – Method. In this work we propose a method, that relies on a proxy distance metric and a few labeled examples. This method enables to learn a domain-specific distance metric for sequences using a triplet network using only a fraction of the required labels.

To learn distance metrics with a triplet network, we need to define the relationship of input triplet examples. That is, given an anchor example, we need to know whether an example is positive, i.e., belongs to the same class or negative, i.e., belongs to a different class.

The relationship between input examples is defined between an anchor example x_a and another example x_j. We define this relationship using a two-step process. This two-step process is depicted in Figure 5.1.

First, if we have information about the labels of both examples, we use this information to determine the similarity relationship. This step is similar to the definition of the relationship in previous works on triplet networks.

If we do not have label information available, we use the Jaccard distance JD as proxy distance metric to determine the relationship between x_a and x_j. If the Jaccard distance is below a threshold t_{pos}, the relationship is *positive* and if it is above another threshold t_{neg}, the relationship is *negative*. If the Jaccard distance is above the positive threshold t_{pos} but below the negative threshold t_{neg}, we define the relationship as *unknown*. If the relationship between a pair of sequences is unknown, we ignore it in the triplet selection process (see Section 5.3.3) of the training phase.

The positive and negative thresholds are hyperparameters and depend on the data domain. The margin between the positive and the negative threshold is meant to increase the accuracy of the approach. That is, only if it is likely that two sequences are similar they should be labeled *positive*, and only if it is improbable that they are similar to *negative*.

Formally, we define the similarity relationship between an anchor sequence x_a and another sequence x_j as follows. Let x^a, x^j be two sequences of tokens, l^a, l^j their respective label, $JD(x^a, x^j)$ the Jaccard distance between the two sequences, and t_{pos}, t_{neg} the thresholds for being a positive or a negative example pair. If a sequence is not labeled, it's label is $\emptyset$.

We then define the similarity relationship R between two sequences as:

$$R(x^a, x^j) = \begin{cases} \text{pos}, & \text{if } l^a \neq \emptyset \text{ and } l^j \neq \emptyset \text{ and } l^a = l^j \\ \text{pos}, & \text{if } l^a = \emptyset \text{ or } l^j = \emptyset \text{ and } JD(x^a, x^j) < t_{pos} \\ \text{neg}, & \text{if } l^a \neq \emptyset \text{ and } l^j \neq \emptyset \text{ and } l^a \neq l^j \\ \text{neg}, & \text{if } l^a = \emptyset \text{ or } l^j = \emptyset \text{ and } JD(x^a, x^j) \geqslant t_{neg} \\ \text{unk}, & \text{otherwise} \end{cases} \tag{5.4}$$

with pos being *positive*, neg being *negative* and unk *unknown*.

5.3 — Experimental Evaluation

Here we present our experimental evaluation of our method. We want to show that i) approximate information in combination with a distance metric is possible, and ii) that adding labeled examples to this process increases the performance.

We evaluate our approach to the task of classifying log lines from log datasets. Log datasets typically consist of multiple log lines, and each log line can be treated as a sequence of tokens. The task is to classify each log according to the print statement that created a specific logline. Loglines that originated from the same print statement should obtain the same label. Since print statements often have variable parts, the resulting log lines may often differ, and the semi-structured nature of such logs makes this a challenging task.

5.3.1 – Baseline. We use our proxy distance metric as the baseline. For evaluation, we calculate the pairwise Jaccard distance from each log line to each other logline. Given this pairwise distances, we can calculate the accuracy, the true acceptance, and the false acceptance rate (see Section 5.3.9). We refer to our baseline method as *Base-JA*.

5.3.2 – Model. Here we detail the model that we use for performing our deep-learning based experiments. In all our experiments we use the same model. This model takes sequences of token ids as input. These sequence of token ids are mapped to sequences of token vectors using a dense token embedding matrix. The model outputs a point in the metric space in case of the triplet network experiments (see Section 5.3.4) and class probabilities in case of classification experiments (see Section 5.3.5).

We use an LSTM [62,193] to encode the token vector sequences. We use dropout [144] to prevent overfitting, and gradient clipping [116] to prevent exploding gradients. There are many more advanced deep learning architectures for modeling sequences. We chose a very basic architecture because we were mainly interested in studying the effects of combining selecting the triplet with labels. Also, LSTMs have shown to be a reliable choice for modeling sequences [95].

We learn our model parameters using RMSProp [159]. RMSProp is a momentum-based variant of stochastic gradient descent. We train our model in mini-batches. We implemented our model in Tensorflow 1.4.0 [1] and Python 3.5.3.

5.3.3 – Triplet Selection. Selecting the right triplets as input for the triplet network is crucial for the learning to converge [129]. We use an online strategy to select the triples for training our network. We use all valid combinations of anchor-positive and anchor-negative examples per mini-batch to train our network. anchor-unknown example pairs are ignored. Valid in this context means triples that violate the condition in Equation 5.2. We additionally add randomly sampled positive examples of each labeled example in a batch to increase the number of valid triplets per mini-batch.

These positive examples are uniformly sampled from the distribution of examples of this the same class.

5.3.4 – Experiment - Metric Learning Using Approximate Information. In this experiment, we test our method for learning a metric space using triplet networks and a combination of a proxy distance metric in combination with labeled examples. To embed the sequences in the metric space, we learn an embedding function using the model that was described in Section 5.3.2. The setup for this experiment is schematically depicted in Figure 5.2. We refer to this method as *LSTM-Triplet*.

5.3.5 – Experiment - Classification with Augmentation. Triplet networks are complex architectures. To justify the complexity, we compare the results of the metric learning experiments with an experiment that uses the same model (see Section 5.3.2). However, instead of learning a metric space, we learn to classify the sequences based on their label. To do so, we extend our model with an addition softmax layer after the encoding. This layer has neurons according to the number of classes available.

For a fairer comparison, we augment our training data by labeling additional classes using the Jaccard distance. We obtain such further labeled examples for classification by labeling any other unlabeled sequence that has a Jaccard distance to the labeled example that is below a certain threshold with the same class. If there are multiple candidates for labeling, we label the example with the lowest Jaccard distance. We refer to this method as *LSTM-Class*.

5.3.6 – Datasets. To evaluate our method, we use three datasets, a UNIX forensic log file [157], and two system log files of computing clusters BlueGene/L and Spirit2 [109].

The UNIX forensic log file contained 11,023 log lines and was extracted from an Ubuntu system. A forensic log is a log that aggregates many different log sources from a computer system into one large log file. Such an aggregated log file can be used for further forensic analysis in cybercrime investigations. The UNIX forensic log lines originate from 852 print statements, i.e., the logs have 852 different classes. The labels of each log line have been assigned manually by using the source code of the as ground truth. We only use a fraction of the computing cluster system logs. We use 474,700 log lines of the BlueGene/L dataset and 716,577 of the Spirit2 dataset. The BlueGene/L log lines originate from 355 different print statements, and the Spirit2 log lines originate from 691 separate print statements.

5.3.7 – Data Pre-Processing. We pre-process each log line before we use it as input to our deep learning models. We transform the log lines to lower-case and split them into tokens based on whitespace characters. We treat special characters such as brackets also as tokens. All our logs are semi-structured, i.e., they have fixed columns at the beginning and a free text part afterward. In case of the UNIX forensic log, we have removed the fixed columns. In case of the BlueGene/L we have replaced the fixed columns with fixed tokens and only use the free text part for learning. We replace the information of the fixed columns because it could be easily extracted without a

Listing 1: Four sample log lines from the UNIX forensic log. Although they are different, they should be labeled the same since they originate from the same print statement.

```
[systemd  pid: 1045] : Startup finished in 33ms.
[systemd  pid: 867] : Startup finished in 53ms.
[systemd  pid: 909] : Startup finished in 34ms.
[systemd  pid: 1290] : Startup finished in 31ms.
```


machine learning approach. We add a particular start and end token to each sequence. After the pre-processing is done, we have a vocabulary of 4114 different tokens for the UNIX dataset, 101,872 tokens for the BlueGene/L dataset and 59,340 unique tokens for the Spirit2 dataset. Finally, pad the sequences by appending zeros so that each sequence has the same length.

5.3.8 – Hold Out Sets. We use a fraction of 0.2 of the UNIX forensic log and a fraction of 0.1 of the other two datasets as the hold-out set for testing our ideas. The UNIX forensic test set has 602 classes. Each class has two members in the median and the standard deviation of 5.92 member sequences. The BlueGene/L dataset has 245 classes, ten members in the median and a standard deviation of 1162. The Spirit2 dataset has 449 classes with three members in the median and a standard deviation of 1510.

5.3.9 – Evaluation. We compare the validation rate *VAL(d)* and the false acceptance rate *FAR(d)* for the baseline experiment and the metric learning experiment. We use the definition of the validation and false acceptance rate that was introduced by Schroff et al. [129], but we adopt it for sequences. The validation rate measures the correctly classified as same sequence pairs at a given distance threshold, whereas the false acceptance rate measures the incorrectly classified as the same sequence of pairs at a given distance threshold. We define d as distance threshold, and $D(x_i, x_j)$ as distance between a pair of sequences x_i, x_j. In case of the baseline experiment, $D(x_i, x_j)$ denotes the Jaccard distance between two token sequences, and in case of the metric learning experiment $D(x_i, x_j)$ denotes the squared L_2 distance. We denote all pairs of sequences that are in the same class as P_{same}. We denote all pairs of sequences of different classes as P_{diff}.

True accepts are the sequences that are correctly identified as belonging to the same class at a given distance threshold. We define the set of all *true accepts* $TA(d)$ as:

$$TA(d) = \{(i,j) \in P_{same}, with D(x_i, x_j) \leqslant d\} \tag{5.5}$$

False accepts are the sequences that are incorrectly identified as belonging to the same class at a given distance threshold. We define the set of all *false accepts* $FA(d)$ as:

$$FA(d) = \{(i,j) \in P_{diff}, withD(x_i, x_j) \leqslant d\} \tag{5.6}$$

The validation rate $VAL(d)$ and the false accept rate $FAR(d)$ for a given distance threshold d are then defined as:

$$VAL(d) = \frac{|TA(d)|}{|P_{same}|}, \; FAR(d) = \frac{|FA(d)|}{|P_{diff}|} \tag{5.7}$$

Furthermore, we compare the accuracy of the classification experiment. In the Base-JA and LSTM-metric experiments, we report the accuracy at the distance threshold when the validation rate is 1.0, i.e. when all pairs that should be classified as the same are classified as same.

5.3.10 – Hyper Parameters and Training Details. We learned our model parameters with a learning rate of 0.01. We decayed the learning rate by 0.95 after each epoch. The choice of our learning rate is empirically determined. Learning rates larger than 0.01 led to a collapse of the learned metric space, where all sequences were mapped to a single point. This collapse is mainly caused by the L2 term in the loss function, which will get too large. Learning rates that were smaller than 0.01 led to a slower convergence.

The token embedding matrix had a dimension of 32 and is initialized with a normal distribution and a standard deviation of 0.5. The number of neurons in the LSTM was 32. The embedding dimension and the LSTM capacity have been empirically determined. Smaller embedding dimensions and capacity caused underfitting, and larger dimension caused a collapse of the embedding space to a single point.

We clipped gradients at 0.5 to prevent exploding gradients, which are a common problem in deep networks such as LSTMs. We dropped out inputs at a rate of 0.1 to prevent overfitting. Both parameters were empirically determined.

We trained with a mini-batch size of 100, which has also been empirically determined. Larger mini-batch sizes worked better but are also slower because the pairwise operations that need to be computed. Smaller mini-batch sizes led to fewer triplets that violated the triplet constrained.

We used RMSProp [159] with ϵ of 1^{-10} and momentum of 0.0, which were the default settings of the implementation that we used. The default settings worked well for our experiments. We trained each model for 30 epochs and randomly shuffled the training data after each epoch. We shuffled the batch such that different pairs of sequences were being used for the pairwise operations. To determine the number of epochs, we looked at the number of triplets that violated the triplet constraint. After 30 epochs, the number usually converges to a very low number. A low number of violating triplets meant that our model was only learning very little. However, since we randomly shuffle our training sequences, the number of triplets that violate the triplet constraint is not a reliable indicator of when to abort the training.

For our LSTM-Triplet experiments, we added two extra positive examples per labeled sequence to each batch. This selection means that for each labeled sequence we randomly sample two sequences of the same class and add these sequences to the mini-batch. Adding more labeled examples to the batch increases the available information, thus speeds up the conversion of the learning process.

Furthermore, we used a Jaccard distance threshold t_{pos} of smaller than 0.3 to determine positive examples, and a Jaccard distance of t_{neg} greater than 0.7 to determine negative examples. We introduced a margin between the positive and the negative examples so to account for the uncertainty that is introduced when using the Jaccard distance as a labeling function. This margin was determined by selecting a margin for two hypothetical sequences of length 20 such they had significant overlap in the positive case and were significantly different in the negative case. We use an α of 0.8 to calculate the hinge loss for our triplet network (see Equation 5.1). We chose a higher margin than Schroff et al. [129] to account for the uncertainty that is introduced by using weak supervision.

We conducted the LSTM-Triplet and the LSTM-Class experiments with 1000, 2500 and 5000 labels. Additionally, we perform the triplet network experiment without additional labels. We chose an increasing amount of labels to see how additional labels increase the learning performance.

5.4 — Results and Discussion

Here we report and discuss the results of our experiments. First, we compare the learned metric spaces. We then report the accuracy of the classification experiments.

In Figure 5.3-5.5 depict the VAL-rate against the FAR-rate for increasing distance thresholds. These plots show the results for the baseline Jaccard distance (see Section 5.3.1) and compare it to the triplet network learned with Jaccard distance and 0, 1000, 2500 and 5000 labeled examples (see Section 5.3.4). The results are interpolated, and the x-axis of the plots is in log-scale. The nature of the datasets can explain the bumps in the baseline. First, the number of log lines per class varies considerably, and there are few more dominant classes. Second, log lines frequently only differ in very few tokens, i.e., a different variable. The Jaccard distance for sequences is the same for sequences that differ the same number of tokens, regardless of which class they are.

The performance of the triplet network on the UNIX forensic dataset without additional labels is comparable to the performance of our baseline, the Jaccard distance. In the case of the Spirit2 dataset, the performance is better, and in the case of the BlueGene/L dataset the performance is slightly worse.

In our triplet network experiments, we used a margin in the heuristics for determining whether another sequence is positive or a negative example related to the anchor example. Due to this margin, one would assume that a distance metric that is learned by such a triplet network performs better than applying the heuristic directly (i.e., our baseline). However, we can only observe this in the case of the Spirit2 dataset.

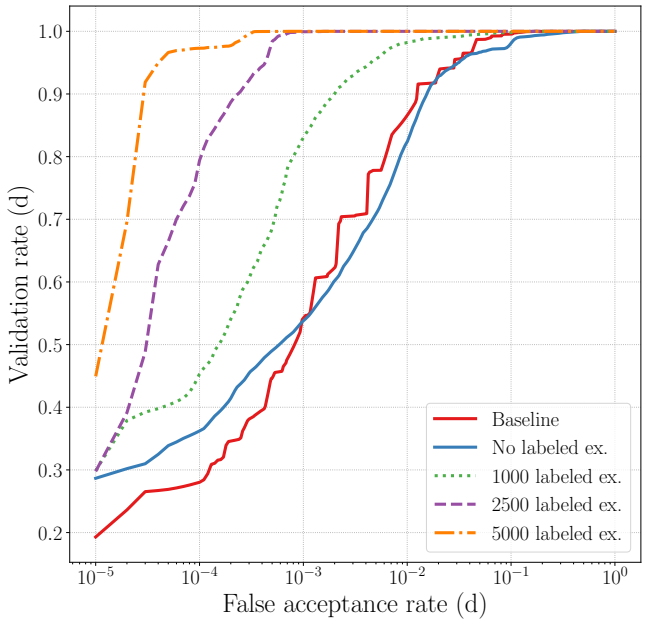

Figure 5.3: VAL/FAR rate UNIX forensic dataset. We show the baseline experiments and the triplet network with different number of additional labeled examples. The bumps in the baseline can be explained by the nature of the datasets.

We argue that this is caused by the way triplet networks learn. We observe that there are consistently many more anchor negative-pairs than anchor-positive pairs during training since more sequences are different from each other than similar. This imbalance leads to a learning behavior that pushes sequences that are different away from each other, but not necessarily close to each other. This process results in well-separated, but split sequence clusters in the metric space, which is reflected in the worse performance.

We hypothesize that it is possible to mitigate this imbalance in negative to positive example pairs by carefully selecting the examples for each batch. However, in a scenario with unknown classes and no labeled data, such a strategy may be difficult to devise.

Learning with triplet networks is more efficient regarding distance comparison operations needed. The computational complexity of pairwise comparisons is $O(n^2)$. In contrast, the triplet network has a computational complexity of $O(n * m^2)$ for training, where m is the size of the mini batch.

Furthermore, in Figures 5.3, 5.4 and 5.5 we can observe that adding labeled examples consistently increases the performance of the metric learning approach.

