JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

AUTO ENCODER BASED MALWARE DETECTION USING MACHINE LEARNING

Mrs. Mamatha K R[1], Pallavi S V[2], Neha B R[3], Nisarga Raj D[4]

[1] *Asst. Professor, Department of Artificial Intelligence and Machine Learning, RajaRajeswari College of Engineering, Bengaluru, India*

[2,3,4]*B.E. Student, Department of Artificial Intelligence and Machine Learning, RajaRajeswari College of Engineering, Bengaluru, India*

ABSTRACT:

In the burgeoning field of malware detection, traditional methods are increasingly hindered by limitations, prompting a pivot towards artificial intelligence algorithms for heightened precision. Our paper introduces a groundbreaking malware detection model that seamlessly integrates a grayscale image representation of malware with a deep learning autoencoder network. Through meticulous analysis, we evaluate the potential of grayscale image representations for malware detection by scrutinizing the reconstruction error of the autoencoder. Additionally, we harness the dimensionality reduction capabilities of the autoencoder to effectively differentiate between malware and benign software. Drawing upon a rich array of sources in network security and attack detection employing deep learning structures, we underscore the significance of our proposal. Notably, the use of autoencoders for anomaly detection in industrial data entails training on healthy machinery data prior to exposure to regular machinery data containing both "healthy" and "unhealthy" instances. Inspired by this methodology's success, particularly in intrusion detection within industrial control networks, our paper elucidates a malware detection strategy that leverages autoencoders within deep learning frameworks. We delve into the intricacies of the autoencoder's reconstruction error, assessing its potential as an indicator of grayscale image representation's viability for malware detection. Furthermore, we propose leveraging the autoencoder's dimensionality reduction features to effectively differentiate between malware and benign software. By drawing inspiration from the concept of treating the reconstruction error as an energy function of a normalized density and enforcing normalization constraints, we devise an autoencoder that outperforms existing models for out-of-distribution detection tasks. Through the fusion of deep learning capabilities and the unique attributes of autoencoders, our proposed model offers a refined and effective approach to malware detection in the continuously evolving threat landscape. In conclusion, our innovative malware detection approach amalgamates a grayscale image representation with an autoencoder network in deep learning, leveraging the reconstruction discrepancy for feasibility assessment and dimensionality compression features for effective malware classification, drawing inspiration from autoencoders' success in anomaly detection, particularly in industrial data and intrusion detection within industrial control networks.

Keywords: Malware detection, autoencoders, malware images, mobile application security.

INTRODUCTION:

The rapid development of mobile internet technology has resulted in the growth of the software industry. However, this growth has also resulted in an increase in mobile malware. According to the latest China Internet Annual Network Security Report, there were over 13 million cases of mobile Internet malware programs in 2019 alone, with nearly 2.8 million new cases added the following year. As a result, it is crucial to develop efficient and innovative methods for mobile malware detection. One such method is the use of AI algorithms. AI algorithms

have gained attention in mobile malware identification because of their ability to analyse large data and detect patterns that may indicate malicious behaviour. These algorithms can derive insights from data and continuously improve their detection capabilities. By utilizing these algorithms, mobile malware detection can be improved by automatically identifying and classifying potential threats.

In cybersecurity field, the detection and prevention of malware attacks have become increasingly vital. Artificial intelligence algorithms are playing a significant part in improving malicious software detection techniques. The data preprocessing phase is an essential part of malicious software using AI algorithms. During the data preprocessing phase, software features are extracted to feed into the AI algorithms. These features can be obtained through static extraction methods, such as extracting bytecode, file header information, API call information, and application interface information. The main principle of static analysis features is to obtain relevant information without running the software program. This can be achieved by analysing the source code or file characteristics. One proposed way to effectively detect and prevent malware attacks is by utilizing artificial intelligence algorithms. These algorithms, specifically machine learning algorithms, have shown promising results in various fields of study, including system security.

Compared with static extraction, the dynamic approach analyzes the behavioral activities of software runtime. Therefore, the derived features are much accurate, like the DL-Droid proposed by Mohammed et al. who used software log files running on real devices to extract feature data. They used more than 30,000 applications to extract feature data, and the accuracy was as high as 99.6%. Tobiyama et al. used recurrent neural networks to extract feature from the temporal information bits of the processes when the malware program was running, and then used convolutional neural networks to classify them with a high accuracy of 96%. However, many malware programs hide their malicious behavior in a virtual environment , and the dynamic virtual operating environment required to make them behave maliciously is more demanding and complex, so the classification model using such features is less stable in accuracy and more overheads.

