Attack Forecast and Prediction

Tobias Budig[a], Elisabeth Goebel[a], Tessa Fischer[a] and Jurek Muff[a]

[a] *Karlsruhe Institute of Technology (KIT), Kaiserstraße 12, 76131 Karlsruhe, Germany*

Abstract

Cyber-attacks are threatening the security of society, infrastructure, health care, economy and politics. However, cyberspace attackers often have a significant first-mover advantage leading to a dynamic arms race with defenders. Cyber Threat Intelligence (CTI) on past attacks bears the potential to predict future attack patterns and prepare appropriate countermeasures. Within this work, we present Attack Forecast and Prediction (*AFP*), which is based on MITRE Adversarial Tactics, Techniques and Common Knowledge (ATT&CK). *AFP* consists of different modules representing different methods and their evaluation. These approaches are hierarchical clustering, principal component analysis (PCA), time series, genetic algorithms (GA) and generative adversarial networks (GAN). The objective of *AFP* is to identify trends in the usage of attack techniques and crafts forecasts and predictions on future malware. *AFP* is evaluated in comparison with a statistical simulation based on the observed probabilities of single techniques. In the end, all approaches showed promising results but the genetic algorithm and the time series performed best compared to the baseline.

The knowledge we gained enables a deeper understanding of adversarial behaviour and helps with threat awareness. Potential applications of our research include proactive allocation of human, technical and research resources.

Keywords

Attack Prediction, Cyber Threat Intelligence, Genetic Algorithm, Time Series Analysis, Generative Adversarial Networks

Contents

1. Introduction

1.1. Motivation

Cyber-security has emerged as one of the most pressing issues confronting our globally connected world. The World Economic Forum estimated the damage related to worldwide cyber-crime to be $3 trillion in 2015. This number is expected to increase by 15% every year, reaching $10.5 trillion annually by 2025 [1] [2]. Consequently, individuals, businesses and governments are becoming increasingly concerned about the costs and threats presented by cyber-crime, espionage, and cyber-warfare [3]. It is expected that the worldwide information security market will reach $170.4 billion in 2022 [4].

Attackers' strategies rapidly develop and are subject to dynamic innovations. The cyber-criminal world is evolving to exploit vulnerabilities in a faster and more profitable way. To counter this threat, new approaches and investments in the field of cyber-security are essential. ”The capabilities, persistence, and complexity of adversarial attacks in the present threat landscape result in a speed race between security analysts, incident responders, and threat actors."[5] Thereby the attacker appears to have a *first mover advantage.* Thus companies are often vulnerable even to relatively basic assaults on their computer networks.

According to the Global Information Security Workforce Study, the global cyber-security workforce will be short by 1.8 million people by 2022, a 20% increase since 2015 [6]. 66% of respondents reported not having enough capacity to address current threats appropriately. The resulting consequences emphasise the importance of gaining knowledge about cyber-attacks to understand adversarial behaviour and increase the efficiency of dealing with threats [6]. Understanding and analysing cyber-attacks that happened in the past and predicting patterns of attacks for the future means improving cyber-security in its ability to enhance one's position in the arms race between adversaries and defenders.

Therefore, the predictive analysis will give organisations an advantage to effectively allocate their scarce defence resources. Although predicting attacks is not a new procedure, automating attack forecasting and predictions were not options due to technological constraints. Instead, attack predictions were primarily based on subjective perceptions of experienced experts from the cyber-threat landscape. However, experienced experts are rare, and their time is even scarcer.

Automation of attack forecasting and predictions would substantially decrease biases in predictions and minimise experts' time spent on generating forecasts.

1.2. Problem statement

Predicting future malware and its functioning is hence of especial interest. Furthermore, predictions can increase cyber-security maturity. Research and development of defensive measures take much time, whereas better prediction can reduce the time advantage possessed by the attackers. It is important to note that security spending is an investment in the future security of a company and should hence follow the dynamics of attacks. Cyber Threat Intelligence (CTI) (e.g. as it is provided by MITRE ATT&CK; for an in-depth discussion on CTI and the relation to MITRE ATT&CK, refer to section 2) represents an opportunity for making predictions more accurate. However, currently, predictions on future developments of attacks are rare and often are primarily based on experiences of some analysts rather than CTI. This low level of automation of prediction processes causes many problems for cyber-security. First, the act of generating predictions distracted experts from their operations, adding to their existing workload. Second, even the best and most experienced experts perceive cyber-attacks from a limited and subjective perspective, leaving predictions more prone to biases.

1.3. Research question and study approach

To support cyber-security staff on a strategic level and make predictions on the development of attacks more accurate, we investigated two questions:

i Are there any patterns or trends in the MITRE ATT&CK dataset that can be used for crafting medium to long term predictions in the threat landscape?

ii How do different algorithms perform to predict future malware?

The above questions were addressed by the following sequence of activities: To address these questions, we collected historic malware data by scraping information about software and their used techniques from the MITRE ATT&CK database. We subsequently cleaned the data and prepared it for further use. In the next step, we first conducted a simple statistical analysis to identify the probability of techniques used by a software and the distribution of the number of techniques per software. We used the above analysis to generate insights into essential malware techniques and how often they have been used in the past. Furthermore, this will provide decision-makers with data instead of intuition to guide their decision-making process. For the next step, a prediction of future combinations of techniques is desirable. After that, we compared approaches to predict impending attacks by fine-tuning the model by propagating trends.

Here we use, on the one hand, genetic algorithms to generate malware predictions [7]. On the other hand, we made a dimensional reduction and hierarchical clustering. Time-series analysis was conducted to identify trends inside these clusters [8] [9]. Lastly, we trained a Generative Adversarial Network (*GAN*).

To evaluate our algorithms, we used the F-score and a simulation-based on probabilities as a baseline.

In contrast to past work (see 2), we use predictive analytics, not on an operational but a strategic level.

1.4. Contribution

In this work, we present *AFP* taking advantage of CTI for automatically crafting predictions on future malware and the attack techniques used. In doing so, *AFP* leverages CTI to infer patterns within a time series of attacks, making it possible to gain insights into attack evolution and development as well as deriving further relevant information (e.g. the popularity of specific attack techniques) and craft forecasts based on this information. In this way, analysts, researchers and security managers can gain insights into the future cyber-threat landscape and preempt threats that are likely to occur in the future. Additionally, cyber-risk managers can perform intelligent and strategic investments relying on *AFP*, as well as staff training to minimise the attackers' first-mover advantage. The ability of *AFP* to predict the future course and development of the threat landscape is a critical step towards increasing levels of cyber-defence and security as well as its automation. The development of an automated prediction process is an essential step towards the strategic defence against cyber-attacks and can significantly increase cyber-security maturity for non-specific attacks. In the long run, this anticipation of attacks can be expanded and used for successful attack prevention.

2. State of research and related work

2.1. Cyber-situational awareness

Situational awareness may be described as "perception of the elements in the environment within a volume of time and space, the comprehension of their meaning and the projection of their status in the near future."[10] In the context of cyber-security, situational awareness may be divided into three levels [11]. These are (1) monitoring of cyber-systems and intrusion detection, (2) understanding of the current situation and its significance for cyber-security, and (3) projection. The last aspect includes predictive capabilities and is hence the reciprocal link to this work.