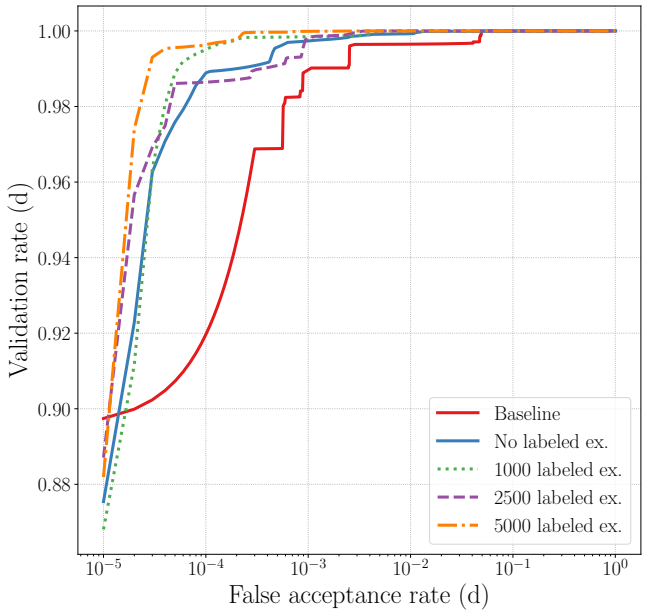

Figure 5.4: VAL/FAR rate Spirit2 dataset.

The more labels are added, the better separated the clusters are.

Apart from that, we have also explored the idea of using the triplet network only with fewer labels and discard the information of the Jaccard distance. In this case, the triplet network overfits and performs worse than the baseline and worse than the triplet trained with only the Jaccard distance.

In Table 5.1 we report the accuracy for our classification experiment (see Section 5.3.5). We calculated the accuracy of the metric-based experiments at the distance threshold d where the VAL-rate was 1.0. We report the distance threshold as well as the accuracy.

In Table 5.1 we can see that already the baseline Jaccard distance classifies the log lines reasonable well. The LSTM trained with a few labels and augmented with Jaccard distance is consistently better, and the triplet network performs better than the LSTM.

We argue that the triplet network performs better in the classification task for two reasons. First, the metric space can deal better with unknown classes. Second, the triplet network can use the available information better. The classification LSTM can only use the information that a sequence belongs to one class and not to all the others. The metric learning LSTM can also utilize the information that is gained when pairwise sequences are from different classes.

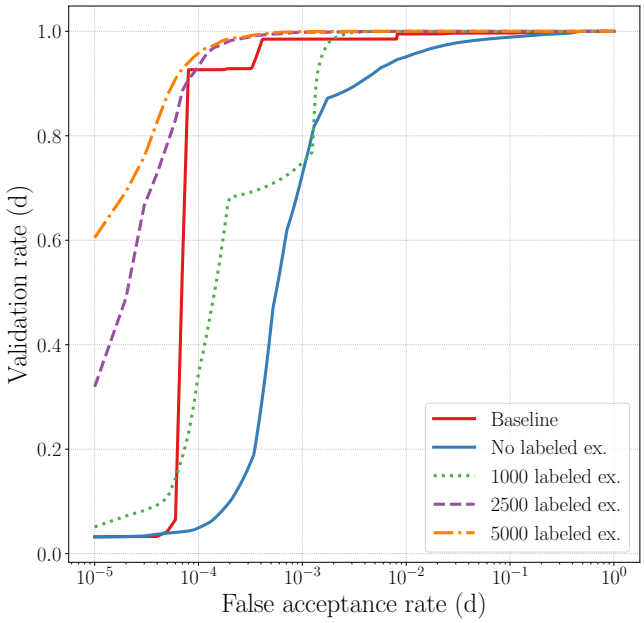

Figure 5.5: VAL/FAR rate BlueGene/L dataset.

5.5 — Related Work

Distance metrics can be learned using Siamese network, and have been originally introduced to distinguish handwritten signatures [18]. They use two identical models and a contrastive energy loss function to learn a distance metric. Siamese networks have been used for deep metric learning in multiple domains, such as face verification [25], image similarity [76] or for learning image descriptors [20].

More recently, Siamese networks have been used to learn distance metrics for sequences and to learn distance metrics for text. Mueller and Thyagarajan propose to use a recurrent Siamese architecture to learn a distance metric for defining similarity between sentences [103]. Neculoiu et al. describe a similar architecture to learn a distance metric between character sequences [105].

Triplet networks have been proposed as an improvement to Siamese architectures [173]. Instead of two identical networks and a contrastive energy function, they learn the distance metric by minimizing a triplet-based ranking loss function. Triplet networks have been successfully applied to a variety of image metric learning tasks [63, 129]. Furthermore, magnet networks have been [122] proposed as an improvement over triplet networks. Magnet networks learn a representation space that allows identifying intra-class variation and inter-class similarity autonomously.

Table 5.1: We compare the accuracy of our classification experiments. We calculated the accuracy of the metric-based experiments at the distance threshold d where the VAL-rate was 1.0.

Dataset	# labels	Base-JA	LSTM-Triplet	LSTM-Class
UNIX Forensic	0	**0.841** (0.69)	0.550 (1.40)	N/A
	1000	N/A	**0.866** (1.26)	0.728 (N/A)
	2500	N/A	**0.998** (0.88)	0.913 (N/A)
	5000	N/A	**0.999** (0.68)	0.960 (N/A)
Blue-Gene/L	0	**0.633** (0.52)	0.612 (0.48)	N/A
	1000	N/A	**0.996** (0.54)	0.971 (N/A)
	2500	N/A	**0.997** (0.52)	0.986 (N/A)
	5000	N/A	**0.999** (0.49)	0.991 (N/A)
Spirit2	0	0.961 (0.51)	**0.990** (0.70)	N/A
	1000	N/A	**0.997** (0.45)	0.973 (N/A)
	2500	N/A	**0.998** (0.45)	0.981 (N/A)
	5000	N/A	**0.999** (0.71)	0.982 (N/A)

5.6 — Chapter Summary

In this chapter, we proposed a novel method for efficiently learning metric spaces for sequential data to address the challenge of the usage of available domain knowledge. The work in this chapter presented two novel ideas. First, the proposed method allows the use of approximate domain knowledge for weak supervision in the learning process. Secondly, the proposed method allows for refining this domain knowledge with existing labeled data, i.e., strong supervision. Both ideas enable the more efficient use of supervision for learning of such metric spaces. Furthermore, we showed that this metric spaces could be used to cluster forensic logs.

More specifically, the contributions to an answer to Research Question 3 and the broader challenge of integrating domain knowledge to the machine learning process are:

- We presented a method for efficient metric learning with sequence data by adapting the triplet network for sequence data using recurrent neural networks (RNN). This method allows for using existing domain knowledge which was available in the form of a heuristic (Jaccard distance).
- We showed, that this method also allows combining such domain knowledge with labeled data, which improves the efficiency of the labeled data for learning such a metric space.s
- We provided empirical evidence on three log-line datasets that this method can be used for clustering forensic logs.

This chapter was the last chapter about our forensic use case. On the example of forensic log clustering, we proposed methods that contribute to the challenge of lack of supervision and the integration of domain knowledge. In the next chapter, we will address the challenge of lacking contextual information.

CHAPTER 6

Automated Decoy Generation

The third challenge that we want to address in this thesis is the lack of contextual information. In this chapter, we address this challenge from the perspective of a concrete example from intrusion detection, more specifically, data leakage detection. One frequent target of data thieves is intellectual property in the form of software, which most commonly is organized in project directories.

Merely monitoring the actions on this software will not reveal enough information to protect against data theft of such software projects. For example, the action of copying a project to an external hard drive can be both malicious and benign. Malicious, if it is copied with the intent to steal it, and benign if it is copied to make a backup.

One way to obtain such contextual information about the intent is by using project decoys. Decoy projects are fake projects that look like real projects but are intrinsically worthless. Hence, any interaction with them is suspicious. Manually creating such decoy projects is a tedious task. That is why in this chapter we explore the task of how to automate the creation of believable decoy project directories. The general idea is to learn the structure of real projects and use this learned structure to sample decoy projects from that structure.

The rest of this chapter is structured as follows. First, in Section 6.1, we introduce the data-theft scenario, in particular software theft. We also outline how decoys can be used to help to detect data-theft. In Section 6.2 we introduce desired decoy properties and motivate why we focus on believability. Next, in Section 6.3 we define our decoy model, and how we learn its parameters from project directories. Then, in Section 6.3.5 we describe decoy directories can be created using such a decoy model. After that, in Section 6.4 we present a user study and a confirmatory case study in form of a simulated data theft experiment that were conducted to evaluate the believability of the generated decoys. Then, in Section 6.6 we give an overview of related work, and find, that none of the existing works have addressed the task of automated, structured decoy generation. We conclude this chapter with a discussion on the limitations of the evaluation and the shortcomings of the proposed method in Section 6.4.3.

6.1 — Introduction

Digital data theft is becoming a significant issue in our ever more digitalized society. The total number of data breach incidents as well as the damage caused are rising at an alarming rate. One way to prevent such data thefts, a security officer could monitor access to file systems, and run data analysis to see whether malicious activities are going on. However, for some monitored actions, it is challenging to understand whether they are malicious or not. For example, consider the act of copying a file from a server. This action could be both, malicious or benign. Malicious, if someone is trying to steal this file, and benign if someone wants to back it up or copy it for legitimate use. In this example, the contextual information about the intent of the action is missing.

One strategy to detect ongoing digital attacks when contextual information about the intent is lacking is to bait thieves with digital decoys. These decoys are intrinsically valueless, therefore any interaction with them is suspicious and will alert the responsible security offer.

One type of decoys is decoy documents. A decoy document is a file which contains seemingly sensitive information. These documents are usually stored with other sensitive documents and closely monitored. Interactions with them are reported and stored. However, while decoy documents are useful tools to obtain intelligence about an ongoing attack, most of a company's intellectual property comprises multiple files and directories, grouped in a project directory. This grouping is especially the case when the intellectual property is software. However, deceptive approaches as detection strategy are hardly used in this setting, since manually creating believable project decoys is a cumbersome, labor-intensive task.

In this chapter, we present a data-driven approach for dynamically creating new project directories from a set of existing directories based on multiple Markov models using maximum-likelihood parameter estimation. After the model is learned, it can be used to sample believable project directories which resemble the original directories and can be used as deceptive bait. This deceptive bait can be placed between "real" projects. Assuming the decoys are adequately monitored, they can aid in the detection of malicious activity, since any interaction with a decoy is per definition suspicious. We have implemented a prototype and conducted a user study to evaluate whether the generated project decoys are perceived as being realistic. Additionally, we have conducted a lab experiment with simulated data thieves in order to demonstrate that decoy projects can aid in detecting ongoing malicious activity.

6.2 — Desired Decoy Properties

When designing decoys, various dimensions can and should be considered. Before we define our decoy model and explain how to create decoys using it, we introduce a list of desirable decoy properties. We then motivate our decision to focus on the believability of these decoys.

Bowen et al., Whitham et al. and Voris et al. introduced partially overlapping set

of properties that effective decoys should have. According to them, decoys should: *be believable [16, 171] and realistic [178]*, be capable of eliciting belief or trust, i.e. capable of being believed appearing true; *be non-interfering [16, 171] and minimize disruption* [178], i.e. should not hinder or obstruct the normal mode of operation; *be enticing [16, 171, 178]*, i.e. be attractive for attackers to interact with; *be conspicuous [16, 171]*, i.e. be easily visible attackers; *be detectable [16, 171]*, i.e. it should be possible to monitor the decoy and notify a security officer of an illegal interaction; *have variability [16, 171] have no distinguishable characteristics [178]*, i.e. if multiple decoys are deployed they should differ from each other in order to prevent counter deception; *be differentiable [16, 171]*, i.e. it should be able for a legitimate user to tell the difference between a decoy and a real object; *have scalable protective coverage [178]*, i.e. it should enable to monitor different types of attacks; *minimize sensitive artifacts and copyright infringement [178]*; *be adaptive [178]*, i.e. - prevent counter deception by blending in to the environment where the decoys are deployed; *be stealth [171]*, i.e. if an alarm of a monitored decoy is triggered it should be done without being visible; *have a specific shelf life [171]*, i.e. more recent data is potentially be more enticing in specific cases.

According to Bowen et al. not all of these properties are applicable in every context, and some of the desired properties even contradict each other, e.g., being believable and being differentiable. Our focus in this chapter is to detect data thefts. Thus we want our projects to be as believable as possible. Therefore, we focus on the believability property when validating our decoys.

6.3 — Decoy Generation

On a high level, our approach can be divided into two phases. First, we learn a decoy model from existing project directories, and then we use this model to sample decoys. The two phases of our approach are depicted in Figure 6.1.

The first step of our learning phase is to normalize our training data. In this step, we replace properties of the source files and directories with placeholders. Examples of such properties are occurrences of the project name or access rights. We use this normalized training data to learn Markov chains and property distributions.

Ideally, one would calculate the probability of all files co-occurring within each other one directory. That is, model the probability distributions of files that often occur together. However, calculating all paired probabilities for n files requires $n * 2^n$ operations, which is unfeasible in practice. Capturing the distribution of folder contents of folders with the same name would only capture the overall distribution of the folder's content, but not the co-occurrence of the files. As a consequence, we approximate the files' co-occurrences with such Markov chains.

We formally define our decoy model in Section 6.3.1, and the normalizations that we use in our prototype in Section 6.3.2. We detail the learning phase in Section 6.3.3, and elaborate the learning giving an example in Section 6.3.4.

We can then use our learned decoy model to sample decoy projects, which is the second phase of our approach. The first step in this phase is to perform random

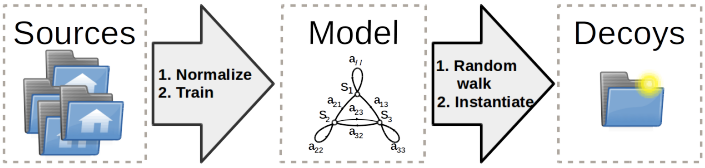

Figure 6.1: Approach on a high level

walks on the stored Markov chains to create an abstract decoy project directory. The random walks are performed recursively on the generated folders, i.e., if a generated folder contains subfolders, the random walk is performed for each sub-folder as well.

The second step is to instantiate this abstract project to a real project directory by replacing the properties placeholders with real values. For example, we replace the project name placeholder with the project name of the decoy. We detail the sampling process in Section 6.3.5.

We base our approach on the assumption that two directories with the same name have similar contents. In this context we use the term similar as in "related in appearance or nature, but not identical". Hence, an optimal decoy would have a file distribution that consisted of the joint probabilities of each file and directory with each file and directory. Since calculating all the joint probabilities for each file and directory is computationally expensive, we reduce the complexity by introducing the assumption that two directories with the same name within a specific context have similar contents. Intuitively, this assumption is based on the idea that humans use directory names to communicate with other humans. That is, directories are named in a way to tell other humans which content they can expect. One can easily verify this assumption in practice. For example, in a directory named "budget", one would expect to find some spreadsheets. Alternatively, in the context of Java software projects, a directory named "src" will most likely contain some packet directories and source code files. The downside of this assumption is that it will not be a correct all the time, for example, a Java project's "src" directory could also contain Javadoc files.

If this assumption does not hold, the learned model will reflect this in the way the samples would be generated. The folder content of decoys that are sampled in such a way could then be chaotic. This chaotic composition of folder contents can potentially give away the deceptive nature of these projects folder to an attacker.

Assumption 6.1. *If two directories within a context have the same relative node name, they have similar contents.*

6.3.1 – Definition of the Decoy Model. In this section, we formally define our decoy model (6.12) in a bottom-up manner.

We will use a (finite) forest, i.e., a finite set of finite trees [35], to model our training file systems.

Definition 6.2. *A* forest *is a finite directed acyclic graph where each node has at most one parent. A node without a parent is called a* root *and a maximal connected components is called a (rooted)* tree*. We use* $\mathtt{ch}(\mathfrak{n})$ *to denote the set of* children *of node* $\mathfrak{n}$*.*

To build the model we will use a training file system that contains many projects which structure objects such as files and directories in the form of a tree and assigns properties to these objects.

Definition 6.3. *We assume a fixed set of objects* O*, a set of attributes* $\mathcal{A}$ *and a (training) file system* TT *over these objects is given. Here an* attribute $\mathfrak{a}$ *is a function from objects to a domain* $\mathtt{Dom}_{\mathfrak{a}}$ *and file system* TT *is a forest over objects.*

Intuitively, Definition 6.3 states that we use a set of project directories as input for our decoy model. These project directories are organized within a directory tree, which is common in Windows- and Unix-based operating systems. An object represents a file or a directory within a project directory, and each file or directory has specific attributes such as a name, permissions or creation- and modification dates.

Some attributes are dependent on the project, for example, filenames containing the project title or creation time of files that will be within the running time of the project. Therefore, to learn a common structure from different projects we "normalize" the project to make them consistent. For example, we replace all occurrences of the project name by a "placeholder". We then refer to the normalized objects as nodes and the normalized TT as NodeTree.

Definition 6.4. *From here on we assume a fixed set* $\mathcal{N}$*, called Nodes, a node tree* NT *over nodes and normalization function* $\mu : \mathsf{O} \to \mathcal{N}$*, mapping objects to nodes, are given.*

The only attribute that we assume is always there is "*name*". The other attributes may depend on the operating system. Therefore, we do not further specify them here. Instead, we assume they are available. We provide an overview of the transformations that we used to normalize the attributes of in the evaluation experiments in the Appendix.