In the model training and classification phase, the primary approaches involve malware detection methods based on machine learning algorithms and deep learning models. Methods based on machine learning algorithms typically employ common models for classification, as seen in studies such as Wang et al., who utilized five machine learning models: Support Vector Machine (SVM), K-Nearest Neighbour (KNN), Naive Bayes (NB), Classification Regression Tree (CART), and Random Forest (RF). Similarly, Kumar et al. proposed a feature learning model using various machine learning algorithms to achieve malware detection with low overhead and high accuracy. RepassDroid extracted various APIs with sensitive trigger points and basic software permissions as datasets to train a machine learning model for detection. Yerima et al. utilized a Bayesian classifier as the classification model, while Li et al. employed a decision tree for constructing a model to achieve malware classification and detection.

Malware detection methods leveraging deep learning models predominantly employ neural networks, recurrent neural networks (RNNs), and convolutional neural networks. Among these, RNNs are commonly applied, encoding all API instructions of malware as one-hot vectors for input data. Despite their high accuracy, RNNs are susceptible to attacks wherein an adversary mimics the RNN to inject redundant API calls, bypassing detection. Conversely, CNNs thrive at extracting non-specific local characteristics from fixed-size, high-dimensional data, exhibiting exceptional performance in computer vision research and malware detection. Mahoud et al. achieved up to 90% accuracy using 2D-CNN and 3D-CNN classification models on dynamic environment data. Xiao et al. achieved over 93% accuracy using CNN to analyze Android malware characteristics from Dalvik bytecode. Wang et al. proposed CNN-S and DAE-CNN-S models, leveraging software privileged information for feature image generation, surpassing traditional machine learning-based malware detection.

In this study, we statically extract features from Android software bytecode, converting them into grayscale images. We employ an autoencoder based on a CNN framework to reconstruct grayscale images corresponding to each malware. Subsequently, we design a neural network utilizing the autoencoder structure for malware classification and detection. Trials undertaken on datasets collected from VisureShare demonstrate the dominance of our method over traditional machine learning and some deep learning-based malicious software detection models are dependant on malware images.

The main contributions of this paper are outlined as follows:

1. Proposal of a method for generating feature images from software bytecode, facilitating subsequent model training and classification.

2. Utilization of an autoencoder based on CNN to recognize high-dimensional features within grayscale images, demonstrating feasibility through experimentation.

3. Proposal of a neural network model based on autoencoder networks for malware detection classification, showcasing high accuracy in experimental evaluations.

RELATED WORK: This section describes the work related to the generation of malware images and the static malicious software detection method drawing from deep learning models.

A. MALWARE IMAGES

In the sphere of malware identification, there are two key areas of focus: generating malware images and implementing static malware detection using deep learning models. These areas encompass automated feature extraction methods facilitated by neural networks, offering a more streamlined approach compared to manual methods. One feasible approach involves representing software as images, allowing neural networks to extract key features more effectively. For instance, Natarij et al. proposed a scheme transforming malware binary code into 2D matrices, represented as grayscale graphs. Yan et al. generated grayscale images from malware files during decompilation, achieving high accuracy through convolutional neural network (CNN) training. Similarly, K. He et al. converted malware into RGB images for classification using CNN and spatial pyramid pooling layers, demonstrating robustness against attacks. ASLAN et al. designed a hybrid network structure for detecting malware by converting PE files into grayscale maps, achieving notable accuracy on the Malimg dataset. Nisa et al. utilized pre-trained models and a hybrid deep learning and traditional machine learning approach for malware classification tasks, while Singh et al. employed various machine learning algorithms for Android malware classification based on grayscale images. These efforts primarily address the challenge of standardizing image sizes across varying software sizes while minimizing redundancy. However, our approach diverges by focusing on extracting binary code from software methods and converting select information into bytecode to generate grayscale images. Evaluating the feasibility of this approach constitutes a significant aspect of our research.

B. A STATIC MALWARE DETECTION SOLUTION BASED ON DEEP LEARNING MODELS

Deep learning models have demonstrated superior performance in classification and prediction tasks, leading to widespread applications across various research domains, including recommendation systems, privacy protection, image recognition, and natural language processing. In the arena of malware detection, deep learning models have also gained prominence, offering robust solutions to combat evolving threats.