2.2. Cyber threat intelligence

CTI is structured information extracted from monitored systems or intrusion detection systems [12]. It includes actionable information on past attacks (evidence-based knowledge). CTI is often divided into four categories: (1) technical, (2) operational, (3) tactical, and (4) strategic threat intelligence [13]. It includes tactics, techniques and attack patterns (TTP), indicators of compromise (IOCs), tools, threat actors, date of discovery. Information extracted from various sources of CTI is leveraged by many analysts to increase the efficacy of defensive measures such as anomaly detection systems, intrusion detection systems, and threat hunting [12].

Since a single individual, security analyst, security researcher or any other expert cannot acquire all information on all threats, there is high importance of sharing CTI among different stakeholders to enable a holistic perspective [12]. Hence, there were significant efforts to formalise and standardise threat sharing and to develop a common language. One of those languages is the Structured Threat Information Expression (STIX) language [1], which is also utilised by MITRE ATT&CK[2].

MITRE ATT&CK is a "globally accessible curated knowledge base and a model of adversarial behaviour in cyber-attacks based on real-world observations." [?] The MITRE Cooperation created ATT&CK out of the need to categorise and structure data pertaining to adversarial behaviour due to the increasing number and relevance of cyber-attacks.

Through MITRE ATT&CK, a common taxonomy has been created to help understand adversarial behaviour and improve defensive actions. MITRE ATT&CK is used as the foundation for developing specific threat models by researchers, analysts and developers. The first model was created in 2013, primarily focusing on the Windows enterprise environment. Since then, the database has been extended to other platforms such as Linux, macOS and Android.

The foundation of MITRE ATT&CK is based on various techniques an adversary can use, representing how a perpetrator will carry out an attack tactic. Each technique is associated with at least one tactic. Tactics can be understood as phases of an attack and are therefore consistent with the cyber kill chain [14]. These tactics answer the question of why an adversary uses a particular technique. The sequence of several techniques used during an attack is defined as software. The software itself can be divided into malicious software (Malware) and legitimate software (Tools).

The most recent version of MITRE ATT&CK represents 552 techniques across 13 tactics and 585 different software. MITRE ATT&CK consists of two parts: The Enterprise version focuses on adversarial behaviour against enterprises, while the mobile version focuses on attacks against mobile devices.

Furthermore, besides CTI for attacker modelling, there is also information on defender modelling, including information on system vulnerabilities. An example of this is the National Vulnerability Database (NVD)[3] or the Common Vulnerabilities and Exposures (CVE) database[4]. This information can be used to understand trends, patterns and developments in software vulnerabilities that would affect the threat landscape.

2.3. CTI Based Predictions For Cyber-Security

Researchers have shown how discrete or continuous models and machine learning methods could be applied to the cyber-security sector in recent years. Husák et al. [9] has made a detailed comparison of predictive methods applicable for both long term investigations and forecasts as well as short term predictions, e.g. used for efficient threat hunting in cyber-security and divided them into

[1] https://www.mitre.org/capabilities/cybersecurity/overview/cybersecurity-blog/stix-20-finish-line

[2] https://attack.mitre.org/

[3] https://nvd.nist.gov/

[4] https://cve.mitre.org/

classes.

2.3.1. Short Term Predictions

Short-term attack projection assists security analysts in identifying the next step of an adversary. One example is the Attack Hypothesis Generator (AHG) by Elitzur et al. [12]. Their work used a knowledge graph of historical malware based on the MITRE ATT&CK, AlienVault Open Threat Exchange (OTX), and VirusTotal. Based on the knowledge graph, they predicted subsequent and linked attack techniques based on observed data. The AHG performed significantly better than an analyst in estimating the next step of an ongoing cyber-attack [12]. Within their work, Elitzur et al. [12] proved that short term predictions could be beneficial for improving cyber-security and setting effective defensive measures in place.

Furthermore, Zhan et al. [15] demonstrated the usage of short term attack predictions relying on honeypots. They enabled attack predictions up to five hours ahead based on data gained from their honeypot. Additionally, Fava et al. [16] presents a methodology for projecting attacks based on information gathered from Intrusion Detection Systems (IDS).

Each of the techniques mentioned above craft predictions to support security analysts at an operational level in their day-to-day work. Husák et al. [9] showed that a wide range of prediction methods achieve up to 90% accuracy in recognising adversarial network behaviour.

Further works are given by Qin and Lee [17]

2.3.2. Medium To Long Term Predictions

Zhang et al. [18] provide a study applying data-mining and machine learning for predicting the ”time to next vulnerability for a given software application”. However, they concluded that the National Vulnerability Database (NVD) has low predictive power. This might be owing to newer attacks exploiting specific vulnerabilities. Furthermore, Ozment [19] highlighted that there is not enough information in freely accessible vulnerability databases, including NVD. That is, CTI about attacks might have higher predictive power than data on vulnerabilities. Further contributions investigating the possibility of predicting vulnerabilities include the works from Alhazmi and Malaiya [20], Abraham and Nahir [21], and Nguyen and Tran [22].

3. Data Set and Preprocessing

We used the data set of observed adversarial software provided by the MITRE ATT&CK Framework. It contains information about 493 software and 552 unique techniques and sub-techniques used by this software. When we started our research, we needed to scrape the data from the MITRE ATT&CK website, aiming to get a matrix of all software and their used techniques. To analyse trends, we added the date of insertion from the MITRE database into our matrix. The result is a matrix with software on the one and techniques on the other axis, identified by their unique ID (e.g. S0056 or T1080) and binary values according to the appearance or non-appearance of a technique in a software. On the 29th of April 2021, MITRE released the newest update, v.9.0, including downloading the needed data in Excel files directly. We went with this data set because it was easier to update the algorithms whenever MITRE published new data.

To be able to test and train the developed algorithms, we decided on an 80:20 data split. We used X the 80% oldest to train the algorithms, while Y the 20% newest were used to test the models. For the simulation approach as well as the genetic algorithm, we need the relative frequencies of a technique appearing in the software of X. We, therefore, included that information also in our data set together with the date per software.

3.1. Challenges

Regarding the structure of the data, we faced three major challenges. First, the resulting matrix is sparse, and only 2.01% of the entries are ones. The reason for this is that many techniques are used in just a few software. Consequently, we reduced the matrix and only considered techniques used in at least four software, which reduced the number of techniques from 552 to 192 by 64.67%. Due to empty software vectors, we also filtered software using less than two techniques. By taking these steps, we retained 95.7123% of the information (number of ones) but reduced the number of techniques to 35.3261% and eliminated degenerated software. We will use the data set for any other algorithm or analysis containing 192 (before 552) techniques and 449 (before 585) software. Hereafter, the 80:20 split for X and Y is performed.

The second challenge considers the elements of the TTP chain. During an attack, a technique is used by the software to implement a certain tactic. First, we wanted to include the tactics from techniques that were used in our analysis. However, this would be very complex to integrate into our algorithms. This is because one technique can be used in several tactics and may not be assigned to a unique tactic. For our models, we ignored the tactics and focused on the techniques and software.

Lastly, the database has not been updated consistently over the last few years. In 2017, there were 149 software inserted at the same date, which was the start date of the database. In 2018, there were more dates of insertion. However, there were often up to 50 software added on the same date. In more recent times, the database was updated more regularly, corresponding to dates with a single software. Therefore, the newer a data point is, the more precise its time of insertion.

3.2. Limitations

The data set of MITRE ATT&CK has two significant limitations regarding the dates and the size.

Firstly, because the data was uploaded in blocks, the 'creation date' of a certain software might not accurately correspond to its first occurrence. Furthermore, 75% of the software have only five different timestamps because they were uploaded in batches. This is why the data set does not have the accuracy to run a significant time-series approach. The most recent software have relatively accurate timestamps. In the future, it will therefore be possible to run time-based algorithms on the data to focus on the development of attacks over a period of time.