In Assumption 6.1 we assume that similar names convey similar contents. Therefore, we group nodes (i.e., normalized files and directories) by their name. We treat a node which represents a file as a node without children.

Definition 6.5. *A* ***NameGroup*** $\mathtt{NG}(\mathfrak{d})$ *for a name* $\mathfrak{d} \in \mathtt{Dom}_{name}$ *is the set nodes that have name* $\mathfrak{d}$*.*

$$\mathtt{NG}(\mathfrak{d}) = \{\mathfrak{n} \in \mathsf{N} \mid \mathit{name}(\mathfrak{n}) = \mathfrak{d}\}$$

Next, we need to estimate the probability of specific nodes co-occurring as children of a node with the same name. In other words, we need to learn the content of a directory. Just taking their frequency may not be sufficient; some nodes will naturally

occur together or will exclude each other. As such a combination might allow an attacker to spot a fake project directory quickly, we need to take such dependencies into account.

Definition 6.6. *A **NameGroupChildren*** $\mathrm{NGC}(\mathrm{d})$ *for a name* $\mathrm{d} \in \mathrm{Dom}_{name}$ *is a set of sets, defined as:*

$$\mathrm{NGC}(\mathrm{d}) = \{\mathrm{ch}(\mathrm{d}') \cup \{\mathrm{s_s}, \mathrm{s_e}\} \mid \mathrm{d}' \in \mathrm{NG}(\mathrm{name}(\mathrm{d}))\}$$

Here, $\mathrm{s_s}$ *represents an artificial node "start state", i.e. the start of a directory and* $\mathrm{s_e}$ *represents the end of a directory.*

Learning the complete joint probability of all nodes with any accuracy is not very realistic because the number of calculations grows exponential with the number of (possible) children nodes of a node. As a compromise, we use Markov chains to model the probabilities. Markov chains allow taking into account pairwise dependencies between the nodes. In order to train our Markov chains, we will treat the children of the nodes of a NameGroup as encountered observations.

Given this construction, the order of the nodes matters; dependencies are only taking into account between the current and next node. After visiting an intermediate node this information is lost. Nodes with a relevant dependency on each other should occur next to each other to ensure the model express the dependency.

Definition 6.7. *To help optimize the amount of information about dependencies that we can capture in our model we use a simple, greedy heuristic "*`sorted`*" where we attempt to maximize the mutual information* $\mathrm{I}(\mathrm{n_i}, \mathrm{n_{i+1}})$ *between pairwise successive nodes within one NameGroup. "*`sorted`*" starts with the start state* $\mathrm{s_s}$*, followed by nodes that are always present. We exempt the nodes that are always present, as calculating the mutual information for these nodes will not improve the information about the dependencies. As a next step, the heuristic continuously selects follow-up nodes with maximum mutual information* $\max(\mathrm{I}(\mathrm{n_i}, \mathrm{n_{i+1}}))$*, where* $\mathrm{n_{i+1}}$ *has not been selected before. In case of a tie we prefer nodes that occur together often, i.e.* $\mathcal{P}(\mathrm{n_{i+1}}|\mathrm{n_i})$ *is higher. "*`sorted`*" always ends with the end state* $\mathrm{s_e}$*, as this is per definition the last state.*

Next, we want to learn a Markov chain with states S and a transition matrix P for a name group NG. That is, we want to learn a Markov chain that represents the distribution of files and directories of all directories with the same name (see Assumption 6.1). To do so, we treat the sets of nodes $\mathrm{ngc} \in \mathrm{NGC}(\mathrm{d})$ as a sequence of "observed events".

Definition 6.8. *Let* d *be a name* $\mathrm{d} \in \mathrm{Dom}_{name}$ *and* $\mathrm{NGC}(\mathrm{d})$ *the name group children of* d*. Then, we define the **states*** S *of the Markov Chain* MC *for a* NGC *with:*

$$\mathrm{S} = (\bigcup_{\mathrm{ngc} \in \mathrm{NGC}(\mathrm{d})} \mathrm{ngc})$$

In definition 6.8, $\bigcup$ denotes the union of sets. To illustrate definition 6.8, consider the following example. Let us assume a the NoudeGroupChildren of a folder named "src", $\mathrm{NGC}('\mathrm{src}')$ which has two elements, $\mathfrak{ngc}_1, \mathfrak{ngc}_2$, which each have two element themselves, i.e. $\mathfrak{ngc}_1 = A, B$ and $\mathfrak{ngc}_2 = B, C$. Then the states S for the Markov Chain $\mathrm{MC}_{\mathrm{src}}$ is: $S = \mathfrak{ngc}_1 \cup \mathfrak{ngc}_2$, which is equivalent to: $S = A, B \cup B, C$, which is equivalent to: $S = A, B, C$.

Furthermore, from now on we refer to a Markov chain MC that for the NameGroupChildren of a node $\mathrm{NGC}(d)$ as MC_d. Next, we estimate transition probabilities of MC_d.

Definition 6.9. *Let us assume that each* $\mathfrak{ngc} \in \mathrm{NGC}(d)$ *has been sorted by the heuristic which was introduced in Definition 6.7. Then the transition probabilities* P *for two successive nodes* $\mathfrak{n}_i$,$\mathfrak{n}_{i+1}$ *of a Markov chain* MC_d *are estimated using a Maximum Likelihood Estimators, using each* $\mathfrak{ngc} \in \mathrm{NGC}(d)$ *as an observed sequence of events.*

The whole structure of our decoy model is combined in a tree model.

Definition 6.10. *A* ***tree model*** *TM is a function that maps a name* $d \in \mathrm{Dom}_{name}$ *to a corresponding Markov chain* MC_d.

$$\mathrm{TM} : d \rightarrow \mathrm{MC}_d$$

In order to capture the probabilities of the node attributes, we learn the probability that certain nodes have. Here Assumption 6.1 is used to cluster the group of properties.

Definition 6.11. *An* ***attribute model*** *captures the frequency of occurrence of an attribute value within a name group. It is a function from* $\mathcal{N}$ *to probability measures* $\mathcal{P}$ *over the domain of* $\mathfrak{a}$. $\mathcal{P}$ *is defined as:*

$$\mathcal{P}(\mathfrak{a}(\mathrm{NG}(\mathfrak{n})) = x) = \frac{\#\{\mathfrak{n}' \in \mathrm{NG}(\mathfrak{n}) \mid \mathfrak{a}(\mathfrak{n}') = x\}}{\#\mathrm{NG}(\mathfrak{n})}$$

and the attribute model AM as:

$$\mathrm{AM} = \mathfrak{n} \times \mathfrak{a} \rightarrow \mathcal{P}(\mathfrak{a}(\mathrm{NG}(\mathfrak{n})))$$

Finally, using the previous definitions, we define our decoy model as follows:

Definition 6.12. *A* ***decoy model*** DM *consists of a tree model* TM *and an attribute model* AM. *Intuitively, the tree model captures the learned, normalized directory structure, whereas the attribute model contains the learned, normalized file- and directory attributes.*

6.3.2 – Normalizations. We normalize our training data in order to minimize project dependent properties in our decoys. Such project dependent properties could easily give away the deceptive nature of the project or reduce the attractiveness of the decoy project to a potential attacker. For example, in a project named X, an attacker would expect a file "main-file-X", not "main-file-Y". Similarly, an attacker could get suspicious if a file or folder with a for this context suspicious owner or date. In this section, we define the transformations that we have implemented in our prototype.

We replace occurrences of a project name in all file- and directory names with a placeholder to eliminate obvious references to a project.

Definition 6.13. *Let* N *be a set of names* $\mathfrak{n}$, $\mathfrak{n} \in \mathsf{N}$, PN *a project name and* PL *a placeholder. Then our name normalization function is defined as:* $\mu_{\mathsf{N}}(\mathfrak{n}) = \mathfrak{n}'$ *where* $\mathfrak{n}'$ *is* $\mathfrak{n}$ *with all occurrences of* PN *replaced with* PL.

For example, consider a project named "A", a placeholder "XXX" and a filename with "A.cpp". The normalized filename is "XXX.cpp", as the project name A was replaced with the placeholder "XXX".

Furthermore, in our prototype, we normalize creation timestamps, last-accessed time stamps, and last-modified timestamps. This normalization allows us to create decoys with a believable time stamp distribution, even though the training sources are from a different time period.

Definition 6.14. *Let* T *be a set of Unix time stamps* $\mathfrak{t}$, $\mathfrak{t} \in \mathsf{T}$ *and* $\min(\mathfrak{t})$ *be the minimum element of* T. *Then our time normalization function* $\mu_{\mathfrak{ts}}$ *is defined as:* $\mu_{\mathfrak{ts}}(\mathfrak{t}) = \mathfrak{t} - \min(\mathfrak{t})$.

Finally, we normalize the owners and the groups of our project files and directories. We only define the normalization function for file ownership here, since the definition of owner group ids is similar.

Definition 6.15. *Let* W *be an indexed set of owner ids* $\mathfrak{w}$. *Then our ownership normalization function is defined as:* $\mu_W(\mathfrak{w}) = \mathfrak{i}$, *where* $\mathfrak{i}$ *the index of* $\mathfrak{w}$ *in* W.

For example, let W be $\{1001, 1002\}$. Then $\mu_W(1001) = 1$, since it is the first element in the set.

6.3.3 – Learning the Decoy Model. Summarizing Section 6.3.1, we present the algorithm to learn a decoy model from a set of project directories. Algorithm 6.1 illustrates the steps of this process in pseudo code. We split the algorithm into two phases, normalization and learning the decoy model (see Figure 6.1).

First, we normalize the input files and directories as described in Definitions 6.13, 6.14 and 6.15. We traverse through all our training files and directories and apply our normalization function on them (Lines 1 - 3).

After that, we learn the decoy model in phase 2. We execute the following four steps for all unique node names $\mathfrak{d}$ in our training nodes (Lines 4 - 10).

1. The first step is to apply our sorting heuristic to the NameGroupChildren of the current node name d.
2. We then use the sorted NameGroupChildren to learn the Markov chain MC_d for this NameGroup d, see Definition 6.9. We store this Markov Chain in our Tree Model TM (Line 7).
3. Finally, we estimate the attribute probabilities for each node of this NameGroup and their children, see Definition 6.11 (Line 8) and store them in our attribute model AM.

Algorithm 6.1 Learning a decoy model in pseudo-code.

```
Data: A training file system TT
Result: A decoy model DM
/* 1.  Normalize data                                                    */
4 for o ∈ TT do
5 |  n ← μ(o)
6 end
  /* 2.  Learn decoy model                                               */
7 for d ∈ unique(NodeNames) do
8 |  NGC(d)′ ← sorted(NGC(d))  TM_d ← learnMM(NGC(d)′)  AM_d ← learnAM(NGC(d)′)
9 end
10 DM ← (TM, AM)
```


6.3.4 – Example: Learning a Tree Model. To illustrate the learning process of a Markov chain of our tree model, we provide an example. Let us assume three nodes which belong to the same NameGroup $NG(f) = \{n_{f1}, n_{f2}, n_{f3}\}$ and their NameGroupChildren $NGC(f) = \{\{n_1, n_3, n_4\}, \{n_2, n_3, n_4\}, \{n_2, n_3\}\}$. In other words, we assume three directories with the same name but from different projects and their content.

First, we define the sorting order for all elements of our NameGroupChildren(f) by using our greedy sorting heuristic (see Definition 6.7). The mutual information of the pair $I(n_1, n_2)$ is 0.64, the mutual information of the pairs $I(n_1, n_4)$ and $I(n_2, n_4)$ is 0.17 and the mutual information of all other pairs is 0. This results into the following order: $n_1 < n_2 < n_4 < n_3$.

Next, we sort the NameGroupChildren(f) using the previously defined order, prepend a start state S_s and an end state S_e. This results in to the $NGC'(f)$ as follows:
{
$\{S_s, n_1, n_4, n_3, S_e\}, \{S_s, n_2, n_4, n_3, S_e\}, \{S_s, n_2, n_3, S_e\}\}$.

After that, we estimate the transition probabilities of a Markov chain by using our sorted NameGroupChildren. The order for estimating the transition probabilities is essential. For example, there is no transition from n_3 to n_4, as in all NameGroupChil-

dren n_3 is followed by S_e.

We visualize the estimated Markov chain after adding all ngcs to the chain in Figure 6.2. In this figure, we depict the Markov chain as a series of as directed, acyclic graphs. The vertices in these graphs represent nodes, i.e., normalized files or directories. Edges represent a transition from a node to another, i.e., we would observe file after the next one when looking at a directory listing. Labels state the transition probability. For readability, we have removed all edges of transitions that will never be observed.

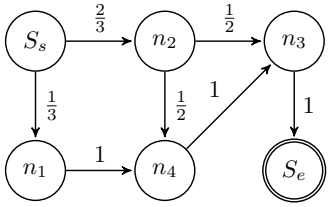

Figure 6.2: The Markov chain for NameGroup(f)

6.3.5 – Sample Decoy Project Directories. In this section, we describe how we sample decoys using our previously learned decoy model. The sampling algorithm is depicted in Algorithm 6.2.

To create project decoys, we rely on a recursive sampling function. The purpose of the sampling function is to perform a random walk of the Markov Chain for a given directory name. This random walk samples the folder contents the current directory. These folder contents may also contain sub-directories. If that is the case, the sampling function is recursively applied on these sub-directories. In detail, the steps of the sampling function are:

1. First we load the Markov chain $\mathrm{mc_d}$ and the attribute model $\mathrm{am_d}$ for this directory d (Lines 2, 3).
2. Next, we perform a random walk on the Markov chain $\mathrm{mc_d}$, starting by its start state $\mathrm{s_s}$ and stopping by its absorbing state $\mathrm{s_e}$. The output of the random walk $\mathrm{walk_d}$ is a set of nodes that will be used to populate the current directory d (Line 4).
3. After that, we will instantiate each node of the random walk (Lines 5-8). That is, we randomly select the attributes for a node according to the stored attribute distribution (Line 6). We then create the file from the node to the current directory by replacing the placeholders with the according to instantiation parameters (Line 7). This process is the "reverse" process of the normalization.
4. Finally, we will recursively call the sampling function for each file and directory that just has been created in d (Lines 9-12).

To invoke our sampling algorithm, we first need to create an output directory for our decoy project (Line 13). Then we call the sampling method on the project root (Line 14), which will then recursively generate the project decoy. The instantiation parameters (e.g., the project name) depend on the implemented normalizations and are provided manually in our case.

Algorithm 6.2 Sampling a decoy project

```
Data: A decoy model DM, instantiation parameters
Result: A decoy project
/* Recursive sampling function                                    */
11 Function (d) sample
12 |  mc_d ← TM(d)  am_d ← AM(d)  walk_d ← randomWalk(mc_d) /* 1.
   |     Populate directory d                                     */
13 |  for n ∈ walk_d do
14 |  |  a ← randomChoice(am_d(n)) instantiate(n, a)
15 |  end
   |  /* 2.  Sample sub-directories                               */
16 |  for d′ ∈ ch(d) do
17 |  |  sample(d′)
18 |  end
19 end
   /* Execution                                                   */
20 createProjectDirectory() sample(projectroot)
```

It is worth noting that with this approach it is possible to generate project decoy directories which contain similarly named sub-directories. However, this is also possible and common in real projects.

6.4 — Evaluation

Our main goal is to detect an ongoing data-theft, which is why we consider believability to be the most important decoy property (see Section 6.2). We attempt to validate the believability of our decoys in two ways: via a user study and via a lab experiments. We present a user study where we ask human judges to distinguish between real project directories and decoys in Section 6.4.1. Then we report on the results of a simulated data theft experiment which we have conducted to collect additional evidence for the believability of the decoys in Section 6.4.2. We conclude by discussing these results in Section 6.4.3. To conduct the experiments, we have implemented a prototype using Python 2.7

6.4.1 – User Study: Evaluating Decoy Believability. One of the main goals of deception is to condition a target's beliefs. In other words, we want an attacker to believe that our generated projects are real. We have conducted a user survey to test

whether human judges can distinguish between real projects and decoy project. We evaluate this user study from a machine learning perspective. That is, we treat the judges as (human) classifier which is tasked to classify a given project to real or decoy.

Survey setup. In this study, we presented 30 software project directories to 27 human judges. Half of the presented software project directories were generated using our prototype. The other half were real project directories. The study participants were asked to decide which ones were real and which ones were decoys. The software project directories were shared, and the participants could click around freely within the projects.

We learned three different decoy models, one for Java projects, one for Python projects and one for Ruby on Rails projects. We chose three different project types to identify differences in the trained model and the effect of the believability of the decoy.

The source projects for the three project types had the structure and properties. On average (mean), the Java projects had 740.6 files, 207.4 folders and a folder tree depth of 4.3); on average(mean), the python projects had 53.5 files and 4.2 folders and a folder tree depth of 1.6; on average (mean) the Ruby on Rails project had 74 files, 37.2 directories, and a folder tree depth of 2.7. The folder tree depth is calculated from the root project folder.

In total, we used 25 open source projects for each project type to learn the decoy models. The number 25 was determined using trial and error and based on believability and variability of generated test decoy projects.