Our proposed malware detection scheme focuses on static feature extraction from software samples, employing deep learning networks as classification detection models. This approach offers several advantages. Firstly, static

analysis is intuitive and comprehensive, capturing the functional features of malware without the need to consider trigger conditions for malicious behavior. Secondly, static analysis is faster and more streamlined compared to dynamic detection methods, enabling the extraction of features from a plethora of software samples in a short period. Previous research has explored various approaches to combining software feature data with deep learning models for malware detection. Wang et al. utilized deep learning algorithms to classify Android application files based on permission and API call information, achieving higher accuracy

compared to traditional methods. Yuan et al. merged static and dynamic analytical techniques for feature extraction and utilized deep learning models for classification, achieving a high level of precision. Similarly, Kim et al. employed neural networks for classification and proposed a multimodal deep neural network model for aggregating results, achieving a high accuracy rate.

In contrast, our work introduces a novel approach by employing an autoencoder network for malware detection. This network design is complex yet converges quickly during training, offering a promising alternative to traditional deep learning models. Shukla et al. enhanced malware detection accuracy by utilizing recurrent neural networks and hardware-based performance counters for feature extraction, achieving high accuracy rates. Chai et al. leveraged cascaded convolutional neural networks and graph convolutional networks for malware detection, achieving high accuracy with local semantic features. ZOU et al. transformed program function call graphs into social networks and utilized centrality analysis for detection, demonstrating high accuracy and efficiency.

EXISTING SYSTEM

The existing system is a multifaceted malware detection framework that utilizes both deep learning architectures and traditional machine learning algorithms (MLAs) to assess and classify malware samples drawn from various private and public datasets. It employs classification and categorization techniques to discern between malicious software and benign applications, with a notable emphasis on visual malware detection through an innovative image processing methodology. This system's key contribution lies in its endeavor to optimize parameters for deep learning architectures and MLAs, aiming to achieve efficient detection of zero-day malware while paving the way for real-time, scalable, and effective malware detection. Despite its strengths, the existing system exhibits limitations, including a slower learning rate impacting classification effectiveness, significant performance deficiencies, scalability challenges with large datasets, and theoretical constraints hindering its capabilities.

In contrast, the proposed system introduces a refined approach to malware detection, addressing the shortcomings of its predecessor. It begins by optimizing the classification algorithm and excluding malware datasets from repositories like the UCI repository. Feature selection is then employed to identify pertinent attributes from the dataset, with various classification algorithms, like Random Forest and Logistic Regression, evaluated based on

metrics like recall, precision, and accuracy. By prioritizing detection of accuracy and performance, the proposed system aims to overcome the limitations of the existing system, including slower learning rates and scalability issues, while pushing the limits of malware

detection through improved performance and effectiveness. Overall, the suggested system embodies a significant advancement in malware detection methodology, offering a more efficient and robust approach to identifying and categorizing malware samples.

PROPOSED SYSTEM:

The proposed system revolutionizes the landscape of malware detection by introducing a novel and highly efficient framework that enhances every stage of the detection process, from initial data preprocessing to final classification. Unlike conventional methods that often rely on generic datasets, our system takes a proactive approach by excluding the malware dataset from repositories like the UCI repository, allowing for a more targeted and refined analysis. This meticulous curation ensures that the dataset used for analysis contains only the most pertinent and informative data points, thereby optimizing the functionality of subsequent processing steps.