Secondly, the data set is limited and does not include all existing techniques and software used for attacks. It is hence not easy to get good results due to the high technique dimensionality. Even in the reduced matrix with 192 techniques, the ratio towards 449 software data points is low. However, the constantly growing database makes it possible to create more precise models in the future.

4. Prediction Approaches

In this section, we will describe the experimental setups of our analysis methods and prediction models. First, we present how we applied hierarchical clustering and principal component analysis to gain insights into the data's structure. Next, we show how the simulation acts as the baseline model to compare against more complex models such as time series analysis, generative adversarial networks and genetic algorithms.

4.1. Data Exploration

4.1.1. Hierarchical Clustering

We considered different clustering algorithms to gain an insight into the structure of the data. Al-Shaer et al. investigated associations in MITRE ATT&CK adversarial techniques and concluded that hierarchical clustering is the most promising technique to achieve this. We tried to find associations between existing software in order to predict new ones; we had a similar aim but worked with different software instead of techniques.

One of the main advantages of hierarchical clustering is that it is not necessary to specify the number of clusters in advance. Instead, this can be done by cutting the tree at a certain distance afterwards. As Al-Shaer et al. described in more detail, agglomerative hierarchical clustering is most suited for the chosen data. In this Bottom-up model, each software is considered a separate cluster based on the distance matrix; the most similar software gets combined into bigger clusters.

	$S_{2,j} = 0$	$S_{2,j} = 1$	**total**
$S_{1,i} = 0$	n_{00}	n_{01}	$n_{0\bullet}$
$S_{1,i} = 1$	n_{10}	n_{11}	$n_{1\bullet}$
total	$n_{\bullet 0}$	$n_{\bullet 1}$	n

Table 1
Definition of parameters for phi coefficient with the *i*-th component of the first software and the *j*-th component [23]

To calculate the distance matrix, choosing a metric that considers the data set's nature is crucial. We picked the phi coefficient, as it is suitable for binary data. It is an empirical non-parametric correlation measure. The phi correlation between any two software S_1 and S_2 is defined as

$$r_\phi(S_i, S_j) = \frac{n_{11}n_{00} - n_{10}n_{01}}{n_{1\bullet}n_{\bullet}n_{1\bullet}n_{0\bullet}}$$

where n_{00} is the number of matching zeros, n_{10} the number of non-matching entries with a one in place in S_i and a zero in the same place. See Tab. 1. Other types of metrics are not further discussed at this point because they are not relevant for the data set presented.

During the process of hierarchical clustering, a linkage method is required. Based on the recommendation by Al-Shaer et al. we decided on Ward's linkage method. It calculates the distance between larger clusters by computing the sum of the squared distances, divided by the product of the cardinality of the two clusters.

In order to be able to combine the ward's linkage method and the phi coefficient, we extended the agglomerative clustering and customised the algorithm.

The result of this is a dendrogram of all software and their associations. The leaves correspond to the similarity or dissimilarity between different software. 8

After developing the hierarchical clustering tree, the final step is to cut the tree at a certain height to create the final clusters. Based on experiments with different prediction methods, we used different methods to cut the tree. Depending on the algorithm, we either cut the tree at a certain height to perform a balanced cut or cut the tree at a determined height. This algorithm aims to create clusters of a similar size and not cut the tree at one defined height.

4.1.2. Principal Component Analysis

To identify hidden factors inside the data, one can use principal component analysis (PCA). It transforms the original data points to a new orthogonal basis. These basis vectors can be sorted by the ratio of variance they cover and then interpreted as underlying factors. For our analysis, we pursued two different approaches. Firstly, we applied the PCA to the whole data set and plotted the results for one, two and three dimensions (section 5.1).

4.2. Simulation

To benchmark our models, we started implementing a simulation of future software as the simplest possible model. For comparison, the evaluation of all models is done with the F-score (see section 5.1.1).

The simulation is based on a probability distribution of techniques. For that, we use the observed relative frequencies of techniques of the training matrix X.

With these probabilities, we simulated a drawing of 100 different software randomly. We followed two different approaches: In the first one, the algorithm always draws 11 techniques for one software based on the calculated probabilities. In the other approach, we draw a random number of techniques individually for each new software so that the number of techniques is not constant. It is based on a Gaussian distribution with mean and variance equal to these values in the observed X matrix.

4.3. Time Series Analysis

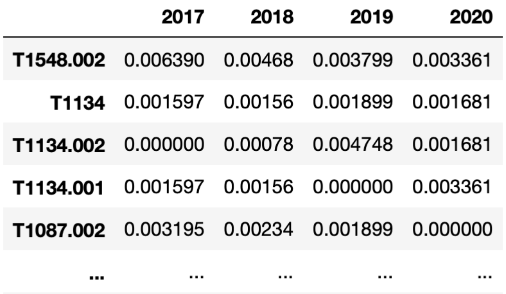
The time-dependent structure of the attack data implies a time series analysis and forecast approach. In this section, we describe the used time-series approach.

First, the data structure of observed software (see section **??**) leads to a vector autoregressive (VAR) approach. One software is represented by a binary vector of techniques with a date. Each technique can be interpreted as a time series itself. This fact is the ”vector” part of the VAR. The auto-regressive nature of this method regresses a technique to itself and all other previous techniques. Therefore, a one-step-ahead VAR(1) approach needs to fit k^2 parameters, where k is the dimension of techniques.

Given that the distribution of timestamps is highly uneven (see section 3.1), we cannot perform a VAR model fitting directly mainly because multiple software have the same timestamp.

Instead, we performed two experiments. First, we did ordinary VAR regression with the newest data with one software per point in time. Second, we aggregated all software per year and analysed them to identify trends.

To conduct the first experiment, we extracted all data points where the timestamp is unique. The new data set X_{short} consists of 42 software ranging from *14-02-2019* to *10-06-2020* with non-uniform distributed dates. This 192 x 42 is the training set for a VAR(n) time series analysis with a lag of n. By using the VAR class of the *statsmodel.tsa* module of the *Scipy* package, we performed six regressions with lags $n \in \{1, \dots, 5, 15\}$.



	2017	2018	2019	2020
T1548.002	0.006390	0.00468	0.003799	0.003361
T1134	0.001597	0.00156	0.001899	0.001681
T1134.002	0.000000	0.00078	0.004748	0.001681
T1134.001	0.001597	0.00156	0.000000	0.003361
T1087.002	0.003195	0.00234	0.001899	0.000000
...	...	...	...	...

Figure 1: Extract from the relative frequencies of a technique per year

Predictions for future software can be made by the forecast method also offered by the imported class. It generates vectors of float values. Therefore, we transformed the prediction to software vectors by setting values above 0.5 to one and the rest to zero. We compare only the first prediction for evaluation because further steps of prediction generate the same software structure (see Figure 10 in appendix A).

The second experiment first involved aggregating all software per year. Next, all values are normalised by dividing them by the number of software in a year. This table of relative frequencies of techniques per year is now our subject of analysis and is shown in fig. 1. A time-series analysis like in the first experiment is not possible due to the small sample size. Three years from 2017 to 2019 and one test year is not enough to get statistically valid results. However, one can see a trend in the frequencies of techniques, as shown in figure 9.