The "real" projects that we use for training were different from the "real" ones that we presented to the survey participants. We obtained the projects by searching GitHub for certain terms, e.g., "Java and Security" and chose popular projects for related topics. Since evaluating 30 projects would take a long time, the study was split into three parts, one for each project type. Each participant was free to choose which of the parts and the number of parts they wanted to complete. Each partial survey contained a section with demographic questions.

We recorded the answers to the surveys as well as the clicks of the participants within the directories. We neither counted survey answers where we did not have any corresponding clicks nor clicks on directories that could not be correlated to participants of the study. The data was collected within the period from February 1st to February 8th in 2016.

In total, we had 27 unique participants, and we have collected 23 complete surveys for Java, 19 for Python and 15 for Rails projects resulting in 570 answers. We did not count 27 answers because there were no clicks on the corresponding projects. Thus in total, we collected 543 valid judgments.

A majority of our study participants (81.5%) were highly educated, i.e., had a college degree or a post-graduate degree and also most of them (70.37%) rated their

software development skills at least average or more. Most of the participants (78.3% / 68.3%) who responded to the Java and Python part of survey stated that they have at least some knowledge of the Java programming language / Python programming language respectively, whereas only 20 percent of the participants had some knowledge of the Rails framework. On average, on a scale from 1 to 5 with 5 being the highest the Java knowledge of our participants was 2.87, the Python knowledge 2.43 and the average knowledge of the Rails framework was 1.40.

To avoid participants' bias towards project names, we replaced the project names that occurred in all files and directories of a project with a randomly numbered placeholder of the form Pxx, where xx was a randomly chosen number between 01 and 20. The study participants were informed about this measure. Apart from that, the participants knew that they were presented partially generated and partially real directory listings. To counter any bias originating from this knowledge, we phrased the evaluation question in a non-suggestive way, i.e., "*The number of real/decoy projects does not necessarily have to be balanced. There may be 0 real projects, and there may also be up to 10 real projects.*". The actual distribution of real and decoy projects per survey was balanced. The total number of answers that we received on decoys was 272, the total numbers that we received on real projects was 271.

Survey results. Table 6.4 shows the confusion matrix of the participant's answers and their accuracy in evaluating the real projects as well as the decoys. This table summarizes the correct and the types of errors that were made, i.e., it shows how many of the real projects were correctly identified as real, how many

Figure 6.3 shows the accuracy of the individual judges. The minimum recorded accuracy was 30%, and about a quarter of the participant's accuracies were below 50% total accuracy while the highest accuracy was 90%. The total average individual performance was 53.3%, which is slightly better than random guesses. Apart from that, the average accuracy per project type was 52.19% correct answers for Java projects, 53.63% for Python projects and 46.32% for Rails projects. In total 51.20% of the 543 answers were correct.

The number of answers on decoys and real projects has been balanced in the surveys. Since the number of answers on real projects was 271, and the number of answers on decoy projects was 272, a combined majority class classifier has have an accuracy of 50.0%. A majority class classifier for the java projects has an accuracy of 50.0%, a majority classifier on the Python projects 50.2%, and a majority class classifier on the Rails projects has an accuracy of 50.0%.

6.4.2 – Case study: Simulated Data Thefts. In this section, we describe the setup and results of an confirmatory case study which simulates data thefts. We have conducted this case study to test whether decoy projects can be distinguished in a lab setting.

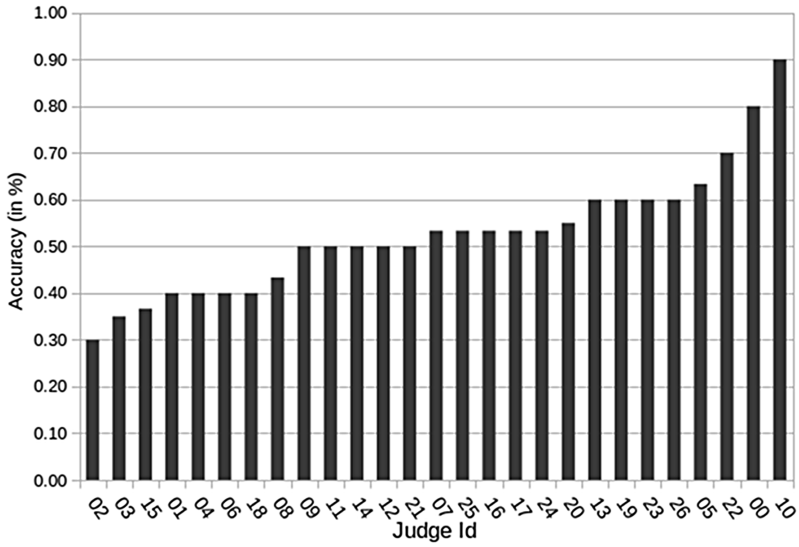

Figure 6.3: Accuracy per person

Figure 6.4: Confusion matrix of valid judgments per project (n = 543). The number in the brackets is the number of judgments relative to the total number of answers per project type.

Knowledge	Answers		
Project Type	Answer		Correct
	Real	Decoy	
Java real	67 (29.3)	47 (20.6)	58.77
Java decoy	62 (27.2)	52 (22.8)	45.61
Python real	45 (25.1)	44 (24.6)	50.56
Python decoy	39 (22.8)	51 (28.5)	56.66
Rails real	31 (22.8)	37 (27.2)	45.59
Rails decoy	36 (26.5)	32 (23.5)	47.06
Total real	143 (26.3)	128 (23.5)	52.76
Total decoy	135 (24.9)	137 (25.2)	50.37

Experiment Setup In this experiment, the participants had to simulate a disgruntled masquerader (see [127]), who wants to steal software from a company server for profit.

First, the participants in the experiments were familiarized with the scenario of the experiment, i.e., they were working for a company that produced security software. They got fired, it was their last day of employment, and in order to take revenge, they wanted to steal some of the software and then sell it on the black market. The participants knew the IP-address of the development server which hosted the software, got the credentials of a co-worker who left it on a post-it under his/her keyboard. The participants did not know which projects the co-worker was working.

Every participant obtained their user account. The assignment of the participants was to log on to the server, find the software projects and identify and steal valuable ones. The participants were instructed to maximize their profit, however, were given a choice whether or not to steal projects at all. Furthermore, they were instructed to motivate their selection of projects to steal as well as each of their performed actions. The participants were also informed that all projects were taken from GitHub and that they could ignore the fact that they were open source.

Additionally, the participants were warned that an anomaly detection system was being used. This anomaly detection system would trigger an alarm if more than three projects were stolen at the same time. We added this constraint because (i) we wanted to prevent data thefts were a student would copy all projects at the same time, (ii) we wanted them to know that they were being monitored so that they would act more cautiously.

In total we prepared ten software projects on the server for each participant. 8 of them were Java projects that we had downloaded from GitHub previously. The other two projects were decoys that we generated using the Java decoy model of our believability experiment (see 6.4.1). The decoys were named after two other real Java projects on GitHub. All software projects were about security technologies. We monitored access to each file and directory as well as commands entered by the participants. The participants were NOT informed that they should look out for decoys.

We conducted this experiment with 33 security-minded, undergraduate students, 25 of them male and 8 of them female. Before experimenting, the students were following a lecture on deception in IT security. Decoy technology for detecting attacks was part of the lecture. The experiment was conducted between February 24th, 2016 and February 27th, 2016.

Results 304 of the 330 project directories were accessed throughout the experiment. On average, while identifying valuable targets, a participant accessed 7.42 (80.59%) of eight real projects and 1.79 (19.41%) of two decoy projects.

29 of the participants "stole" at least one software projects, and four refused to steal anything at all. In total, 49 real and 11 decoy projects were stolen, i.e., 81.67% of

the stolen projects were real. The three most frequent reasons for selecting a project to steal were: exciting content (15 times, 51.72%), project name (7 times, 24.13%) and project size (6 times, 20.69%), with multiple reasons being possible. 28 of the 29 participants who stole a project did not steal all ten projects because they were afraid that the anomaly detection system would trigger an alarm if they stole more. In average, each data thief, i.e., a participant that stole at least one project, stole 2.07 projects. Out of the four participants who refused to steal a project, three justified their actions by their fear that the whole setup was a deception attempt, therefore the risk of stealing data would be too high. The remaining participant refused to steal data because he had ethical concerns to involve an innocent co-worker in a data-theft crime.

All 29 participants that decided to steal a software project had accessed a decoy project before, which equals a detection rate of 100% in this setting. Apart from that, in total 14 projects were suspected to be a decoy project, four (28.58%) of them were actual decoys, and 10 (71.42%) of them were real projects.

6.4.3 – Discussion. A perfect decoy would be indistinguishable from the object it tries to mimic, therefore ideally the performance of the judges should be close to random. In our believability study, the participants were able to identify real projects and decoys in 51.57% of the answers. In our data-theft experiment, the participants stole 18.33 percent of the decoys, which differs 1.67 percent from 20 percent of the projects they would have stolen if they acted purely random. Additionally, none of the participants in neither experiment were able to differentiate between real projects and decoys correctly. These results suggest that the participants could hardly tell the difference between a decoy and a real project, regardless of whether they were able to look at the content of files or not.

We assume that a value of a project that is stored in a directory can only be determined if the contents of the directory were accessed. Moreover, we assume that a potential rational data thief wants to estimate whether the data they are accessing is valuable, i.e., a data thief will not steal random directories. The participants in our experiment, who were asked to assess the value of the project that they were about to steal accessed in average 9.21 out of ten project directories, which suggests that in this experiment this was indeed the case.

In all of the cases where a project was stolen a decoy project was accessed before. Assuming proper monitoring of the project checkouts, this equals a detection rate of 1.00 of all data thefts in our experiment. This suggests that decoy projects are also believable in this setting.

As of now, attackers can be pretty certain that each project they are looking at is real. Another benefit of using decoy projects is that data thieves then have to spend additional effort to evaluate the projects on their realism. That is unless they want to risk being detected or stealing valueless decoys.

6.5 — Limitations and Future Work

We have presented a data-driven approach to creating decoy projects. These decoys may aid detecting insider attacks by placing them closely monitored between real projects. We have empirically shown that the generated decoys are believable and can be used to detect data thefts. However, we did so in a limited way. In this section, we want to discuss the limitations of our setup as well as the limitations of our approach.

Whether or not a project is considered realistic depends on many different variables such as the project name, the context where the project occurs or the content of the files. We based our experiments on freely available open source projects. The choice of projects positively influences whether or not a project is perceived to be a decoy. For example, comparing a well-structured decoy to a chaotic, half-finished real project may change believability in favor of the decoy. To mitigate this, we have selected the real projects for evaluations according to their popularity on GitHub, assuming that this also indicates a certain level of believability.

In both experiments, we do not take into account some of the factors that can reveal a decoy. For example, we ignored that an attacker could perform long-term passive reconnaissance on the projects or use different channels to ascertain the value of projects.

Also, in our simulated data theft experiment, the attackers had only a time-window of three days to steal the projects. In a realistic environment, the time span between evaluation and an actual theft can be much larger, which renders actual detection more difficult. However, at least a suspicious activity would have been reported when the attacker first assessed the projects.

Another limitation of our methods is that the sample of participants of the user study (see 6.4.1) and the simulated data theft experiment (see Section 6.4.2) are very likely not representative for the population of an attacker, which limits the conclusions that one can draw about the perceived believability for the desired target population.

Apart from that, the simulated data theft experiment has not been set up as a controlled experiment, but rather as a confirmatory case study. Such a case study does not provide any insights into the external validity of the results, but it would be useful to refute a hypothesis. Our results suggest that the students were not capable of distinguishing between real and decoy projects in a lab setting. As a consequence, we cannot refute the hypothesis that they can distinguish, but draw no further conclusions from that.

We evaluated the perceived believability of the generated decoys by treating the human judges as a classifier. The way this user study has been set up has implications on the conclusions that one can draw about the generalizability of the results. Instead of providing evidence for the hypothesis that the human judges are not capable of distinguishing between real and decoy projects with reasonable confidence, our evaluation methodology provides insights on the performance of the human judges on the task and provides expectations on future performance.

Our proposed approach has some limitations. First of all, currently we need to

specify instantiation parameters such as the name of the project decoy manually. We assume that these parameters contribute to the effectiveness of our approach, thus ideally they are also learned from the source projects as well as the target context.

Next, our approach will not work if an attacker already knows which project they want to steal, because then they do not need to browse other projects to determine their value. In addition to that, we are not able to tell the reason why an attacker accessed a decoy directory. For example, a decoy directory could be accessed by accident or out of curiosity without evil intention.

Another shortcoming is that our approach does not generate content. This shortcoming is especially crucial when the material that is used to learn the model contains sensitive information. In our prototype, we have implemented two simple strategies to mitigate this problem, i.e., fill the files with random data and copying data from a public file repository instead of using the real files. In addition to that, our approach potentially leaks sensitive information about the structure of the projects and the names of files and directories that we used for training the model.

To address the limitations mentioned above and thereby improve our approach, we plan to include methods for learning and generating the patterns of existing file names so that we can create new ones. Furthermore, we believe that hybrid methods which are based on templates as well as on data could further improve the believability of the generated project directory structure. Apart from that, we are investigating methods to learn instantiation parameters such as the project name from existing sources.

6.6 — Related Work

Deception as an element of warfare has been used through history and has continuously evolved. Nowadays the use of deception is tightly ingrained in military doctrines around the world. When computers grew more important, and digitally attacks on these systems started to happen, deception soon became a part of the digital strategy to protect computer systems and network from malicious individuals and groups. One of the first published reports on using digital deception was Clifford Stoll's report on his attempts to catch a German hacker working for the Soviet secret service [145]. Lance Spitzner coined the term honeypot and honeytoken [143], which are descriptive synonyms for computing resources that are created to deceive an attacker. Since then a variety of so-called "honey objects" or "honey approaches" have been proposed, where the term "honey"indicates that deceptive measures are part of the proposed approach. Bringer et al.'s study elaborates recent advances within the field of honeypots [17].

A few works on deceptive files and file structures have been reported in the literature. Yuill et al. coined the term "Honey Files" [191]. In their work, they placed enticingly named files within a honeynet to track intrusions. Rowe has described an approach to generate fake on-line file systems [125]. To generate a decoy file system, he relies on randomly copying files from existing file systems as well as randomly

changing the structure of the directories. Bowen et al. designed a system which generates and distributes fake documents in order to detect insider attacks [16]. The approach includes generating and exposing documents including fake tokens and multiple ways to monitor whether abuse has happened. Ben Whitham introduced the concept of Canary Files [177]. While a honey document should entice an attacker to interact with it, a canary document tries to blend in perfectly with the surrounding real files in order to aid detection of unauthorized file operations. In addition to that, Ben Whitham introduced the Serinus system [176], which automates canary file generation and aims to maximize the files' capability to blend in. The Serinus system uses different sources to generate the file names, such as a set of harvested documents from the Internet, the enterprise environment as well as the target directory where the canary file should be placed. Finally, Voris et al. suggest using automated translation to improve the generation of honey documents to force the attacker to spend more time to translate and evaluate whether it is his targeted file [170].

In this chapter, we introduce a data-driven approach to create believable project directories that can be deployed to detect data theft attempts. Previous approaches focused on creating and placing single decoy documents([191], [16], [170], [177]). Conceptually we also create decoys to detect data theft. However, instead of creating single documents we focus on creating project directories with a believable structure, file names, and properties.

6.7 — Chapter Summary

In this chapter, we presented a data-driven approach based on first-order Markov chains that can be used to automate the generation of decoy project directories within a specific, local context. We implemented a prototype of this approach, generated decoy project directories and validated their perceived believability via a user study with 27 participants and 543 evaluated projects, which treats human judges as a classifier that has to decide whether a given project is real or not. This classifier was evaluated on three different project types. Of all judgments, 26.3% were true positives, 25.2% were true negatives, 23.5% were false positives, and 24.9% were false negatives. A truly random classifier would judge 25% to each correct and wrong judgment. Hence, we conclude that the performance of further human judges will be close to random performance as well. A random performance for a binary classification task means that a classifier is not capable of distinguishing the analyzed objects.

If deployed in a real scenario, such decoys could obtain additional information about users' intentions, which indirectly addresses the challenge of the lack of contextual information.

In detail, this chapter contributes to the answer of Research Question 4 in the following two ways:

- We presented a novel method for the automated generation of believable decoy project directories. This method learns the structure of project directories in a data-driven way instead of being manually created, which reduces the labor

involved in creating such decoys.

- We evaluated the believability of the generated decoys via two means: a user-study and a simulated data-theft experiment, which both suggest that the generated decoys cannot be distinguished from real projects.

Chapter 7

Conclusions

Many tasks in information security require data to be analyzed to provide security to systems and information. The quantity of data that needs to be analyzed is huge, and it is expected to grow exponentially. Manually sifting through the data to protect information systems is cumbersome and costs a lot, as it consumes much time of security experts. Hence, in many cases, security experts desire to automate such information security tasks.

One way to achieve such automation is by using machine learning algorithms. Machine learning algorithms learn models from data such that they can solve a well-defined task. For example, a machine learning algorithm can learn to recognize malicious software binaries from a set of malicious and a set of benign binaries.