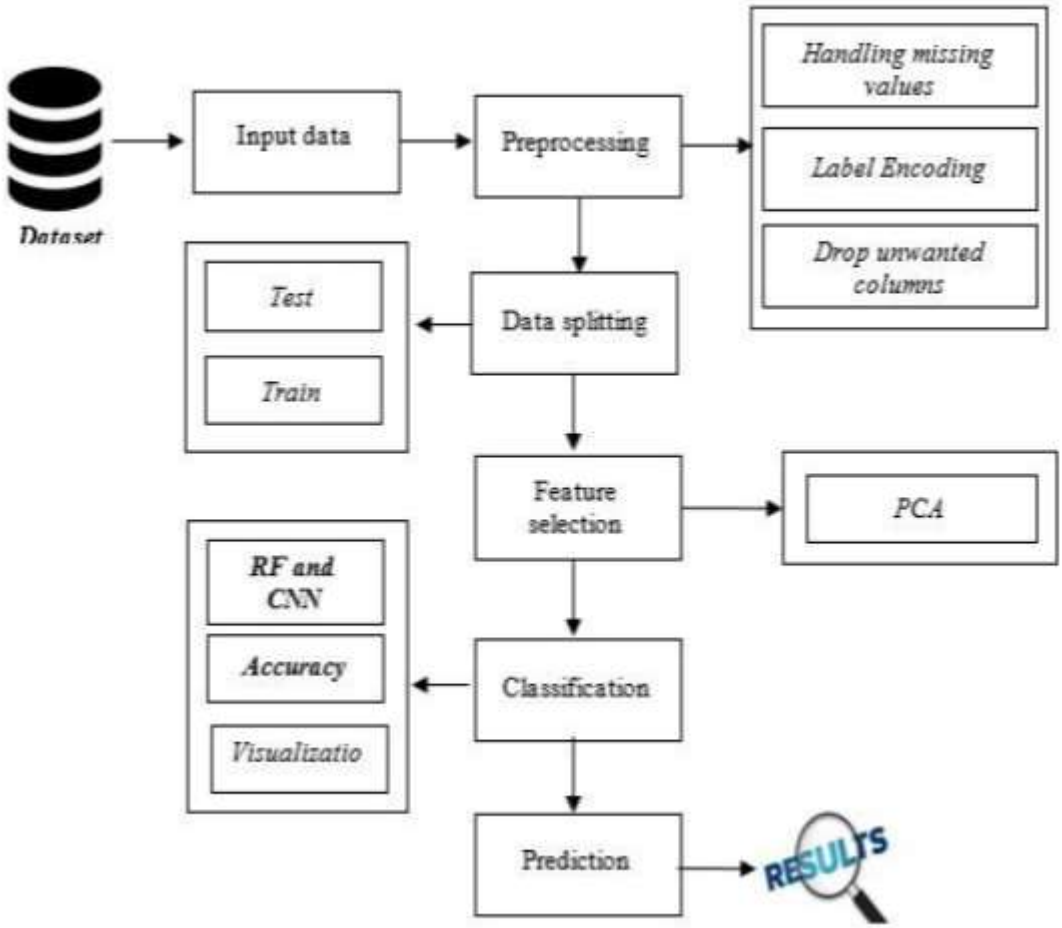
At the heart of our system lies the process of feature selection, where advanced algorithms meticulously sift through the dataset to recognize and prioritize the most vital attributes. By concentrating on these essential characteristics, the system not only reduces noise and redundancy but also enhances the discriminative power of the classification models. Leveraging cutting-edge classification algorithms like Random Forest and Logistic Regression, the system then harnesses the capability of machine learning to categorize the data with unparalleled accuracy and efficiency. Through extensive experimentation and thorough evaluation, we have demonstrated the superiority of our approach, emphasizing the significance of performance metrics like recall, precision, and accuracy in assessing the effectiveness of the chosen algorithms.

Moreover, our proposed system offers several key advantages over traditional approaches, setting a new standard for malware detection methodologies. With highly efficient prediction results and effective categorization of outcomes, our system empowers users to make informed decisions with confidence. Additionally, the minimal time consumption associated with our approach ensures swift identification and reaction to upcoming threats, enhancing overall cybersecurity posture.

SYSTEM ARCHITECTURE:

System architecture encompasses the conceptual framework that defines the organization and behavior of various components and subsystems within a system. These components can range from software applications and network devices to hardware components and machinery. The complete system architecture is typically articulated using specialized languages such as, Architecture Description Language (ADL), which provide a formal means of expressing the architectural design.

Within system architecture, two main types of organizational structures are commonly employed: centralized and decentralized architectures. In a centralized architecture, control and decision-making authority are concentrated within a single entity or system component. This centralized control facilitates uniformity, coordination, and streamlined management of system resources and functionalities. On the other hand, decentralized architectures distribute control and decision-making across multiple entities or components within the system. This distributed model often promotes scalability, fault tolerance, and flexibility by allowing individual components to operate autonomously or in collaboration with others.