4.4. Genetic Algorithm

Genetic algorithms (GA) are a class of algorithms based on the biological process of evolution [24]. The core idea is that generations of individuals are generated in an iterative process, improving their fitness over time. This can be compared to the Darwinian theory of ”survival of the fittest”. Generating new generations and individuals is achieved through different methods of selection, crossovers, and mutation types. The basic procedure of a GA are the following steps, compare figure 2 [25]:

1. the problem to be optimised is coded in order to be able to map it in binary representation.
2. an initial population of individuals is created and initialised.
3. all individuals of the generation are evaluated using a fitness function.
4. two parents are selected using a selection method.
5. using a crossover method, the binary information of the parents is used to produce offspring.

6. a fixed mutation probability can mutate alleles of the offspring.
7. the newly produced individuals complement, or replace, the old generation.
8. procedure from 4. onwards is iteratively repeated until a fixed termination condition is reached.

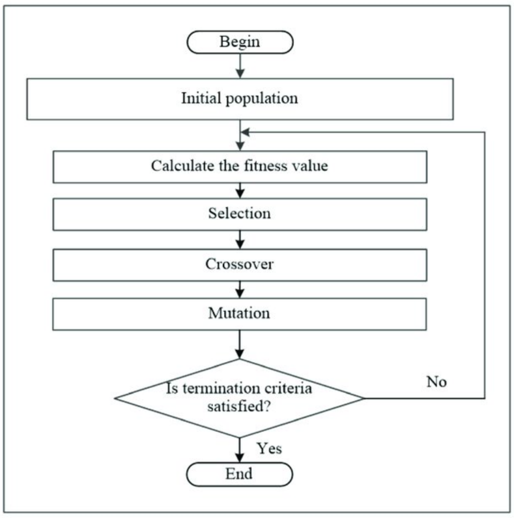

Begin
Initial population
Calculate the fitness value
Selection
Crossover
Mutation
Is termination criteria satisfied?
No
Yes
End

Figure 2: Flowchart of a standard genetic algorithm (GA) [26]

4.4.1. Initialisation

For the GA, we interpret software as an individual while the particular techniques represent the chromosome. Thus, each technique is represented by either a zero or a one, depending on its occurrence in the respective software. This leads to a chromosome consisting of a vector of zeros and ones, whereby our problem itself already provides a binary encoding.

In the initialisation, the starting population size can be set variably. We used 20 individuals as this number leads to the best trade-off between results and run-time behaviour. With larger populations, the complexity and thus also the run-time behaviour of the algorithm became significantly longer, while the outcomes, the F-score, did not improve significantly. With less than 20 generations, however, slightly worse detection results were the consequence. Similar to the simulation, we use the observed mean of ones to generate this number of ones in the chromosome of the starting population randomly.

An essential factor to consider when implementing a GA is the trade-off between exploration and exploitation. During exploration, the algorithm tries to cover the search space as well and as completely as possible. It searches for new solutions in new regions to eventually find not just any but the best possible local optimum, ideally one that is close to the global optimum. Exploitation means that already existing solutions are used and refined to improve their fitness, i.e., reaching a local optimum. Crossover and mutation are both methods of exploring the problem space, whereas selection is used to exploit the 'good' genetic material in the current set of individuals.

However, these are not two separate and diverse concepts. Crossover and mutation both have effects that work towards both exploitation, and exploration [27]. For this reason, it is crucial to test the interaction of the various methods in different combinations and with different parameters in order to achieve the best possible trade-off between exploitation and exploration.

4.4.2. Fitness function

The calculation of fitness is the most critical component of the GA, as the improvement of individuals, and thus the solution, is based on it. Our fitness function consists of three terms, probability term, correlation term and adjustment term, based on which the fitness of each individual is calculated. The combination of the three terms has the consequence that in the algorithm's search space, not only one global optimum exists, but several local ones.

Probability term

Since some techniques occur more frequently in software than others, we want to consider these probabilities in the fitness calculation by implementing a probability term *PT*. The probability of each technique is calculated from the MITRE ATT&CK database, as seen in section 4.2. We thus obtain a probability vector $q \in \mathbb{R}^n$ with the probabilities of all n techniques. The calculation involves multiplying each individual in the respective population $P \in \mathbb{R}^{n \times m}$, where m are the generated software with n potential techniques with the vector q.

Due to the binary structure of the individuals, a vector $PT \in \mathbb{R}^m$ is obtained in whose n rows are the summed probabilities of each predicted software.

$$\mathrm{PT} = q \cdot P \in \mathbb{R}^m \tag{1}$$

Correlation term

Some techniques are more likely to appear together or, conversely, appear separately. To include this correlation behaviour into the model, we added a correlation term *CT*. We start with a given generated software vector $p \in \mathbb{R}^n$ with n techniques and the technique correlation matrix $C \in \mathbb{R}^{n \times n}$.

First, we subtract the value one from the main diagonal to remove self-correlation. Otherwise, this term would increase the fitness function without any correlation. The unit matrix E represents this.

$$\hat{C} = C - E \tag{2}$$

Now, we multiply p by $\hat{C}$ to identify the correlation value per existing "1" in the generated software. This resulting vector

$$a = \hat{C} \cdot p \in \mathbb{R}^n \tag{3}$$

is now multiplied by the generated software. The sum of correlations k for the used techniques is the result.

$$k = p^T \cdot a \in \mathbb{R} \tag{4}$$

Lastly, this metric is normalised

$$\hat{k} = \frac{k}{|p|}. \tag{5}$$

To do this computation for m populations in parallel, we extended the ideas mentioned above. Now, we start with

$$A = \hat{C} \cdot P \in \mathbb{R}^{n \times m} \tag{6}$$

where $P \in \mathbb{R}^{n \times m}$ is the population matrix of m generated software with n techniques.

Masking the correlation value matrix A again with P results in a $m \times m$ matrix because each p_i vector of P and each a_j vector of A are summed together.

$$K = P^T \cdot A \in \mathbb{R}^{m \times m} \tag{7}$$

However, this is not our goal. We are only interested in $p_i \cdot a_i$ elements. Finally, we get the correlation term by only use the diagonal elements $\mathrm{CT} = \mathrm{diag}(K)$.

Adjustment term

In order to predict realistic software, another term had to be included in the fitness function. For this purpose, the mean of the techniques in each software of the MITRE ATTA&CK data set was calculated. On average, 12 different techniques were used per software. By considering the mean, we want to ensure that the predicted software does not deviate too far from the 12 techniques to avoid predicting degenerated software. First, we calculated the occurrence $o \in \mathbb{R}^n$ of ones for each predicted software from $P \in \mathbb{R}^{n \times m}$ in parallel.

$$o = P \cdot \mathbb{1} \in \mathbb{R}^n \tag{8}$$

where P was multiplied by $\mathbb{1}$, a vector of ones, so that the occurrence vector o represents the sum of ones and thus the techniques used per software.

In the next step, the difference between the vector o and the mean value $\mu = 12$ is calculated by subtracting μ from all the values in o, which leads to the difference term

$$\mathrm{DT} = o - \mu \cdot \mathbb{1} \in \mathbb{R}^n\,. \tag{9}$$

Now a branch is made depending on whether the deviation from the mean value is downward or upward. On the one side, if the predicted software has less than μ techniques, then the difference term DT is negative. On the other side, it is positive.

The first sub-term considers positive differences. Here, all deviations are squared and thus become positive. Then the vector is multiplied by -1 or +1 depending on the original sign of the value.