However, to apply machine learning algorithms to information security tasks, a number of challenges need to be addressed. In this thesis, we focus on three of these challenges: the lack of supervision, the integration of domain knowledge and the lack of contextual information. The first challenge that we addressed is the lack of supervision. One of the most commonly used type of machine learning algorithms, i.e., supervised machine learning, needs supervision in the form of labeled data to learn. However, such labels are often not available, and the validity of labels for information security is often short because the context changes frequently. The second challenge that we addressed is the integration of domain knowledge. In some cases, domain knowledge is available, but it cannot readily be integrated into the machine learning methods, as these methods are designed to learn from data. The third challenge that we addressed is the lack of contextual information. Often in information security, logs need to be analyzed to determine whether a user's sequence of actions is malicious or not. However, in cases this information is not sufficient, and additional contextual information is required to learn the intent of the user.

To address these challenges and identify concrete problems we dived into two representative use cases. The first use case is selected from the domain of information forensics, while the second use case from intrusion detection, more specifically, data theft detection.

For the first use case, we selected the problem of signature extraction from forensic logs and its related problem, log clustering of forensic logs. In forensic log analysis,

labels are typically not available. Furthermore, the context of forensic log analysis continuously changes, which would render existing supervised data in any form obsolete quickly. Moreover, domain knowledge about the logs is readily available, as the logs are typically created by software and operating systems of the computer systems that are analyzed by the forensic investigator. Therefore, we consider this problem to be representative for the first two challenges, the lack of supervision and the integration of domain knowledge to machine learning. For the second use case, we chose the problem of data leakage detection. In this context, actions that are monitored lack information about the intent of a user. This lack of clarity about the intent renders the detection of data theft challenging because an action such as copying a project could be both malicious and benign: malicious if the user copies a project to steal it, benign if the user copies the project to back it up. Consequently, we consider this problem representative for the lack of contextual information.

In this thesis, we addressed these three challenges by asking four research questions that we derived from these two use cases. In the first use case we were addressing security tasks that deal with forensic logs. Forensic logs are semi-structured sequences of tokens, which means that the input data for a machine learning problem is high-dimensional, and the tokens have complex relationships with each other.

To understand whether machine learning algorithms are capable of modeling the structure such data, we asked our first research question: *How can we automate the signature extraction of semi-structured log sequences with labeled data?*. This question led to a deep learning-based method for signature extraction. We found that deep-learning methods are well-suited of modeling such semi-structured log sequences and outperformed state-of-the-art methods on the forensic log clustering task. If that were not the case, attempts to address the lack of supervision would likely fail.

However, as previously mentioned, labels for forensic logs are not readily available in the context of a forensic analysis. Hence, we asked the following research question: *How can we automate the clustering of semi-structured log sequences without supervised data?*. Answering this question led to a deep-learning based method that clusters forensic logs according to their signatures without relying on supervision in the form of labeled data. Such clustering can be helpful as a pre-processing step in the further forensic analysis. For example, an investigator may use such clustering to find recurring patterns of logged actions such as system startups or user logins.

Often labels are not available, but other domain knowledge is. For example, in the context of the task of clustering forensic logs to their signatures, domain knowledge such as the edit distance between loglines indicates whether these loglines should be clustered together. Machine learning is designed to learn structures directly from data. Hence it is difficult to use such knowledge. Therefore, we asked the following research question: *How can we use approximate domain knowledge to aid the automation of semi-structured log line clustering?*. Addressing this question led to a deep learning-based method that allows to perform clustering of forensic logs by integrating such domain knowledge. Additionally, it allows combining the available domain knowledge with supervised data. Which means that labels can be used much more efficiently, or

heuristics can be refined by adding a few labeled examples.

The main challenge in the second use case was that merely analyzing logs did not reveal the intent of an action. Such a lack of contextual information meant that an action can be either malicious or benign. One way to address this lack of contextual information is by using deceptive mechanisms to gather the missing information. In our case, we wanted to create worthless decoy projects that would be placed on a project repository between real projects. Since these projects are worthless, any interaction with them is suspicious. Manually creating such projects that are believable, i.e., look like real projects is cumbersome. Consequently, we asked the following research question: *How can we automate the generation of believable decoys objects from structured data?*. By answering this question, we developed a method that used Markov-chains to generate decoys projects from existing projects. We verified with a user study that generated decoys were hard to distinguish from real projects, which is an essential requirement for such decoys that they can be deployed.

For these two domains, we advanced the state of the art in the following ways. First, for the forensic log analysis, we proposed a method for signature extraction from forensic logs. The novelty of this method is to use a machine learning model to predict the variable and fixed parts such that in a post-processing step the signatures can be extracted. Next, we proposed an unsupervised method for clustering forensic logs along their signatures. The novelty of this method is to use a machine learning model to reduce log lines to a lower dimensional representation such that their sequence structure is captured. This reduced log-line representation can consequently be used for clustering the lines according to their signatures. Then, we proposed a method for integrating domain knowledge with machine learning. The novelty here is a customized loss term, which enables the integration of the domain knowledge. Also, this loss term allows combining labeled data with domain knowledge, which leads to a more efficient use of the labels. Finally, we proposed a method a data-driven method for creating believable decoy projects. The novelty of this method is to use generative machine learning to create believable decoys, a direction of research which to our knowledge has not been explored previously.

Our overarching goal was to demonstrate the potential and benefits of novel mostly deep learning methods for the automation of data analysis tasks in information security. For the task of clustering forensic logs to their signatures, we empirically showed that: it is possible to do so without supervision and that it is possible to integrate domain knowledge in the learning process, hence use such knowledge as weak supervision. Additionally, we can combine this domain knowledge with a few labels and therefore use these labels more efficiently. Furthermore, we empirically showed that it is possible to create believable decoy projects in a data-driven way, i.e., in an automated way. Such decoy projects could be used to gather contextual information in scenarios when such context is missing.

7.1 — Limitations of the proposed work

In this thesis, we proposed four novel methods that address three challenges for applying machine learning to information security tasks. We will first discuss the limitations of the proposed methods, and then we will discuss general limitations and other challenges in applying machine learning for information security.

The three proposed methods of the forensic use case in Chapter 3, 4 and 5 are based on deep learning techniques. Deep learning techniques, in turn, are based on certain assumptions and come with limitations that we have not addressed. One of these in this thesis unaddressed challenges is how to interpret DL models, the features learned and their results. Furthermore, DL models are limited by the quality of the data that is used for learning them. The quality of the data has to be ensured to obtain reliable results, e.g., by checking for biases in the data. Other, currently unaddressed challenges are the transfer models between domains, e.g., for different types of log files, and the integration of new data into existing models.

In Chapter 4, we propose an unsupervised method for log clustering. This method groups logs based on the signatures that produced them. The result of this method is clusters of logs, which could be used for labeling logs for further analysis. A currently unaddressed challenge is to extract the actual signature from these clusters.

In Chapter 5, we make the learning of a metric space more efficient concerning the number of labels used. A challenge that is currently not addressed is the selection of the triplets for the learning process. Triplets, i.e., an anchor, a positive and a negative example carry more information for the learning process the more they violate the triplet constraint (see Chapter 5.2.1). Hence, selecting the right triplets for the learning process to converge. As of now, we select the triplets randomly in an online fashion, which slows down the learning process.

In Chapter 6, we proposed a method for generating decoy projects in a data-driven way, and we provided empirical evidence that the generated projects could not be distinguished from real projects. To evaluate this method, we treat our judges as a classifier and evaluate the performance of such a classifier. This evaluation method provides insights about the expectation about further performance but does not provide any conclusions about the truth of the hypothesis whether the judges are capable of distinguishing between real and decoy projects. Furthermore, we conducted a confirmatory case study in a lab setting, which only provides insights on the external validity if a negative result would have been reported. That is, such a case study would have been useful to refute our hypothesis that the participants are incapable of distinguishing between real and decoy projects. Furthermore, we have identified a few open challenges to this method. First, we would need to ensure that decoys generated by our method fulfill to other desirable properties that decoys should have, for example having a considerable variability or being adaptable. Next, we should verify that decoy projects can indeed be used to detect intrusions in a real-life setting. Furthermore, we did not generate the names or the contents of the project decoys. As a consequence, inspecting the contents reveal clues about the

nature of these projects. However, since any interaction is already suspicious by definition, this limitation has little implications on the usage. Finally, we do not offer a direct solution to the challenge of lacking contextual information. Instead, we are proposing an alternative route to obtain such information. Machine learning with ambiguous patterns remains a challenging task, and we hypothesize that there are other ways of directly using machine learning for this challenge.

In this work, we addressed three challenges of using machine learning on information security tasks: the lack of supervision, the integration of domain knowledge and the lack of contextual information. While these three challenges tackle core problems of the application of machine learning on information security tasks, other challenges need to be addressed, which we did not address. Here we mention three of these challenges which we consider to be important. First, there exists a semantic gap between results and the interpretation of these results. That is, when a machine learning model predicts a result, it is difficult to understand for a human operator what that a result means and how to approach the underlying problem. Consider an intrusion detection system that only raises an alarm, but does not tell the operator about the context or the nature of this alarm. Then, the utility of such a system for an operator is fairly limited, because the operator lacks the knowledge to come up with a proper response to such an incident. Another significant challenge that needs to be addressed is the collaboration between machine learning models and human operators. Machine learning models should be able to incorporate feedback from operators, and they should be able to change parameters based on the desired context. Apart from that, the challenge of a lack of transparency in machine learning models also should be addressed. Legislation such as the General Data Protection Regulation (GDPR) in the EU requires that automated decisions by algorithms need to be justifiable. Hence, mechanisms that explain predictions of machine learning models and explain the inner workings of such models are required.

7.2 — Directions for Future Research

Signature extraction and its related problem log clustering have been addressed in a supervised way in Chapter 3 and in an unsupervised way in Chapter 4. Both problems are a necessary pre-processing step for further forensic analysis of the logs. Thus, potential research directions are identifying recurring patterns in the log files and ways that can benefit the forensic investigator. For example, one could research ways how to label recurring patterns in the log files to aid analysis of an ongoing investigation. A potentially fruitful direction is to combine our work with methods from process mining. For example, one could first cluster the forensic logs using our unsupervised method to obtain cluster ids which could be used as event ids. Then, in a second step one could use process discovery techniques [164] to obtain a process model, which could help a forensic investigator to analyze the processes on this system. Another potentially fruitful research direction is the exploration of combinations with other representation learning algorithms and other clustering

methods, which could potentially increase the performance of the method.

An important realization for the work presented in Chapter 5 is that ranking problems are much easier to solve than direct classification problems. For example, it is much easier to rank whether statement A is more insulting than statement B than to rate the level of insult on a scale from 1 to 10 for both statements. Furthermore, approximate knowledge in the form of heuristics is often much more straightforward to obtain than labeled data. These possibilities in combination open many different research paths, which may also be interesting for other domains where heuristics are available. For example, it could be possible to improve an approach for authentication by combining a fingerprint-similarity heuristic with labeled examples.

One way to directly improve the method is research on selecting the triplets that are used for learning. Selecting the triplets that carry the most different information seems to be essential to deep metric learning with triplet networks. Such methods could utilize locality sensitive hashing and select triplets that carry more information based on their distance. Another way to select triplets is to draw from methods of the domain of active learning.

Metric spaces offer another potential advantage to other forms of representation, namely that the location and distances between items in that space are associated with a meaning. Potential research directions are verifying these claims, and if this holds for deep metric learning, devise methods for labeling and interpreting such metric spaces. This research direction would touch the previously outlined challenge of semantic gap.

Finally, the idea to treat the learning problems as ranking problems allows potentially for the use of heuristics in problems of other domains and other data types. For example, one could imagine a face detection method that combines existing face similarity heuristics with labeled information.

In the chapter on generating decoy projects, we lay the foundations of a model-based decoy generation approach. We evaluated the believability of the generated decoys. There are multiple ways to continue research on this topic. To improve the approach, one could investigate the generation of content for the generated files. Additionally, the applicability of decoys to detect intrusions or data-theft in a realistic setting needs to be validated. Another, potentially fruitful research direction could be the investigation on ways for extracting human-interpretable and adjustable rules from the trained models. Also, many generative deep learning models have been proposed recently. Examples of such generative models are Generative Adversarial Networks [48] or Variational Autoencoders [75]. We hypothesize that such generative deep learning methods offer many possibilities for generating different types of decoy objects.

Bibliography

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, and Others. TensorFlow: A System for Large-Scale Machine Learning. In *OSDI*, volume 16, pages 265–283, 2016.

[2] Martín Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep Learning with Differential Privacy. In Shai Halevi, editor, *Proceedings of the ACM Conference on Computer and Communications Security*, volume 24-28-Octo, pages 308–318. ACM, 2016.

[3] ACM. ACM Computing Classification System, 2012.

[4] Michal Aharon, Gilad Barash, Ira Cohen, and Eli Mordechai. One graph is worth a thousand logs: Uncovering hidden structures in massive system event logs. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 5781 LNAI, pages 227–243. Springer, 2009.

[5] Abdalnaser Algwil, Dan Ciresan, Beibei Liu, and Jeff Yan. A security analysis of automated chinese turing tests. In *Proceedings of the 32nd Annual Conference on Computer Security Applications - ACSAC '16*, volume 5-9-Decemb, pages 520–532. Association for Computing Machinery, 2016.

[6] Bander Alsulami, Edwin Dauber, Richard Harang, Spiros Mancoridis, and Rachel Greenstadt. *Source Code Authorship Attribution Using Long Short-Term Memory Based Networks*, volume 10492 LNCS. 2017.

[7] Muhamad Erza Aminanto, Rakyong Choi, Harry Chandra Tanuwidjaja, Paul D. Yoo, and Kwangjo Kim. Deep Abstraction and Weighted Feature Selection for Wi-Fi Impersonation Detection. *IEEE Transactions on Information Forensics and Security*, 13(3):621–636, 2018.

[8] Mozhgan Azimpourkivi, Umut Topkara, and Bogdan Carbunar. A secure mobile authentication alternative to biometrics. In *ACM International Conference Proceeding Series*, volume Part F1325, pages 28–41, 2017.

[9] William C Barker. Guidelines for Identifying an Information System as a National Security System. Technical report, 2003.

[10] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[11] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[12] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[13] Samarth Bharadwaj, Himanshu S. Bhatt, Mayank Vatsa, and Richa Singh. Domain Specific Learning for Newborn Face Recognition. *IEEE Transactions on Information Forensics and Security*, 11(7):1630–1641, 2016.

[14] Aparna Bharati, Richa Singh, Mayank Vatsa, and K.W. Kevin W. Bowyer. Detecting Facial Retouching Using Supervised Deep Learning. *IEEE Transactions on Information Forensics and Security*, 11(9):1903–1913, 2016.

[15] Kevin Borgolte, Christopher Kruegel, Giovanni Vigna, Kevin Borgolte, Christopher Kruegel, and Giovanni Vigna. Meerkat : Detecting Website Defacements through Image-based Object Recognition. In *USENIX Security*, pages 595–610, 2015.

[16] Brian M B M Bowen, Shlomo Hershkop, A D Angelos D Keromytis, and S J Salvatore J Stolfo. *Baiting inside attackers using decoy documents*. Springer, 2009.

[17] Matthew L Bringer, Christopher A Chelmecki, and Hiroshi Fujinoki. A survey: Recent advances and future trends in honeypot research. *International Journal*, 4, 2012.

[18] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a" siamese" time delay neural network. In *Advances in Neural Information Processing Systems*, pages 737–744, 1994.

[19] Kai Cao and Anil K. Jain. Latent orientation field estimation via convolutional neural network. In *Proceedings of 2015 International Conference on Biometrics, ICB 2015*, pages 349–356, 2015.

[20] Nicholas Carlevaris-Bianco and Ryan M Eustice. Learning visual feature descriptors for dynamic lighting conditions. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 2769–2776. IEEE, 2014.

[21] Gert Cauwenberghs. A fast stochastic error-descent algorithm for supervised learning and optimization. In *Advances in neural information processing systems*, pages 244–251, 1993.

[22] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the Devil in the Details: Delving Deep into Convolutional Nets. *arXiv preprint arXiv: …*, pages 1–11, 2014.

[23] Z Chen, B Yu, Y Zhang, J Zhang, and J Xu. Automatic mobile application traffic identification by convolutional neural networks. In *Proceedings - 15th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, 10th IEEE International Conference on Big Data Science and Engineering and 14th IEEE International Symposium on Parallel and Distributed Proce*, pages 301–307, 2016.

[24] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, 2014.

[25] S Chopra, R Hadsell, and Y LeCun. Learning a similarity metric discriminatively, with application to face verification. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 539–546, 2005.

[26] Zheng Leong Chua, Shiqi Shen, Prateek Saxena, and Zhenkai Liang. Neural Nets Can Learn Function Type Signatures From Binaries. In *26th USENIX Security Symposium (USENIX Security 17)*, volume 17, pages 99–116, 2017.

[27] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[28] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Gated feedback recurrent neural networks. In *International Conference on Machine Learning*, pages 2067–2075, 2015.

[29] Dan Cireşan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *Computer vision and pattern recognition (CVPR), 2012 IEEE conference on*, pages 3642–3649. IEEE, 2012.