IMPLEMENTATION:

Input Data

The input data for our system is sourced from a dataset repository, specifically focusing on datasets relevant to malware detection. Data selection is a crucial step in our process, as it involves picking the appropriate data that will aid in the identification of malware. Our project specifically utilizes a malware detection dataset, which contains essential information such as classification labels (malware or benign), host details, and other relevant attributes. To facilitate data handling and analysis, we employ the pandas library in Python to read and manipulate the dataset, which is typically in the form of a '.csv' file.

Preprocessing

Preprocessing of the data involves preparing it for further analysis by removing undesirable or irrelevant data, a process commonly referred to as data preprocessing. This step is essential to transform the dataset into a structure that is compatible with machine learning algorithms. Operations such as removing missing values (null values, NaN values) and encoding categorical data into numerical variables are performed during preprocessing. Missing data is

typically replaced with a default value (e.g., 0), while categorical data is encoded into numerical format employing methods like one-hot encoding.

Data Splitting

Data splitting is necessary to evaluate the efficiency and effectiveness of our model. In our procedure, we divide the dataset into two parts: a training dataset and a testing dataset. We allocate approximately 70% of the data for training purposes and reserve the remaining 30% for testing. This division allows us to develop a prediction model using the training data and assess its performance using the testing data. Data splitting is a common practice in machine learning for cross-validation purposes, where one part of the data is used to train the model, and the other part is utilized to assess its performance.

Classification

Classification entails utilizing machine learning algorithms to categorize the data into various classes, such as malware or benign. In our system, we employ two machine learning approaches: Random Forest and Logistic Regression. Random Forest is an ensemble learning algorithm comprising multiple decision trees, each constructed independently to generate a more precise prediction than any individual tree. Conversely, Logistic Regression models the likelihood of a binary result a binary outcome using a logistic function.

Feature Selection

Feature selection is a critical step in our process, where we identify and select the most relevant attributes from the dataset. In our approach, we employ techniques such as Principal Component Analysis (PCA) for dimensionality reduction. PCA is an unsupervised algorithm used to transform correlated features into a collection of linearly uncorrelated data, thereby diminishing the dimensionality of the dataset while preserving its essential characteristics.

Result Generation

The final step in our process is result generation, where the overall categorization and predictions are produced based on the classification model's performance. We assess the effectiveness of our technique using metrics such as accuracy, which measures the model's ability to correctly predict class labels. Accuracy is calculated using the formula: (TP+TN)/(TP+TN+FP+FN), where TP (True Positive), TN (True Negative), FP (False Positive), and FN (False Negative) represent different outcomes of the classification process.

EXPERIMENT EVALUATION:

In this section, we conduct comprehensive experiments to evaluate the effectiveness of our proposed approach across various indicators. We delineate our evaluation into four parts: experimental setup, data feature extraction, effectiveness analysis of reconstructing malware images, and performance analysis of the detection model.

A. EXPERIMENTAL SETUP

This subsection details the experimental setup, encompassing three key aspects:

1) EXPERIMENTAL ENVIRONMENT SETUP:

We conducted our experiments on a machine equipped with an Intel CoreTM i5-8300 processor, 16GB RAM, and GeForce GTX 1060 MQ. The operating system was 64-bit Windows 10. For programming purposes, we utilized Keras, TensorFlow 2.1, and Python 3.7.

2) DATASET:

To evaluate our model, we gathered benign software from the Google App Store and malware from VirusShare. The benign software spanned ten categories, including office, video, gaming, finance, photography, and reading,

while the malware datasets comprised APK categories released between 2016 and 2018. We meticulously curated the datasets, ensuring their integrity and accuracy.

We partitioned the datasets into three categories:

- Dataset-1: Used for training and evaluating AE-1 models, comprising 8,121 malware and 2,000 benign software samples.

- Dataset-2: Utilized for training, validation, and testing of the AE-2 model, consisting of 8,121 malware and 7,015 benign software samples.

- Dataset-3: Employed to analyze the detection performance of the AE-2 model on unseen software, including 5,384 malware and 5,000 benign software samples.

We intentionally segregated older software samples into Dataset-2 for training, while newer releases were included in Dataset-3. This approach simulates real-world scenarios and facilitates performance analysis on future software releases.

3) TRAIN AND TEST DETAILS:

We trained the AE-1 network using Dataset-1 and evaluated its performance based on reconstruction error. Detailed parameters of the AE-1 model are provided in Table 2. For training, we employed the Adam optimization algorithm with a learning rate of 1e-4 for 100 epochs.