Finally the negative values are cancelled out and the remaining positive squared differences are divided by a penalty factor PF, leading to the first sub-term *PST.*

$$\mathrm{PST} = \frac{d^2}{\mathrm{PF}} \in \mathbb{R}^n \tag{10}$$

The second sub-term *BST* considers the negative differences, i.e., predicted software uses less than μ techniques. The deviations are then each divided by a bonus factor BF.

$$\mathrm{BST} = \frac{d}{\mathrm{BF}} \in \mathbb{R}^n \tag{11}$$

Since the differences are negative and the adjustment term *AT* is subtracted in the final fitness function, the bonus has a positive effect on fitness. The distinction between positive and negative differences is necessary because otherwise, the algorithm tends to include more than 12 techniques in the predicted software. This behaviour happens because it would increase the fitness scores since they are partly based on the occurrence probabilities, and thus, more techniques used lead to higher fitness scores.

The final adjustment function is now composed of the sum of the two terms *PST* and *BST.*

$$\mathrm{AT} = \mathrm{PST} + \mathrm{BST} \in \mathbb{R}^n \tag{12}$$

The entire fitness function *FT* consists of the three main terms described above. The correlation term and the probability term are summed up and are included positively, while the adjustment term is included negatively by being subtracted.

$$\mathrm{FT} = (\lambda \cdot \mathrm{PT} + (1 - \lambda) \cdot \mathrm{CT}) - \mathrm{AT}) \tag{13}$$

Besides the BF and the PF, there is another degree of freedom. The factor λ influences the degree to which the correlation and the relative probabilities of the techniques are included in the fitness function.

4.4.3. Selection

The selection function selects from the set of individuals those that should be used to generate new individuals. There are different selection methods, and we tested several like roulette wheel selection and a simple tournament selection method. The implemented tournament selection is a straightforward method of selection. It involves randomly selecting two individuals from the current population and comparing their fitness scores. The individual with the higher score wins the tournament and is included as a child in the new generation to be crossed using the crossover methods described in 4.4.4. This procedure is repeated until an entirely new child generation has been generated. Since we did not expect good results with this method, we implemented a different, more complex selection method.

The roulette wheel selection method considers the relative fitness scores of each individual, which have to be a positive value. Due to the architecture of our fitness function, fitness scores might be negative, owing to which we had to adapt these scores first. We subtract the smallest fitness value in the population from all other fitness values in the respective population. This results in the smallest fitness value becoming zero and all other values correspondingly non-negative. The resulting positive fitness values are then used for roulette wheel selection. This selection method is a fitness proportional selection method, where the individual crossover probability is calculated based on the individual fitness divided by the sum of the fitness of the whole population. The fitness values are normalised so that the sum of the resulting fitness values is one.

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_i} \tag{14}$$

where p_i is the selection probability of an individual, f_i the respective fitness score and $\sum_{j=1}^{N} f_i$ the sum of all fitness in the population.

This selection method can be thought of as a roulette wheel, where each individual takes up an area on the wheel depending on their fitness. The probability of selecting a potential mate depends on the individuals' fitness relative to the rest of the population. Our implementation allows us to individually specify the number of individuals that should be selected for crossover.

4.4.4. Crossover

The crossover operator is the implementation of recombination in the GA. Pairs of individuals (parents) are crossed by exchanging segments of the respective bit strings between the two parent individuals. This creates new individuals (children) based on the genetic makeup of the two parents. The number of crossover points in the chromosome is usually one or two, implemented using one-point crossover and two-point crossover [28]. However, research has shown that often a more significant number of crossover points can be beneficial [29] [30]. For the one-point crossover method, two selected individuals are passed and then cut at a random location in the chromosome. Two new offspring are then generated by rejoining the two parent individuals at their intersections. Correspondingly, in the two-point crossover method, the passed individuals are cut at two points and then rejoined by combining the respective regions of the parents to create two new offspring; compare fig. 3.

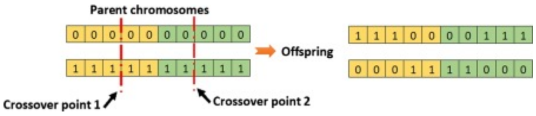


Figure 3: Visualisation of the two point crossover operator [31]

Another suitable operator, the uniform crossover operator, produces on average $(L/2)$ crossovers on chromosome strings of length L. We implemented all three crossover operators to test whether the results differ significantly between methods. The uniform crossover function is passed two parent individuals $A^{(t)}$ and $B^{(t)}$, determined by the selection operator in the previous step, as well as an exchange probability p_s. The function returns two generated children C^{t+1} and D^{t+1} for the next generation of individuals $t + 1$ the exchange of bits is calculated and performed separately for each position of the chromosome. First, a random number between 0 and 1 is drawn, resulting in u. If this number is less than or equal to the exchange probability p_s, the respective bit is exchanged [32].

$$u \leq \ p_s \tag{15}$$

$$c_i^{t+1} = b_i^t \tag{16}$$

$$d_i^{t+1} = a_i^t \tag{17}$$

This process is repeated up to the length L of the parent chromosome. If $u > p_s$ the bits of the parent individuals are copied to the children without flipping them. We used the default value of 0.5 for the exchange probability p_s. Deviations from this value did not yield significant improvements in the context of our problem's structure and data quality. An adjustment of the exchange probabilities depending on whether the respective bit is a zero or a one is to be discussed. Since the target fitness function tends to favour software with many ones, it could be advantageous to reduce the probability of a 0 to 1 swap compared to a 1 to 0 swap.

4.4.5. Mutation

The mutation function intends to flip bits and thus prevent particular parts of the chromosome from being identical in individuals. This is also to prevent the search for a solution only in a subspace of the original search space. Therefore, the mutation method increases the exploration of the GA. The probability that a bit mutates is set by a parameter p_m and is usually low in the range of 0.001 - 0.01.

In our implementation, an array with the shape of the current population is randomly filled with float numbers between 0.0 and 1.0. An element-wise comparison is then performed, and wherever the values of the randomly generated numbers are below the probability p_m, the corresponding bit is inverted. In the end, a mutated population is returned.

4.4.6. Main algorithm

In the main algorithm, the GA is executed, parameters are set, and the respective selection, crossover, fitness and mutation functions are called, compare 2. Furthermore, the best fitness score in each generation is stored and the course of the GA is plotted.

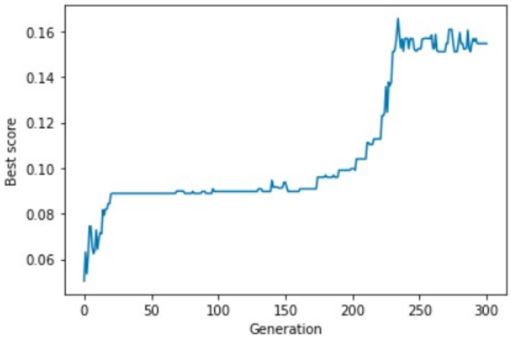


Figure 4: Evolution of the fitness scores in different generations of the GA

After initialising the start population with the parameter *population_size*, the main script executes the GA up to a predefined number of generations, represented by *maximum_generation*. Another essential part of the main algorithm is the choice of the replacement strategy. This strategy defines how newly created individuals are selected or what proportion of the parents should be replaced by the children. The choice is crucial because, in addition to the crossover, mutation and selection functions, it can improve the balance between exploration and exploitation of the algorithm. This, in turn, significantly influences the search performance of the GA, as it influences the extent to which the search space is explored or exploited. However, there is no general best replacement strategy, as the choice depends on many problem-specific factors [33]. Generally, a distinction is made between generational (non-overlapping) GAs and steady-state (overlapping) GAs. In generational GAs, the entire parent generation is replaced by a completely new offspring generation. In contrast, in steady-state GAs the new generation consists of parts of the parent generation and the newly created offspring.