[30] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[31] Adam Czajka, Kevin W. Bowyer, Michael Krumdick, and Rosaura G. Vidalmata. Recognition of Image-Orientation-Based Iris Spoofing. *IEEE Transactions on Information Forensics and Security*, 12(9):2184–2196, 2017.

[32] Eduardo Jose Da Silva Luz, Gladston J.P. P Moreira, Luiz S. Oliveira, William Robson Schwartz, and David Menotti. Learning Deep Off-the-Person Heart Biometrics Representations. *IEEE Transactions on Information Forensics and Security*, 13(5):1258–1270, 2018.

[33] Andrew M Dai and Quoc V Le. Semi-supervised sequence learning. In *Advances in neural information processing systems*, pages 3079–3087, 2015.

[34] Hanjun Dai, Bo Dai, and Le Song. Discriminative embeddings of latent variable models for structured data. In *International Conference on Machine Learning*, pages 2702–2711, 2016.

[35] Reinhard Diestel. Graph theory. 2005. *Grad. Texts in Math*, 101, 2005.

[36] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. DeepLog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 1285–1298, 2017.

[37] Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A Smith. Transition-based dependency parsing with stack long short-term memory. *arXiv preprint arXiv:1505.08075*, 2015.

[38] Dzmitry Bahdana, Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation By Jointly Learning To Align and Translate. *Iclr 2015*, pages 1–15, 2014.

[39] Anselmo Ferreira, Luca Bondi, Luca Baroffio, Paolo Bestagini, Jiwu Huang, Jefersson A. Dos Santos, Stefano Tubaro, and Anderson Rocha. Data-Driven Feature Characterization Techniques for Laser Printer Attribution. *IEEE Transactions on Information Forensics and Security*, 12(8):1860–1873, 2017.

[40] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, NY, USA:, 2001.

[41] Qiang Fu, Jian-guang Lou, Yi Wang, and Jiang Li. Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis. In *ICDM*, volume 9, pages 149–158, 2009.

[42] Christian Galea and R.A. Reuben A. Farrugia. Matching Software-Generated Sketches to Face Photographs with a Very Deep CNN, Morphed Faces, and Transfer Learning. *IEEE Transactions on Information Forensics and Security*, 13(6):1421–1431, 2018.

[43] Haichang Gao, Wei Wang, Jiao Qi, Xuqin Wang, Xiyang Liu, and Jeff Yan. The robustness of hollow CAPTCHAs. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security - CCS '13*, pages 1075–1086, 2013.

[44] Shenghua Gao, Yuting Yingying Zhang, Kui Jia, Jiwen Lu, and Yuting Yingying Zhang. Single Sample Face Recognition via Learning Deep Supervised Autoencoders. *IEEE Transactions on Information Forensics and Security*, 10(10):2108–2118, 2015.

[45] Stuart Geman, Renc Doursat, and Elie Bienenstock. Neural Networks and the Bias/Variance Dilemma. *Neural Computation*, 4(1):1–58, 1992.

[46] Ester Gonzalez-Sosa, Julian Fierrez, Ruben Vera-Rodriguez, and Fernando Alonso-Fernandez. Facial soft biometrics for recognition in the wild: Recent works, annotation, and COTS evaluation. *IEEE Transactions on Information Forensics and Security*, 13(8):2001–2014, 2018.

[47] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[48] 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.

[49] Gaurav Goswami, Mayank Vatsa, and Richa Singh. Face Verification via Learned Representation on Feature-Rich Video Frames. *IEEE Transactions on Information Forensics and Security*, 12(7):1686–1698, 2017.

[50] Alan Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6645–6649. IEEE, 2013.

[51] Alex Graves. Generating Sequences with Recurrent Neural Networks. *Technical Reports*, pages 1–43, 2013.

[52] Alex Graves, Navdeep Jaitly, and Abdel-rahman Mohamed. Hybrid speech recognition with deep bidirectional LSTM. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, pages 273–278. IEEE, 2013.

[53] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2017.

[54] Y. Yi Han, Sriharsha Etigowni, H. Liu, Saman Zonouz, Athina Petropulu, Hua Li, Saman Zonouz, and Athina Petropulu. Watch me, but don't touch me! contactless control flow monitoring via electromagnetic emanations. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 1095–1108, 2017.

[55] E.M. M Hand and R. Chellappa. Attributes for improved attributes: A multi-task network utilizing implicit and explicit relationships for facial attribute classification. In *31st AAAI Conference on Artificial Intelligence, AAAI 2017*, pages 4068–4074, 2017.

[56] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *european conference on computer vision*, pages 346–361. Springer, 2014.

[57] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[58] Pinjia He, Jieming Zhu, Shilin He, Jian Li, and Michael R. Lyu. An evaluation study on log parsing and its use in log mining. *Proceedings - 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2016*, pages 654–661, 2016.

[59] Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Others. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6):82–97, 2012.

[60] Geoffrey E Hinton. Deep belief networks. *Scholarpedia*, 4(5):5947, 2009.

[61] Briland Hitaj, Giuseppe Ateniese, and Fernando Perez-Cruz. Deep Models under the GAN: Information leakage from collaborative deep learning. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 603–618, 2017.

[62] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

[63] Elad Hoffer and Nir Ailon. Deep metric learning using triplet network. In *International Workshop on Similarity-Based Pattern Recognition*, pages 84–92. Springer, 2015.

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

[65] Yuchi Huang, Qingshan Liu, Shaoting Zhang, and Dimitris N Metaxas. Image retrieval via probabilistic hypergraph ranking. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3376–3383. IEEE, 2010.

[66] IEEE. IEEE Taxonomy 2017, 2017.

[67] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. feb 2015.

[68] 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.

[69] Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. On Using Very Large Target Vocabulary for Neural Machine Translation. dec 2014.

[70] Melvin Johnson, Mike Schuster, Quoc V Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, and Others. Google's Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation. *arXiv preprint arXiv:1611.04558*, 2016.

[71] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *International Conference on Machine Learning*, pages 2342–2350, 2015.

[72] Haribabu Kandi, Deepak Mishra, and S.R.K.S. Subrahmanyam R.K.Sai Gorthi. Exploring the learning capabilities of convolutional neural networks for robust image watermarking. *Computers and Security*, 65:247–268, 2017.

[73] John G Kemeny and James Laurie Snell. *Finite markov chains*, volume 356. van Nostrand Princeton, NJ, 1960.

[74] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. dec 2014.

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

[76] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML Deep Learning Workshop*, volume 2, 2015.

[77] Alex Krizhevsky. cuda-convnet: High-performance c++/cuda implementation of convolutional neural networks. *Source code available at https://github.com/akrizhevsky/cuda-convnet2 [March, 2017]*, 2012.

[78] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[79] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*, 2016.

[80] Tilman Lange, Volker Roth, Mikio L Braun, and Joachim M Buhmann. Stability-based validation of clustering solutions. *Neural computation*, 16(6):1299–1323, 2004.

[81] Aleksandar Lazarevic, Levent Ertoz, Vipin Kumar, Aysel Ozgur, and Jaideep Srivastava. A comparative study of anomaly detection schemes in network intrusion detection. In *Proceedings of the 2003 SIAM International Conference on Data Mining*, pages 25–36. SIAM, 2003.

[82] James Lewis. Economic Impact of Cybercrime— No Slowing Down. Technical report, McAffee.

[83] Yanxiong Li, Xue Zhang, Xianku Li, Yuhan Zhang, Jichen Yang, and Qianhua He. Mobile Phone Clustering from Speech Recordings Using Deep Representation and Spectral Clustering. *IEEE Transactions on Information Forensics and Security*, 13(4):965–977, 2018.

[84] Z Lin, Y Huang, and J Wang. RNN-SM: Fast Steganalysis of VoIP Streams Using Recurrent Neural Network. *IEEE Transactions on Information Forensics and Security*, 13(7):1854–1868, 2018.

[85] Nianfeng Liu, Haiqing Li, Man Zhang, Jing Liu, Zhenan Sun, and Tieniu Tan. Accurate iris segmentation in non-cooperative environments using fully convolutional networks. In *2016 International Conference on Biometrics, ICB 2016*, 2016.

[86] Xiaoxiang Liu, Lingxiao Song, Xiang Wu, and Tieniu Tan. Transferring deep representation for NIR-VIS heterogeneous face recognition. In *2016 International Conference on Biometrics, ICB 2016*, 2016.

[87] Da Luo, Rui Yang, Bin Li, and Jiwu Huang. Detection of double compressed AMR audio using stacked autoencoder. *IEEE Transactions on Information Forensics and Security*, 12(2):432–444, 2017.

[88] David Maier. The complexity of some problems on subsequences and supersequences. *Journal of the ACM (JACM)*, 25(2):322–336, 1978.

[89] Adetokunbo Makanju, A Nur Zincir-Heywood, and Evangelos E Milios. A Lightweight Algorithm for Message Type Extraction in System Application Logs. *IEEE Transactions on Knowledge and Data Engineering*, 24(11):1921–1936, 2012.

[90] Adetokunbo A. O. Makanju, A. Nur Zincir-Heywood, and Evangelos E. Milios. Clustering event logs using iterative partitioning. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '09*, page 1255. ACM, 2009.

[91] Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *International Conference on Artificial Neural Networks*, pages 52–59. Springer, 2011.

[92] Sunu Mathew, Michalis Petropoulos, Hung Q Ngo, and Shambhu Upadhyaya. A data-centric approach to insider attack detection in database systems. In *International Workshop on Recent Advances in Intrusion Detection*, pages 382–401. Springer, 2010.

[93] Felix Mayer and Martin Steinebach. Forensic image inspection assisted by deep learning. In *ACM International Conference Proceeding Series*, volume Part F1305, pages 1–9, 2017.

[94] William Melicher, Blase Ur, Sean M Segreti, Saranga Komanduri, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. Fast, Lean, and Accurate: Modeling Password Guessability Using Neural Networks. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 175–191, 2016.

[95] Gábor Melis, Chris Dyer, and Phil Blunsom. On the state of the art of evaluation in neural language models. *arXiv preprint arXiv:1707.05589*, 2017.

[96] Dongyu Meng and Hao Chen. MagNet: a Two-Pronged Defense against Adversarial Examples. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 135–147, 2017.

[97] David Menotti, Giovani Chiachia, Allan Pinto, William Robson Schwartz, Helio Pedrini, Alexandre Xavier Falcão, Anderson Rocha, Alexandre Xavier Falcão, and Anderson Rocha. Deep Representations for Iris, Face, and Fingerprint Spoofing Detection. *IEEE Transactions on Information Forensics and Security*, 10(4):864–879, 2015.

[98] Tomas Mikolov, K Chen, G Corrado, and J Dean. Efficient Estimation of Word Representations in Vector Space. *ArXiv e-prints*, 2013.

[99] Tomas Mikolov, Kai Chen, Greg S Corrado, Jeffrey Dean, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeffrey Dean. Distributed Representations of

Words and Phrases and their Compositionality. In C J C Burges, L Bottou, M Welling, Z Ghahramani, and K Q Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013.

[100] Tom M Mitchell et al. Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45(37):870–877, 1997.

[101] Paritosh Mittal, Mayank Vatsa, and Richa Singh. Composite sketch recognition via deep network - A transfer learning approach. In *Proceedings of 2015 International Conference on Biometrics, ICB 2015*, pages 251–256, 2015.

[102] Daisuke Miyamoto, Hiroaki Hazeyama, and Youki Kadobayashi. An evaluation of machine learning-based methods for detection of phishing sites. In *Advances in Neuro-Information Processing*, pages 539–546. Springer, 2008.

[103] Jonas Mueller and Aditya Thyagarajan. Siamese Recurrent Architectures for Learning Sentence Similarity. In *AAAI*, pages 2786–2792, 2016.

[104] Neeru Narang and Thirimachos Bourlai. Gender and ethnicity classification using deep learning in heterogeneous face recognition. In *2016 International Conference on Biometrics (ICB)*, pages 1–8, 2016.

[105] Paul Neculoiu, Maarten Versteegh, Mihai Rotaru, and Textkernel B V Amsterdam. Learning Text Similarity with Siamese Recurrent Networks. *ACL 2016*, page 148, 2016.

[106] Minh Hai Nguyen, Dung Le Nguyen, Xuan Mao Nguyen, and Tho Thanh Quan. Auto-detection of sophisticated malware using lazy-binding control flow graph and deep learning. *Computers and Security*, 76:128–155, 2018.

[107] Rodrigo Frassetto Nogueira, Roberto De Alencar Lotufo, Rubens Campos MacHado, Roberto Lotufo, and Rubens Campos MacHado. Fingerprint Liveness Detection using Convolutional Neural Networks. *IEEE Transactions on Information Forensics and Security*, 6013(c):1–1, 2016.

[108] Vahid Noroozi, Lei Zheng, Sara Bahaadini, Sihong Xie, and Philip S. Yu. SEVEN: Deep SEmi-supervised verification networks. In *IJCAI International Joint Conference on Artificial Intelligence*, pages 2571–2577, 2017.

[109] Adam J Oliner and Jon Stearley. What Supercomputers Say : A Study of Five System Logs. In *Dsn*, pages 575–584. IEEE, 2007.

[110] Genki Osada, Kazumasa Omote, and Takashi Nishide. *Network Intrusion Detection Based on Semi-supervised Variational Auto-Encoder*, volume 10493 LNCS. 2017.

[111] Margarita Osadchy, Julio Hernandez-Castro, Stuart Gibson, Orr Dunkelman, and Daniel Perez-Cabo. No Bot Expects the DeepCAPTCHA! Introducing Immutable Adversarial Examples, with Applications to CAPTCHA Generation. *IEEE Transactions on Information Forensics and Security*, 12(11):2640–2653, 2017.

[112] Oxford. Definition of Markov Chain in English.

[113] Nicolas Papernot, Patrick Mcdaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *Proceedings - 2016 IEEE European Symposium on Security and Privacy, EURO S and P 2016*, pages 372–387, 2016.

[114] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks. In *Proceedings - 2016 IEEE Symposium on Security and Privacy, SP 2016*, pages 582–597, 2016.

[115] Nicolas Papernot, Patrick D McDaniel, Ian J Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical Black-Box Attacks against Machine Learning. In Xun Yi, editor, *ASIA CCS 2017 - Proceedings of the 2017 ACM Asia Conference on Computer and Communications Security*, pages 506–519. ACM, 2016.

[116] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. *ICML (3)*, 28:1310–1318, 2013.

[117] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global Vectors for Word Representation. In *EMNLP*, volume 14, pages 1532–1543, 2014.

[118] N.a Phan, Y.b Wang, X.c Wu, and D.a Dou. Differential privacy preservation for deep auto-encoders: An application of human behavior prediction. In *30th AAAI Conference on Artificial Intelligence, AAAI 2016*, number Arfken 1985, pages 1309–1316, 2016.

[119] Le Trieu Phong, Yoshinori Aono, Takuya Hayashi, Lihua Wang, and Shiho Moriai. Privacy-Preserving Deep Learning via Additively Homomorphic Encryption. *IEEE Transactions on Information Forensics and Security*, 13(5):1333–1345, 2018.

[120] Huafeng b Qin and Mounim A. M.A.a El-Yacoubi. Deep Representation-Based Feature Extraction and Recovering for Finger-Vein Verification. *IEEE Transactions on Information Forensics and Security*, 12(8):1816–1829, 2017.

[121] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[122] Oren Rippel, Manohar Paluri, Piotr Dollar, and Lubomir Bourdev. Metric learning with adaptive density discrimination. *arXiv preprint arXiv:1511.05939*, 2015.

[123] Andrew Rosenberg and Julia Hirschberg. V-Measure: A Conditional Entropy-Based External Cluster Evaluation Measure. In *EMNLP-CoNLL*, volume 7, pages 410–420, 2007.

[124] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[125] Neil C Rowe. Automatic detection of fake file systems. 2005.

[126] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, and Others. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[127] Malek Ben Salem, Shlomo Hershkop, and Salvatore J Stolfo. A survey of insider attack detection research. In *Insider Attack and Cyber Security*, pages 69–90. Springer, 2008.

[128] Jürgen Schmidhuber. Deep Learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[129] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 815–823, 2015.

[130] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[131] Roei Schuster, Vitaly Shmatikov, and Eran Tromer. Beauty and the Burst: Remote Identification of Encrypted Video Streams. In *USENIX Security*, number 2, pages 1–26, 2017.

[132] Ashlesh Sharma, Vidyuth Srinivasan, Vishal Kanchan, and Lakshminarayanan Subramanian. The Fake vs Real Goods Problem. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '17*, volume Part F1296, pages 2011–2019, 2017.

[133] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[134] Shiqi Shen, Shruti Tople, and Prateek Saxena. A uror: defending against poisoning attacks in collaborative deep learning systems. In Davide Balzarotti, editor, *Proceedings of the 32nd Annual Conference on Computer Security Applications - ACSAC '16*, volume 5-9-Decemb, pages 508–519. ACM, 2016.

[135] T. Shibahara, K. Yamanishi, Y. Takata, D. Chiba, M. Akiyama, T. Yagi, Y. Ohsita, and M. Murata. Malicious URL sequence detection using event de-noising convolutional neural network. In *IEEE International Conference on Communications*, pages 1–7, 2017.

[136] Eui Chul Richard Shin, Dawn Song, and Reza Moazzezi. Recognizing Functions in Binaries with Neural Networks. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 611–626, 2015.