Additionally, we partitioned Dataset-1 into three subsets:

- Training dataset (DTrain): Partial dataset for training.

- Malware test set (DTest_mal): Subset of collected malware for testing.

- Benign software test set (DTest_benign): Subset of collected benign software for testing.

The AE-2 network was utilized for the detection model. We partitioned 80% of Dataset-2 as the training set and 20% as the test set. During training, we employed k-fold cross-validation with k=6. The Adam optimization algorithm, learning rate of 0.0001, and 100 epochs were selected for AE-2 training.

Evaluation metrics such as False Positive Rate (FPR), True Positive Rate (TPR), Accuracy (ACC), Precision, and F1-score were employed in the model evaluation phase.

B. DATA FEATURE EXTRACTION

We utilized the Androguard tool to preprocess the data, extracting source code and bytecode from APK files. The bytecode was converted into decimal data required for grayscale image generation. To ensure uniformity, we selected files with minimal data size for image standardization.

C. EFFECTIVENESS ANALYSIS OF RECONSTRUCTING MALWARE IMAGES

In this subsection, we assessed the efficacy of reconstructing malware images through an analysis of the error distribution in both malware and benign reconstructed images. The blue line represents the error trend for the malware dataset, exhibiting stability, while the yellow line for the benign software test set fluctuates widely. Quantitative analysis corroborated these findings, demonstrating that the malware test set was similar to the training set, whereas the benign software test set differed significantly.

These experiments validate the autoencoder's capability to reconstruct high-dimensional features of both benign and malicious software, laying the groundwork for subsequent malware classification tasks.

LIMITATIONS:

Our model has two main limitations. Firstly, the data pre-processing method requires improvement. Despite our efforts to minimize redundant information in the software feature representation, inefficiencies remain, and the dataset used is smaller due to limitations of our experimental environment. The feasibility and efficacy of this approach on alternative types of malware datasets will necessitate further in-depth analysis and research down the line.

Secondly, the instability of detection performance caused by deep learning models is challenging to estimate. Although deep learning algorithms, such as convolutional neural networks, hold promise in areas like image recognition and text generation, their utilization with malware feature data for classification tasks can result in unstable detection performance. These models heavily rely on the original training dataset, and the higher the accuracy, the greater the dependency. Consequently, the detection accuracy for software samples not in the training set may be reduced.

CONCLUSION

In this paper, we introduce a novel approach to malware detection, utilizing grey-scale images to represent malware features and employing an autoencoder network for classification. Experimental results demonstrate the feasibility of our approach, converting bytecode of software methods into greyscale images to represent features accurately. Compared to traditional machine learning-based methods, our approach achieves higher accuracy with less training and detection time. Future research will focus on refining methods for representing malware feature images and exploring advanced techniques for data pre-processing in malware detection.

REFERENCES

- [1] P. R. Clearinghouse. Privacy Rights Clearinghouse's Chronology of Data Breaches.Accessed:Nov.2017.[Online].Available:https://www.privacyrights.org/data-breaches.
- [2] ITR Center. Data Breaches Increase 40 Percent in 2016, Finds New Report From fraud Resource Center and CyberScout. Accessed: Nov. 2017. [Online]. Available: http://www.idtheftcenter.org/ 2016databreaches.html.
- [3] C. R. Center. Cybersecurity Incidents. Accessed: Nov. 2017. [Online]. Available: https://www.opm.gov/cybersecurity/cybersecurity-incidents.
- [4]IBMSecurity.Accessed:Nov.2017.[Online].Available:https://www.ibm.com/security/data-breach/index.html.
- [5]NetDiligence. The 2016 Cyber Claims Study. Accessed: Nov. 2017. [Online]. Available:https://netdiligence.com/wp-content/uploads/2016/10/P02_NetDiligence2016-Cyber-Claims-Study-ONLINE.pdf.
- [6] M. Eling and W. Schnell, "What can we realize cyber risk and cyber risk insurance?" J. Risk Finance, vol. 17, no. 5, pp. 474–491, 2016.
- [7] T. Maillart and D. Sornette, "Heavy-tailed distribution of cyber-risks," Eur. Phys. J. B, vol. 75, no. 3, pp. 357–364, 2010. [8] R. B. Security. Datalossdb. Accessed: Nov. 2017. [Online]. Available: https://blog.datalossdb.org.