We decided to use a general replacement strategy, where the offspring generation replaces the parent generation. The disadvantage of this method is that possibly good individuals from the previous generation are lost, and thus the average fitness of the total population decreases. However, this also means that the chance of finding less good individuals (local optimum) is lower. In our case, the population's average fitness is secondary, and avoiding being 'stuck' in a local optimum is more critical.

In our implementation of the GA, a loop is used to generate two new individuals at a time using uniform crossover and add them to a new generation. The candidates for the crossing and creating the child generation are selected from the parent generation through roulette wheel selection. As soon as the newly generated child generation reaches the size of the parent generation, it is replaced. Subsequently, a mutation is applied to the newly created individuals.

As the data basis for the GA, we used X to train the algorithm, while Y was used to evaluate the predicted software, stated in section **??**.

4.4.7. Optimising parameters

In the previous sections, we defined the parameters BF, PF and p_m. They significantly influence the behaviour of the GA and the trade-off between exploration and exploitation. These hyper-parameters can be optimised regarding better F-scores (see section 5.1.1). A global optimisation routine like simulated annealing [34] is utilised to produce the results stated in section 5.1.2. While running the optimiser, we fixed the seed for the random number generators to ensure reproducibility. Moreover, we selected the starting values manually and gave them boundaries to restrict search space. The target function of the optimiser is the negative mean of 30 F_1 scores of generated software by the GA.

For comparison, we also tested a local optimisation routine. However, this L-BFGS algorithm [35] converged to the start values, and were therefore stuck in a local minimum.

4.5. Generative Adversarial Network

We aim to generate new software out of observed ones. Generative Adversarial Networks (GAN) are a specific type of neural network (NN) and are trained to produce new instances of objects similar to those they were trained on. This is reached by two neural networks (NN), that compete against each other. The first generates new samples, in our case software, out of noise input. The other discovers fake data [36]. After training, the generated software looks like the observed software.

To create a GAN, one need to perform these steps

1. set up the GAN by defining the generator and discriminator,
2. provide samples of real software vectors,
3. create fake software,
4. train the GAN,

which are shown in detail in the following paragraphs.

Defining the GAN model

Due to the small given data set and the exploitative intention, we start with simple models. The generator aims to generate software the discriminator is not able to differentiate from real software. The NN has l input neurons, where l is the dimension of the latent space. Here we set $l = 50$. There are 150 hidden neurons in the middle layer with relu activation function and 192 neurons in the output layer with linear activation representing all techniques. The discriminator tries to filter out fake software generated by the generator. It is a NN with 192 input neurons, 25 hidden neurons with relu activation and one output neuron with sigmoid activation. Both networks are using the adam optimiser.

For training, the GAN is provided by real software samples drawn from the observed data matrix X.

Moreover, the generator creates new software out of a multivariate Gaussian noise vector with dimension l.

Finally, we train the GAN by training the discriminator on the real samples for this batch. Next, the generator creates new software, which the discriminator then evaluates. Now, the generator is trained to fool the discriminator. In this setup, we trained batch-wise with the size of 64 and 2000 iterations.

5. Discussion

5.1. Evaluation

5.1.1. F-score

We used the F-score to measure the performance of the predictions. It is the harmonic mean of precision and recall scores and therefore combines them. It is defined as [37]

$$\mathrm{F} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}. \tag{18}$$

The F-score thus combines the following two goals:

1. It evaluates whether the predicted techniques are all relevant, and how many of our predicted techniques were used in a single software of the test set. (Precision)
2. It assesses, the number of used techniques successfully predicted inside a single software of the test set (Recall).

For the simulation and all other methods, we compared every predicted software pair-wise to all the software from the test set. Only the best F-score is returned because the aim is to get as close (according to the F-score) to one of the software in the test set as possible.

To gain robust statistical statements, not only one software is generated per method, but many. For each of the more than 30 predictions per method, we calculate the F-score as stated above. This procedure ensures comparability between the different prediction approaches.

5.1.2. Results

Data exploration After cleaning the data by removing degenerate software and techniques, the dimension is reduced by 64.7% from 552 × 585 to 192 × 449. Despite that, we kept 95.7% of the information in the data. This shows that the matrix was strongly sparse in the beginning. By reducing the matrix, we kept the information but decreased its complexity significantly.

Next, hierarchical clustering sorted the software based on the similarity or dissimilarity to one another. The resulting dendrogram in fig. 8 shows clear clusters. This could enable prediction approaches within clusters in the future.

After that, we used the data of the clusters to apply PCA. In this case, shown in fig. 5, it was not possible to recognise a structure of temporal developments, and the data points were distributed randomly. This is the reason why we did not pursue this approach any further in our research. In the future, when data with different time stamps are available, it will most likely become possible to gain deeper insights with this approach.

In addition, fig. 7 shows a transformation and plot of all software into three dimensions done by PCA. Here, one can see a concentration of software at one point. Even the explained variance per new basis vector implies the lack of principal components. The most important new axis captures 21.4% of the variance. It is followed by 3.6% and 3.2%. This low level further gradually decreases. In short, we can not identify underlying factors.

Predictions To evaluate complex prediction methods, we first needed to set a baseline. It is, in our case, the simulation with a resulting average F-score of 0.42 based on 100 simulated software.

Next, we look at the VAR(n) time-series model. Table 2 presents the results for $n \in \{1, \dots, 5, 15\}$ lags. Here, the

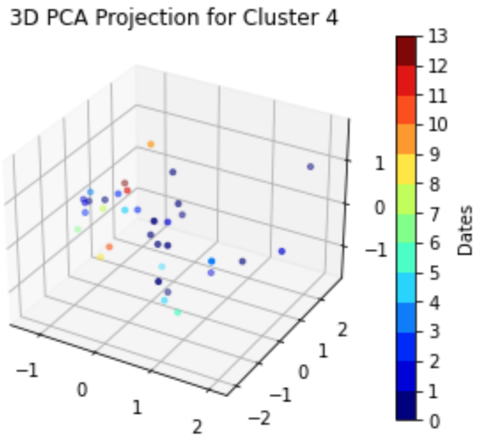


Figure 5: Software clusters with 3 dimensions after performing PCA, colored with reference to the respective timestamps

Lag *n*	**1**	**2**	**3**	**4**	**5**	**15**
F-score avg	0.46	0.52	0.55	0.52	0.61	0.50
F-score min	0.27	0.0	0.29	0.0	0.40	0.0
F-score max	0.73	0.74	0.71	0.74	0.83	0.80
F-score first	0.53	0.58	0.67	0.50	0.62	0.67
# Techniques	20	10	6	4	7	5
Index of first	16	16	58	57	58	48

Table 2
Comparison of results of VAR(n) first prediction for different lag sizes. The F-score is calculated by using 30 samples. For contrast, here are shown the average, minimum value, and maximum of the score, as well as the number of techniques used by this software and that of the best corresponding software in the test set

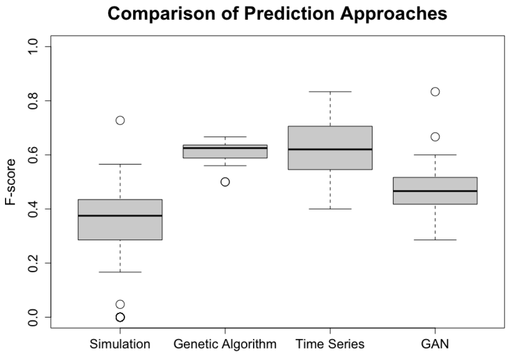


Figure 6: Boxplot showing the comparison in terms of the F-score results of different prediction approaches. We used the simulation as a baseline and ran the Time Series with VAR(5).