[137] K. Shiraga, Y. Makihara, D. Muramatsu, T. Echigo, and Y. Yagi. GEINet: View-invariant gait recognition using a convolutional neural network. In *2016 International Conference on Biometrics, ICB 2016*, pages 0–7, 2016.

[138] Vitaly Shokri, Reza and Shmatikov. Privacy-Preserving Deep Learning. In *Proceedings of the ACM Conference on Computer and Communications Security*, volume 2015-Octob, pages 1310–1321, 2015.

[139] Patrice Y Simard, David Steinkraus, John C Platt, and Others. Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR*, volume 3, pages 958–962, 2003.

[140] K Simonyan and A Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *ArXiv e-prints*, 2014.

[141] Suphannee Sivakorn, Iasonas Polakis, and Angelos D. Keromytis. I am Robot: (Deep) learning to break semantic image CAPTCHAs. In *Proceedings - 2016 IEEE European Symposium on Security and Privacy, EURO S and P 2016*, pages 388–403, 2016.

[142] Richard Socher, Milind Ganjoo, Christopher D Manning, and Andrew Ng. Zero-shot learning through cross-modal transfer. In *Advances in neural information processing systems*, pages 935–943, 2013.

[143] Lance Spitzner. Honeypots: Catching the insider threat. In *Computer Security Applications Conference, 2003. Proceedings. 19th Annual*, pages 170–179. IEEE, 2003.

[144] 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.

[145] Cliord P Stoll. The cuckoo's egg: Tracing a spy through the maze of computer espionage, 1989.

[146] Catherine A Sugar and Gareth M James. Finding the number of clusters in a dataset: An information-theoretic approach. *Journal of the American Statistical Association*, 98(463):750–763, 2003.

[147] Lichao Sun, Yuqi Wang, Bokai Cao, P.S. Philip S. Yu, Witawas Srisa-An, and A.D. Alex D. Leow. *Sequential Keystroke Behavioral Biometrics for Mobile User Identification via Multi-view Deep Learning*, volume 10536 LNAI. 2017.

[148] Yi Sun, Xiaogang Wang, and Xiaoou Tang. Deep Learning Face Representation by Joint Identification-Verification. In *Advances in Neural Information Processing Systems*, volume 3, pages 1988–1996, 2014.

[149] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[150] Christian Szegedy, W Zaremba, and I Sutskever. Intriguing properties of neural networks. *arXiv preprint arXiv: ...*, pages 1–10, 2013.

[151] Liang Tang, Tao Li, and Chang-shing Perng. LogSig : Generating System Events from Raw Textual Logs. In *Cikm*, pages 785–794. ACM, 2011.

[152] Stefan Thaler, Jerry den Hartog, and Milan Petkovic. Towards Creating Believable Decoy Project Folders for Detecting Data Theft. In *Data and Applications Security and Privacy {XXX} - 30th Annual {IFIP} {WG} 11.3 Conference, DBSec 2016, Trento, Italy, July 18-20, 2016. Proceedings*, pages 161–169, 2016.

[153] Stefan Thaler, Vlado Menkovski, and Milan Petković. Towards a neural language model for signature extraction from forensic logs. In *2017 5th International Symposium on Digital Forensic and Security (ISDFS)*, pages 1—-6. IEEE, 2017.

[154] Stefan Thaler, Vlado Menkovski, and Milan Petkovic. Towards unsupervised signature extraction of forensic logs. In *Benelearn 2017: Proceedings of the Twenty-Sixth Benelux Conference on Machine Learning, Technische Universiteit Eindhoven, 9-10 June 2017*, page 154, 2017.

[155] Stefan Thaler, Vlado Menkovski, and Milan Petković. Unsupervised Signature Extraction from Forensic Logs. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2017, Skopje, Macedonia, September 18-22, 2017, Proceedings*. Springer, 2017.

[156] Stefan Thaler, Vlado Menkovski, and Milan Petković. Deep metric learning for sequential data using approximate information. In *Machine Learning and Data Mining in Pattern Recognition*. Springer, 2018.

[157] Stefan Thaler, Vlado Menkovski, Milan Petković, and Milan Petkovic. *Unsupervised Signature Extraction from Forensic Logs*, volume 10536 LNAI. Springer, 2017.

[158] Robert Tibshirani, Guenther Walther, and Trevor Hastie. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(2):411–423, 2001.

[159] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4(2), 2012.

[160] Lam L.a Tran, D.b Deguang Kong, H.b Hongxia Jin, and J.a Ji Liu. Privacy-CNH: A Framework to Detect Photo Privacy with Convolutional Neural Network using Hierarchical Features. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pages 1–7, 2016.

[161] Lior Uzan and Lior Wolf. I know that voice: Identifying the voice actor behind the voice. In *Proceedings of 2015 International Conference on Biometrics, ICB 2015*, pages 46–51, 2015.

[162] Risto Vaarandi. A Data Clustering Algorithm for Mining Patterns From Event Logs. In *Computer Engineering*, pages 119–126, 2003.

[163] Risto Vaarandi and Mauno Pihelgas. LogCluster - A Data Clustering and Pattern Mining Algorithm for Event Logs. In *12th International Conference on Network and Service Management - CNSM 2015*, pages 1–8. IEEE Computer Society, 2015.

[164] Wil Van der Aalst, Ton Weijters, and Laura Maruster. Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge & Data Engineering*, (9):1128–1142, 2004.

[165] Ashish Vaswani, Yonatan Bisk, Kenji Sagae, and Ryan Musa. Supertagging with LSTMs. In *Proceedings of the Human Language Technology Conference of the NAACL*, 2016.

[166] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning - ICML '08*, pages 1096–1103. ACM, 2008.

[167] Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *Journal of Machine Learning Research*, 11(Oct):2837–2854, 2010.

[168] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3156–3164, 2015.

[169] Luis Von Ahn, Manuel Blum, Nicholas J Hopper, and John Langford. CAPTCHA: Using hard AI problems for security. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 294–311. Springer, 2003.

[170] Jonathan Voris, Nathaniel Boggs, and Salvatore J Stolfo. Lost in translation: Improving decoy documents via automated translation. In *Security and Privacy Workshops (SPW), 2012 IEEE Symposium on*, pages 129–133. IEEE, 2012.

[171] Jonathan A Voris, Jill Jermyn, Angelos D Keromytis, and Salvatore J Stolfo. Bait and snitch: Defending computer systems with decoys. 2013.

[172] Dayong Wang and Anil K. Jain. Face retriever: Pre-filtering the gallery via deep neural net. In *Proceedings of 2015 International Conference on Biometrics, ICB 2015*, pages 473–480, 2015.

[173] Jingbin Jiang Wang, Yang Song, Thomas Leung, Chuck Rosenberg, Jingbin Jiang Wang, James Philbin, Bo Chen, and Ying Wu. Learning fine-grained image similarity with deep ranking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1386–1393, 2014.

[174] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[175] Jason Weston and Michael Karlen. Natural Language Processing (Almost) from Scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.

[176] Ben Whitham. Automating the generation of fake documents to detect network intruders. *International Journal of Cyber-Security and Digital Forensics (IJCSDF)*, 2(1):103–118, 2013.

[177] Ben Whitham. Canary Files: Generating Fake Files to Detect Critical Data Loss from Complex Computer Networks. In *The Second International Conference on Cyber Security, Cyber Peacefare and Digital Forensic (CyberSec2013)*, pages 170–179. The Society of Digital Information and Wireless Communication, 2013.

[178] Ben Whitham. Design requirements for generating deceptive content to protect document repositories. 2014.

[179] Junyuan Xie, Ross Girshick, and Ali Farhadi. Unsupervised Deep Embedding for Clustering Analysis. *arXiv preprint arXiv:1511.06335*, 2015.

[180] Eric P Xing, Michael I Jordan, Stuart J Russell, and Andrew Y Ng. Distance metric learning with application to clustering with side-information. In *Advances in neural information processing systems*, pages 521–528, 2003.

[181] Zhengzheng Xing, Jian Pei, and Eamonn Keogh. A brief survey on sequence classification. *ACM Sigkdd Explorations Newsletter*, 12(1):40–48, 2010.

[182] Guanshuo Xu, Han-zhou Wu, Student Member, and Yun-qing Shi. Structural Design of Convolutional Neural Networks for Steganalysis. 23(5):708–712, 2016.

[183] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057, 2015.

[184] Wei Xu, Ling Huang, Armando Fox, David Patterson, Michael I Jordan, Ling Huang, Armando Fox, David Patterson, and Michael I Jordan. Detecting Large-Scale System Problems by Mining Console Logs. In *22nd ACM Symposium on Operating Systems Principles*, pages 117–131. ACM, 2009.

[185] Xiaojun Xu, Chang Liu, Qian Feng, Heng Yin, Le Song, and Dawn Song. Neural network-based graph embedding for cross-platform binary code similarity detection. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 363–376, 2017.

[186] Yuanshun Yao, Bimal Viswanath, Jenna Cryan, Haitao Zheng, and B.Y. Y Ben Y. Zhao. Automated crowdturfing attacks and defenses in online review systems. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 1143–1158, 2017.

[187] Risa Yashiro, Takanori Machida, Mitsugu Iwamoto, and Kazuo Sakiyama. *Deep-learning-based security evaluation on authentication systems using arbiter PUF and its variants*, volume 9836 LNCS. 2016.

[188] Jian Ye, Jiangqun Ni, and Yang Yi. Deep Learning Hierarchical Representations for Image Steganalysis. *IEEE Transactions on Information Forensics and Security*, 12(11):2545–2557, 2017.

[189] Quanzeng You, Jiebo Luo, Hailin Jin, and Jianchao Yang. Robust Image Sentiment Analysis Using Progressively Trained and Domain Transferred Deep Networks. In *AAAI*, pages 381–388, 2015.

[190] Jun Yu, Baopeng Zhang, Zhengzhong Kuang, Dan Lin, Jianping Fan 0001, and Jianping Fan. iPrivacy: Image Privacy Protection by Identifying Sensitive Objects via Deep Multi-Task Learning. *IEEE Trans. Information Forensics and Security*, 12(5):1005–1016, 2017.

[191] Jim Yuill, Mike Zappe, Dorothy Denning, and Fred Feer. Honeyfiles: deceptive files for intrusion detection. In *Information Assurance Workshop, 2004. Proceedings from the Fifth Annual IEEE SMC*, pages 116–122. IEEE, 2004.

[192] Sergey Zagoruyko and Nikos Komodakis. Learning to compare image patches via convolutional neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pages 4353–4361. IEEE, 2015.

[193] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

[194] Matthew D Zeiler, Graham W Taylor, and Rob Fergus. Adaptive deconvolutional networks for mid and high level feature learning. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2018–2025. IEEE, 2011.

[195] Jishen Zeng, Shunquan Tan, Bin Li, and Jiwu Huang. Large-Scale JPEG Image Steganalysis Using Hybrid Deep-Learning Framework. *IEEE Transactions on Information Forensics and Security*, 13(5):1200–1214, 2018.

[196] Jun Zhang, Zhenjie Zhang, Xiaokui Xiao, Yin Yang, and Marianne Winslett. Functional mechanism: regression analysis under differential privacy. *Proceedings of the VLDB Endowment*, 5(11):1364–1375, 2012.

[197] Qi Zhang, Haiqing Li, Zhenan Sun, Zhaofeng He, and Tieniu Tan. Exploring complementary features for iris recognition on mobile devices. In *2016 International Conference on Biometrics, ICB 2016*, 2016.

[198] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: an efficient data clustering method for very large databases. In *ACM Sigmod Record*, volume 25, pages 103–114. ACM, 1996.

[199] Zijing Zhao and Ajay Kumar. Accurate Periocular Recognition under Less Constrained Environment Using Semantics-Assisted Convolutional Neural Network. *IEEE Transactions on Information Forensics and Security*, 12(5):1017–1030, 2017.

[200] Haoti Zhong, Hao Li, Anna Squicciarini, Sarah Rajtmajer, Christopher Griffin, David Miller, and Cornelia Caragea. Content-driven detection of cyberbullying on the instagram social network. In *IJCAI International Joint Conference on Artificial Intelligence*, volume 2016-Janua, pages 3952–3958, 2016.

[201] Yang Zhong, Josephine Sullivan, and Haibo Li. Face attribute prediction using off-the-shelf CNN features. In *2016 International Conference on Biometrics, ICB 2016*, 2016.

[202] Zhi-Hua Zhou. A brief introduction to weakly supervised learning. *National Science Review*, 2017.

[203] Jianqing Zhu, Shengcai Liao, Dong Yi, Zhen Lei, and Stan Z. Li. Multi-label CNN based pedestrian attribute learning for soft biometrics. In *Proceedings of 2015 International Conference on Biometrics, ICB 2015*, pages 535–540, 2015.

Appendix A

Summary

Automation for Information Security using Machine Learning

Information security addresses the protection of information and information systems from unauthorized access, use, disclosure, disruption, modification, or destruction in order to provide confidentiality, integrity, and availability. One crucial part of information security is to collect data from these information systems and analyze them to understand which actions have been conducted on this information system and whether they were malicious or not.

Such data analysis in information security comes in many forms. The most basic one involves the manual labor of a domain expert. This expert sifts through the data for intriguing clues on the task that she is trying to complete. In many cases, this process of manually analyzing data is cumbersome, labor-intensive and error-prone. Relying on manual labor is particularly a problem when the amount of data that needs to be analyzed is large.

To address these data analysis challenges, security professionals often use software tools to aid them. Such tools automate repetitive data analysis tasks by formulating rules on the data, which extract and aggregate valuable information from large piles of data. As a result, such tools drastically reduce the efforts on data analysis. However, larger, more complex problems require a large number of rules. The set of rules may be incomplete, contradict each other or may not generalize well.

Machine learning is a sub-field of artificial intelligence that aims to learn rules from data to solve a task, rather than letting an expert define them manually. However, applying machine learning solutions to security problems is not straightforward, and a number of challenges need to be addressed. In this thesis, we focus on three main challenges, the lack of supervision, the usage of additional domain knowledge and the lack of contextual information. If used in a supervised way, machine learning requires labels, but in information security, such labels may not be readily available, difficult to obtain, or may become obsolete because the context changes fast. In many cases, in information security additional domain knowledge is available, but there are no standard mechanisms to integrate this knowledge into the learning process, which results in less efficient learning. Finally, in some information security scenarios, the intent of an action matters, but it cannot be simply distinguished by merely analyzing

the logs. For example, copying a project from a repository can be a benign or a malicious action - benign if the intent is to create a backup, malicious if the intent is to steal this project.

In this thesis, we work on two use cases that are representative of the three challenges that we want to address. The first use case deals with the information forensics problem of signature extraction from forensic logs. Forensic logs are semi-structured, high dimensional sequences which are produced by the software processes on the investigated computer. These logs are typically large and labeled data is not typically available since the processes that create these logs change frequently. Furthermore, additional domain knowledge about these logs is readily available, since these logs are created by software. The second use case deals with the detection of data theft. In this scenario, only recording and analyzing users actions may not be sufficient to determine whether a data theft has happened.

To address these three challenges, we contribute in the following ways: First, we devise a Deep Learning - based method for extracting signatures from forensic logs. This method uses labeled log lines to learn a model that is capable of extracting the signatures. This method validates our assumption that such a model is capable of learning the complex relationship within the forensic logs. Secondly, we proposed a method that clusters forensic log-lines according to their signatures in an unsupervised way. Since the second method does not rely on human input, it addresses the lack of supervision. Thirdly, we propose a method that efficiently allows combining available heuristic domain knowledge with labeled knowledge. We evaluated the approaches of the forensic use case on a forensic log and two large system logs. The results of the experiments showed improved accuracy over baseline approaches and more efficient use of human labor by reducing the need for expert input. Thus, our method reduces the need for supervision and adds a pathway to combine available knowledge in the machine learning process. Finally, we devise a data-driven method for generating decoy project folders to detect data theft. We confirm the believability of the generated decoys via a user study. Hence, these decoy projects look real but are intrinsically valueless which renders any interaction with them suspicious. Monitoring interactions with these decoys can be a complementary measure to detect data theft, and thereby reducing the lack of contextual information.

Our overarching goal was to automate data analysis tasks in information security. To this end, we have empirically shown that we can use labels more efficient, address log clustering tasks without labels and create believable deceptive objects that could be used to gather additional contextual information. We believe that a similar strategy could be deployed for other data theft scenarios, with different types of objects. In the forensic use case, similar procedures as the ones demonstrated in this thesis could be applied to automate other tasks, for example, malware analysis or the detection of phishing attacks. The proposed techniques could also be used for the analysis of other data types such as spatial data.

Appendix B

Curriculum Vitae

Stefan was born on February 12th, 1985 Rum in Tirol. He collected knowledge about many theoretical aspects of computer science at the University of Innsbruck, where he obtained both a BSc (in 2009) and an MSc degree (in 2011). Apart from that, he studied for one semester at the University of Copenhagen, Denmark as an exchange student. In Innsbruck, he specialized in data interlinking technologies and knowledge management systems. After that, Stefan obtained a Ph.D. degree from the Technical University of Eindhoven (in 2019), where he gained knowledge about machine learning, in particular, Deep Learning within the context of information security.

Professionally, Stefan deepened his software engineering skills in working for startups. Among others, he helped developing software that enables people to rent out their private parking lots and a software that enabled people to rent and share their cars. In addition to that, Stefan was involved in the efforts to build a startup that aimed to build a high-tech marketing tool.

For a about three years, Stefan researched the combination of gamification and data interlinking at the Semantic Technology Institute in Innsbruck. At TU/e, he first worked on advanced deceptive technologies, and then switched to Deep Learning in the context of information security.

Apart from that, Stefan spent around 2.5 years as a member of the TU/e W&I Ph.D. council, where he helped improve the experience of his fellow Ph.D. students by organizing professional and social events, communicating and dealing with problems that arose.

Appendix C

Survey Methodology

To obtain the literature for the survey presented in Chapter 2 (i.e., Section 2.3), we followed the following protocol. We chose to restrict the sources of this survey by the venue. We selected security venues that published DL approaches and Machine Learning venues that published DL-based approaches on information security issues. Our venue selection criteria were the following. We selected the top 20 journals and conferences based on Security and Cryptography, Data mining, Artificial intelligence on Google Scholar's ranking. We added the top 20 Security conferences from Microsoft Academic Research. We searched for DL approaches in the Security conferences and Journals, and we searched for Security approaches, and DL approaches in the Machine learning areas. We limited the venues by name and the areas of DL and Security by keywords. We have obtained the keywords of DL by conventional technologies and the keywords for Security from the ACM CCS [3] and the IEEE taxonomy [66].

To refine the selection of papers for this survey, we selected papers according to the following criteria. We excluded papers that only proposed ideas, but did not conduct any experiments. We excluded invited papers, talks, posters and workshop papers. We excluded papers that claimed to use DL, but only reference general resources such as Goodfellow et al.'s DL book [47] without providing any additional detail that would enable researchers to reproduce results. Further, we excluded papers that were not explicitly aiming to achieve a security goal. For example, we excluded approaches on object detecting in videos, even though it could be argued that these approaches are useful for surveillance purposes.

We conducted the literature search on the 9th of May 2018 and we time-range of the survey was from 1st of January 2007 to the 8th of May 2018. The begin of 2007 was chosen because it approximately marks the begin of the advent of deep neural networks [128].

We used Scopus [1] and Google Scholar [2]. We have conducted the keyword- and venue-based literature search on Scopus since it is the most comprehensive database of peer-reviewed computer sciences literature. For two venues, NDSS and Usenix-security, we used Google scholar because Scopus did not index them.

[1] https://www.scopus.com/search/form.uri?display=basic
[2] https://scholar.google.com/

Using the previous search criteria, we obtained 177 papers, which we reviewed more carefully. Of these papers, we only kept 77 that did fit our previously defined criteria. We list these venues in alphabetical order.

Security venues We included literature of the following security conference proceedings and journals: ACNS, ACSAC, ARES, ASIACCS, ASIACRYPT, CCS, Computers and Security, CRYPTO, ESORICS, EUROCRYPT, Fast Software Encryption, FC, IACR, ICB, ICC, IEEE Transactions on Dependable and Secure Computing, IWSEC, Journal of Cryptology, Privacy Enhancing Technologies Symposium, RAID, S&P, SACMAT, SIGCOMM, SOUPS, Theory of Cryptography, Transactions on Information Forensics and Security, TrustCom, WISEC, WPES.

Machine learning venues We included literature of following machine learning conference proceedings and journals: ACCV, ACM SIGKDD, ACM Transactions on Intelligent Systems and Technology, ACM Transactions on Knowledge Discovery from Data, Advances in Data Analysis and Classification AISTATS, BioData Mining, BMVC, Computer Vision and Image Understanding, CVPR, Data Mining and Knowledge Discoveries, ECCV, ECML PKDD, ICCV, ICDM, ICIP, IEEE Computer Society Conference on Computer Vision and Pattern Recognition, IEEE International Conference on Big Data, IEEE Transactions on Image Processing, IEEE Transactions on Knowledge and Data Engineering, IEEE Transactions on Pattern Analysis and Machine Intelligence, Image and Vision Computing, International Conference on Pattern Recognition, International Journal of Computer Vision, Journal of Visual Communication and Image Representation, Knowledge and Information Systems, Machine Vision and Applications Medical Image Analysis, PAKDD, Pattern Recognition, Pattern Recognition Letters, RecSys, SDM, shops, Social Network Analysis and Mining, Statistical Analysis and Data Mining, WACV, WSDM

Appendix D

Publications and Awards

Appendix D contains a list of my publications and awards that I have received so far. First, in Section D.1 I list the publications that are relevant for this thesis, and then, in Section D.2 I list other publications. Both sections are ordered by year of publication. In this listing, the order of the authors determines the amount of contribution to the piece of work. First authorship in this listing means that I have designed and implemented the experiments if any and that I have written the largest part of the publication. Third authorship in this listing means that I was provided minor contributions to that publication.

A complete and up to date listing of my publications can be found on my Google Scholar profile[1].

D.1 — Publications relevant for this thesis

2018 Stefan Thaler, Vlado Menkovski, and Milan Petković. Deep metric learning for sequential data using approximate information. In *Machine Learning and Data Mining in Pattern Recognition*, pages 269–282, Springer, 2018

2017 Stefan Thaler, Vlado Menkovski, and Milan Petković. Unsupervised Signature Extraction from Forensic Logs. In *Machine Learning and Knowledge Discovery in Databases - European Conference*, ECML PKDD 2017, *Skopje, Macedonia, September 18-22, 2017, Proceedings.*, pages 305–316, Springer, 2017

Stefan Thaler, Vlado Menkovski, and Milan Petkovic. Towards unsupervised signature extraction of forensic logs. In *Benelearn 2017: Proceedings of the Twenty-Sixth Benelux Conference on Machine Learning, Technische Universiteit Eindhoven, 9-10 June 2017*, pages 154–160, 2017

[1] https://scholar.google.nl/citations?user=kwYKk5EAAAAJ&hl=en

Stefan Thaler, Vlado Menkovski, and Milan Petković. Towards a neural language model for signature extraction from forensic logs. In *2017 5th International Symposium on Digital Forensic and Security (ISDFS)*, pages 1–6. IEEE, 2017

2016 Stefan Thaler, Jerry den Hartog, and Milan Petkovic. Towards Creating Believable Decoy Project Folders for Detecting Data Theft. In *Data and Applications Security and Privacy {XXX} - 30th Annual {IFIP} {WG} 11.3 Conference, DBSec 2016, Trento, Italy, July 18-20, 2016. Proceedings*, pages 161–169, Springer, 2016

Technical Reports

2018 Stefan Thaler, Vlado Menkovski, and Milan Petković. Deep Learning in Information Security. *arXiv cs.CR*, pages 1–36, 2018

Grants and Awards

2017 Best paper award for "Towards a neural language model for signature extraction from forensic logs"

2016 Best presentation award for the talk "Towards Creating Believable Decoy Project Folders for Detecting Data Theft" at the 2nd Cyber Security Workshop in the Netherlands.

D.2 — Other publications

2019 Stefan Thaler and Vlado Menkovski. The Role of Deep Learning in Improving Healthcare. In *Data Science for Healthcare: Methodologies and Applications*, pages 1–42. Springer International Publishing AG, to appear in 2019

2015 Stefan Thaler, Jerry den Hartog, Dhouha Ayed, Dieter Sommer, and Michael Hitchens. Cross-Domain Attribute Conversion for Authentication and Authorization. In *Ares*, pages 652–659. {IEEE} Computer Society, 2015

2013 Stefan Thaler. Game based interlinking. In Georg Güntner und Sebastian Schaffert Christoph Bauer, editor, *Qualitätssicherung bei Annotationen*, pages 5–10. Salzburg Research Forschungsgesellschaft, 2013

2012 Stefan Thaler, Elena Simperl, and Stephan Wölger. An Experiment in Comparing Human-Computation Techniques. *IEEE Internet Computing*, pages 52–58, 2012

Elena Simperl, Stephan Wölger, Stefan Thaler, Barry Norton, and Tobias Bürger. Combining human and computation intelligence: the case of data interlinking tools. *International Journal of Metadata, Semantics and Ontologies*, 7(2):77–92, 2012

Stefan Thaler, Elena Simperl, Katharina Siorpaes, and Stephan Wölger. SpotTheLink: A Game-based Approach to the Alignment of Ontologies. *Collaboration and the Semantic Web: Social Networks, Knowledge Networks, and Knowledge Resources: Social Networks, Knowledge Networks, and Knowledge Resources* pages 40–63, 2012

Dieter Fensel, Birgit Leiter, Stefan Thaler, Andreas Thalhammer, Anna Fensel, and Ioan Toma. Knowledge Modeling of On-line Value Management. In *The Semantic Web: ESWC 2012 Satellite Events*, pages 245–258. Springer Berlin Heidelberg, 2012

Dieter Fensel, Birgit Leiter, Stefan Thaler, Andreas Thalhammer, and Ioan Toma. Effective and efficient on-line communication. In *2012 23rd International Workshop on Database and Expert Systems Applications*, pages 294–298. IEEE, 2012

2011 Stefan Thaler, Elena Paslaru Bontas Simperl, and Katharina Siorpaes. SpotTheLink: A Game for Ontology Alignment. *Wissensmanagement*, 182:246–253, 2011

Stefan Thaler, Katharina Siorpaes, David Mear, Elena Simperl, and Carl Goodman. SeaFish: a game for collaborative and visual image annotation and interlinking. In *The Semanic Web: Research and Applications*, pages 466–470. Springer, 2011

Stefan Thaler, Elena Simperl, and Katharina Siorpaes. Spotthelink: playful alignment of ontologies. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, pages 1711–1712. ACM, 2011

Thesis

2011 Stefan Thaler. *Game based multimedia Interlinking*. Masterthesis, Leopold Franzens Universität, Innsbruck Austria, 2011

2008 Stefan Thaler. *Visions of Semantic Content Management Systems*. Bachelor thesis, Leopold Franzens Universität, Innsbruck, Austria, 2008

Stefan Thaler. *Automatisierte Inventarerkennung in Textdokumenten*. Bachelor thesis, Leopold Franzens Universität, Innsbruck, Austria, 2008

Technical Reports

2011 Stefan Thaler, Katharina Siorpaes, Elena Simperl, and Christian Hofer. A survey on games for knowledge acquisition. *Rapport technique*, STI, pages 1–26, 2011

Grants and Awards

2011 Performance grant of the University of Innsbruck for the year 2011.

IPA Dissertations

Titles in the IPA Dissertation Series since 2015

S.-S.T.Q. Jongmans. *Automata-Theoretic Protocol Programming*. Faculty of Mathematics and Natural Sciences, UL. 2016-01

S.J.C. Joosten. *Verification of Interconnects*. Faculty of Mathematics and Computer Science, TU/e. 2016-02

M.W. Gazda. *Fixpoint Logic, Games, and Relations of Consequence*. Faculty of Mathematics and Computer Science, TU/e. 2016-03

S. Keshishzadeh. *Formal Analysis and Verification of Embedded Systems for Healthcare*. Faculty of Mathematics and Computer Science, TU/e. 2016-04

P.M. Heck. *Quality of Just-in-Time Requirements: Just-Enough and Just-in-Time*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2016-05

Y. Luo. *From Conceptual Models to Safety Assurance – Applying Model-Based Techniques to Support Safety Assurance*. Faculty of Mathematics and Computer Science, TU/e. 2016-06

B. Ege. *Physical Security Analysis of Embedded Devices*. Faculty of Science, Mathematics and Computer Science, RU. 2016-07

A.I. van Goethem. *Algorithms for Curved Schematization*. Faculty of Mathematics and Computer Science, TU/e. 2016-08

T. van Dijk. *Sylvan: Multi-core Decision Diagrams*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2016-09

I. David. *Run-time resource management for component-based systems*. Faculty of Mathematics and Computer Science, TU/e. 2016-10

A.C. van Hulst. *Control Synthesis using Modal Logic and Partial Bisimilarity – A Treatise Supported by Computer Verified Proofs*. Faculty of Mechanical Engineering, TU/e. 2016-11

A. Zawedde. *Modeling the Dynamics of Requirements Process Improvement*. Faculty of Mathematics and Computer Science, TU/e. 2016-12

F.M.J. van den Broek. *Mobile Communication Security*. Faculty of Science, Mathematics and Computer Science, RU. 2016-13

J.N. van Rijn. *Massively Collaborative Machine Learning*. Faculty of Mathematics and Natural Sciences, UL. 2016-14

M.J. Steindorfer. *Efficient Immutable Collections*. Faculty of Science, UvA. 2017-01

W. Ahmad. *Green Computing: Efficient Energy Management of Multiprocessor Streaming Applications via Model Checking*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2017-02

D. Guck. *Reliable Systems – Fault tree analysis via Markov reward automata*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2017-03

H.L. Salunkhe. *Modeling and Buffer Analysis of Real-time Streaming Radio Applications Scheduled on Heterogeneous Multiprocessors*. Faculty of Mathematics and Computer Science, TU/e. 2017-04

A. Krasnova. *Smart invaders of private matters: Privacy of communication on the Internet and in the Internet of Things (IoT)*. Faculty of Science, Mathematics and Computer Science, RU. 2017-05

A.D. Mehrabi. *Data Structures for Analyzing Geometric Data*. Faculty of Mathematics and Computer Science, TU/e. 2017-06

D. Landman. *Reverse Engineering Source Code: Empirical Studies of Limitations and Opportunities*. Faculty of Science, UvA. 2017-07

W. Lueks. *Security and Privacy via Cryptography – Having your cake and eating it too*. Faculty of Science, Mathematics and Computer Science, RU. 2017-08

A.M. Şutîi. *Modularity and Reuse of Domain-Specific Languages: an exploration with MetaMod*. Faculty of Mathematics and Computer Science, TU/e. 2017-09

U. Tikhonova. *Engineering the Dynamic Semantics of Domain Specific Languages*. Faculty of Mathematics and Computer Science, TU/e. 2017-10

Q.W. Bouts. *Geographic Graph Construction and Visualization*. Faculty of Mathematics and Computer Science, TU/e. 2017-11

A. Amighi. *Specification and Verification of Synchronisation Classes in Java: A Practical Approach*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2018-01

S. Darabi. *Verification of Program Parallelization*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2018-02

J.R. Salamanca Tellez. *Coequations and Eilenberg-type Correspondences*. Faculty of Science, Mathematics and Computer Science, RU. 2018-03

P. Fiterău-Broştean. *Active Model Learning for the Analysis of Network Protocols*. Faculty of Science, Mathematics and Computer Science, RU. 2018-04

D. Zhang. *From Concurrent State Machines to Reliable Multi-threaded Java Code*. Faculty of Mathematics and Computer Science, TU/e. 2018-05

H. Basold. *Mixed Inductive-Coinductive Reasoning Types, Programs and Logic*. Faculty of Science, Mathematics and Computer Science, RU. 2018-06

A. Lele. *Response Modeling: Model Refinements for Timing Analysis of Runtime*

Scheduling in Real-time Streaming Systems. Faculty of Mathematics and Computer Science, TU/e. 2018-07

N. Bezirgiannis. *Abstract Behavioral Specification: unifying modeling and programming*. Faculty of Mathematics and Natural Sciences, UL. 2018-08

M.P. Konzack. *Trajectory Analysis: Bridging Algorithms and Visualization*. Faculty of Mathematics and Computer Science, TU/e. 2018-09

E.J.J. Ruijters. *Zen and the art of railway maintenance: Analysis and optimization of maintenance via fault trees and statistical model checking*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2018-10

F. Yang. *A Theory of Executability: with a Focus on the Expressivity of Process Calculi*. Faculty of Mathematics and Computer Science, TU/e. 2018-11

L. Swartjes. *Model-based design of baggage handling systems*. Faculty of Mechanical Engineering, TU/e. 2018-12

T.A.E. Ophelders. *Continuous Similarity Measures for Curves and Surfaces*. Faculty of Mathematics and Computer Science, TU/e. 2018-13

M. Talebi. *Scalable Performance Analysis of Wireless Sensor Network*. Faculty of Mathematics and Computer Science, TU/e. 2018-14

R. Kumar. *Truth or Dare: Quantitative security analysis using attack trees*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2018-15

M.M. Beller. *An Empirical Evaluation of Feedback-Driven Software Development*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2018-16

M. Mehr. *Faster Algorithms for Geometric Clustering and Competitive Facility-Location Problems*. Faculty of Mathematics and Computer Science, TU/e. 2018-17

M. Alizadeh. *Auditing of User Behavior: Identification, Analysis and Understanding of Deviations*. Faculty of Mathematics and Computer Science, TU/e. 2018-18

P.A. Inostroza Valdera. *Structuring Languages as Object-Oriented Libraries*. Faculty of Science, UvA. 2018-19

M. Gerhold. *Choice and Chance - Model-Based Testing of Stochastic Behaviour*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2018-20

S.M.J. de Putter. *Verification of Concurrent Systems in a Model-Driven Engineering Workflow*. Faculty of Mathematics and Computer Science, TU/e. 2019-01

S.M. Thaler. *Automation for Information Security using Machine Learning*. Faculty of Mathematics and Computer Science, TU/e. 2019-02