Var(*n*) predicts 30 forecasts of software. The mentioned table shows the corresponding average, maximum and minimum F-score. The average ranges from 0.46 to 0.61, peeking at a lag of five.

In addition, for the one-step-ahead forecast, the F-score and the index of the hit entry in the test set are shown. For all lags, the first F-score was higher than average. Therefore, only the first forecast could be used as a heuristic.

The GAN performed with an average F-score of 0.46, slightly higher than the simulation. Since we had 3500 parameters and 350 data points, the risk of over-fitting the model was high. Nevertheless, it would be interesting to review the model again in future when more data is available.

The genetic algorithm performed best of all the methods explored. For evaluation, we executed the GA 50 times, with a maximum generation size of 450 and a population size of 40 individuals each time. Here we used the optimised parameters (see **??**), although these differ depending on the crossover and selection methods used.

In the resulting 50 final generations each with 40 predicted software, we selected the best one, i.e. the one with the highest fitness score. A selection of e.g. the five most successfully predicted software led to equivalent results since the five best software from each run of the GA hardly differed in their binary structure.

We used the reduced matrix with 192 techniques in the GA. With the help of the tournament selection and uniform crossover operator, we achieved a mean F-score of 0.63.

With the roulette selection, against our expectations, only an F-score of 0.58 was achieved. In addition, the run-time performance of the roulette selection was worse. Fortunately, the variance of the individual F-scores was relatively low, with scores varying only between F-values of 0.52 - 0.8. We used uniform crossover and one-point crossover for predictions with the GA, as the results with two-point crossover was — with an average F-score of 0.53 — significantly worse. However, the two-point crossover produced a larger selection of different software, while the predicted software in the other two methods was often identical in several runs of the GA.

All our prediction approaches predicted different software from the test set. It therefore resulted in no single identified software, but somewhat various developments that must be considered in a differentiated manner. This result also reflects the reality, as software attacks are diverse and cannot be reduced to one software.

5.2. Future research

The dendrogram resulting from the hierarchical clustering can now be used in the next step to work further at a contextual level. One can investigate how to categorise future software or analyse clusters for trends for which one must have a deep understanding of the different software.

There were challenges due to the nature of the data set. The difficulties in predicting new software in a time series approach with only a few time stamps were shown in this work. Additionally, the model is probably over-fitted due to $k^2 = 192^2$ parameters but only 42 data points for a VAR(1) model and.

In the future, all of the presented models will perform better with more data to train their systems. Therefore, the search for additional sources to have a more accurate and broad data set will be crucial for further approaches to software prediction. The MITRE ATT&CK database will grow and enables new and more complex models.

Additionally, there are various options to optimise the genetic algorithm even further. One idea is to consider other factors in the fitness function like a friends-measure. Here, we account for higher-order technique connections.

In conclusion, we expect better results in the future due to more available data. Also, further adjustments could be made to the GA, especially to the selection and crossover function, or new methods could be developed that consider even more specifically the binary structure of the problem. Another way to improve the results is to optimise the parameters further, as even small changes can achieve significant differences in the precision of the predictions.

6. Conclusion

At the beginning, we identified two questions, which now can be answered. First, we wanted to know whether there are patterns or trends in software attacks that can be used for medium to long term predictions. Through hierarchical clustering, we can confirm that clusters and, therefore, patterns exist. With this result, experts can use this to classify upcoming software or discover concrete trends. We analysed the clusters with PCA but could not observe any principal components or underlying factors.

The second question we raised concerned the performance of different algorithms to predict future malware. We followed different approaches and compared the performance with the F-score. A statistical simulation, based on the relative frequencies of observed techniques, serves as our baseline. More complex models such as the genetic algorithm or the time series provide meaningful predictions whose F-score are higher than these of the simulation. As the data grows, one can build on these findings and optimise predictions. In contrast, GANs did not lead to significant predictions for now due to the lack of data.

Taking everything into account, we can say that there are relevant trends and patterns which can be used to predict future malware with approaches like time series and genetic algorithms.

References

[1] P. Boden, The emerging era of cyber defense and cybercrime, 2016. URL: https://www.microsoft.com/security/blog/2016/01/27/the-emerging-era-of-cyber-defense-and-cybercrime/.

[2] D. Freeze, Cybercrime to cost the world $10.5 trillion annually by 2025, 2021. URL: https://cybersecurityventures.com/cybercrime-damages-6-trillion-by-2021/.

[3] B. Oberzaucher, 2019. URL: https://www.andritz.com/spectrum-en/latest-issues/issue-39/digitalization-as-a-megatrend.

[4] R. Contu, Forecast analysis: Information security, worldwide, 2q18 update, 2018. URL: https://www.gartner.com/en/documents/3889055.

[5] V. Mavroeidis, S. Bromander, Cyber threat intelligence model: an evaluation of taxonomies, sharing standards, and ontologies within cyber threat intelligence, in: 2017 European Intelligence and Security Informatics Conference (EISIC), IEEE, 2017, pp. 91–98.

[6] A. Frost, P. Sullivan, 2017 global information security workforce study, 2017.

[7] S. Ijaz, F. A. Hashmi, S. Asghar, M. Alam, Vector based genetic algorithm to optimize predictive analysis in network security, Applied Intelligence 48 (2018) 1086–1096.

[8] S. J. Yang, H. Du, J. Holsopple, M. Sudit, Attack projection, Cyber Defense and Situational Awareness (2014) 239–261.

[9] M. Husák, J. Komárková, E. Bou-Harb, P. Čeleda, Survey of attack projection, prediction, and forecasting in cyber security, IEEE Communications Surveys & Tutorials 21 (2018) 640–660.

[10] M. R. Endsley, Situation awareness global assessment technique (sagat), in: Proceedings of the IEEE 1988 national aerospace and electronics conference, IEEE, 1988, pp. 789–795.

[11] M. R. Endsley, Toward a theory of situation awareness in dynamic systems, in: Situational awareness, Routledge, 2017, pp. 9–42.

[12] A. Elitzur, R. Puzis, P. Zilberman, Attack hypothesis generation, in: 2019 European Intelligence and Security Informatics Conference (EISIC), IEEE, 2019, pp. 40–47.

[13] D. Chismon, M. Ruks, Threat intelligence: Collecting, analysing, evaluating, Technical Report, MWR InfoSecurity, 2015.

[14] E. M. Hutchins, M. J. Cloppert, R. M. Amin,

Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains, Leading Issues in Information Warfare & Security Research 1 (2011) 80.

[15] Z. Zhan, M. Xu, S. Xu, Characterizing honeypot-captured cyber attacks: Statistical framework and case study, IEEE Transactions on Information Forensics and Security 8 (2013) 1775–1789.

[16] D. S. Fava, S. R. Byers, S. J. Yang, Projecting cyberattacks through variable-length markov models, IEEE Transactions on Information Forensics and Security 3 (2008) 359–369.

[17] X. Qin, W. Lee, Attack plan recognition and prediction using causal networks, in: 20th Annual Computer Security Applications Conference, IEEE, 2004, pp. 370–379.

[18] S. Zhang, X. Ou, D. Caragea, Predicting cyber risks through national vulnerability database, Information Security Journal: A Global Perspective 24 (2015) 194–206.

[19] J. A. Ozment, Vulnerability discovery & software security, Ph.D. thesis, University of Cambridge, 2007.

[20] O. H. Alhazmi, Y. K. Malaiya, Prediction capabilities of vulnerability discovery models, in: RAMS'06. Annual Reliability and Maintainability Symposium, 2006., IEEE, 2006, pp. 86–91.

[21] S. Abraham, S. Nair, A predictive framework for cyber security analytics using attack graphs, arXiv preprint arXiv:1502.01240 (2015).

[22] V. H. Nguyen, L. M. S. Tran, Predicting vulnerable software components with dependency graphs, in: Proceedings of the 6th International Workshop on Security Measurements and Metrics, 2010, pp. 1–8.

[23] R. Al-Shaer, J. M. Spring, E. Christou, Learning the associations of mitre att & ck adversarial techniques, in: 2020 IEEE Conference on Communications and Network Security (CNS), IEEE, 2020, pp. 1–9.

[24] D. Whitley, A genetic algorithm tutorial, Statistics and computing 4 (1994) 65–85.

[25] M. Mitchell, An introduction to genetic algorithms, MIT press, 1998.

[26] K. Höschel, V. Lakshminarayanan, Genetic algorithms for lens design: a review, Journal of Optics 48 (2019) 134–144.

[27] A. E. Eiben, C. A. Schippers, On evolutionary exploration and exploitation, Fundamenta Informaticae 35 (1998) 35–50.

[28] K. A. De Jong, W. M. Spears, An analysis of the interacting roles of population size and crossover in genetic algorithms, in: International Conference on Parallel Problem Solving from Nature, Springer, 1990, pp. 38–47.

[29] G. Syswerda, Uniform crossover in genetic algorithms, in: Proceedings of the third international conference on Genetic algorithms, Morgan Kaufmann Publishers, 1989, pp. 2–9.

[30] K. De Jong, W. Spears, On the virtues of parameterized uniform crossover, in: Proceedings of the 4th international conference on genetic algorithms, Morgan Kaufmann Publishers, 1991, pp. 230–236.

[31] Z. C. Dagdia, M. Mirchev, When evolutionary computing meets astro-and geoinformatics, in: Knowledge Discovery in Big Data from Astronomy and Earth Observation, Elsevier, 2020, pp. 283–306.

[32] T. Dominik-Gwiazda, Genetic algorithms reference, volume i, crossover for single-objective numerical optimization problems, 2006.

[33] Y. Wu, J. Liu, C. Peng, A new replacement strategy for genetic algorithm and computational experiments, in: 2014 International Symposium on Computer, Consumer and Control, IEEE, 2014, pp. 733–736.

[34] C. Tsallis, D. A. Stariolo, Generalized simulated annealing, Physica A: Statistical Mechanics and its Applications 233 (1996) 395–406.

[35] R. Pytlak, Conjugate gradient algorithms in nonconvex optimization, volume 89, Springer Science & Business Media, 2008.

[36] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial nets, Advances in neural information processing systems 27 (2014).

[37] D. M. Powers, Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation, arXiv preprint arXiv:2010.16061 (2020).

A. Appendix

https://github.com/ElisaGoebel/Attack-Forecast-and-Prediction

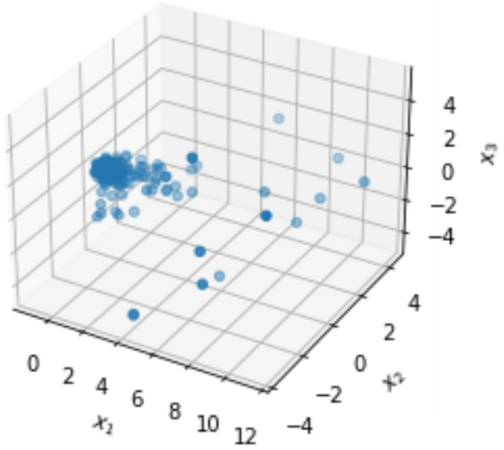


Figure 7: Plot of all software after transformation into 3 dimensions with PCA.

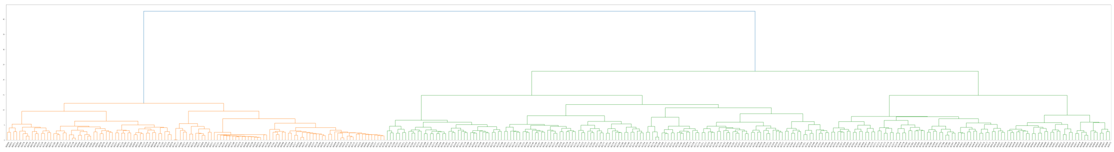
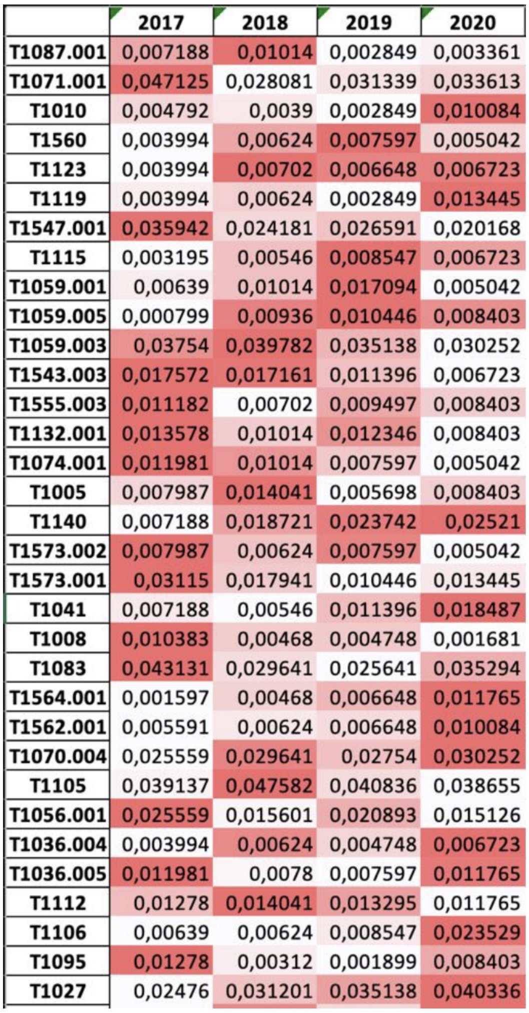


Figure 8: Plot of agglomerative hierarchical clustering of software attacks



	2017	2018	2019	2020
T1087.001	0,007188	0,01014	0,002849	0,003361
T1071.001	0,047125	0,028081	0,031339	0,033613
T1010	0,004792	0,0039	0,002849	0,010084
T1560	0,003994	0,00624	0,007597	0,005042
T1123	0,003994	0,00702	0,006648	0,006723
T1119	0,003994	0,00624	0,002849	0,013445
T1547.001	0,035942	0,024181	0,026591	0,020168
T1115	0,003195	0,00546	0,008547	0,006723
T1059.001	0,00639	0,01014	0,017094	0,005042
T1059.005	0,000799	0,00936	0,010446	0,008403
T1059.003	0,03754	0,039782	0,035138	0,030252
T1543.003	0,017572	0,017161	0,011396	0,006723
T1555.003	0,011182	0,00702	0,009497	0,008403
T1132.001	0,013578	0,01014	0,012346	0,008403
T1074.001	0,011981	0,01014	0,007597	0,005042
T1005	0,007987	0,014041	0,005698	0,008403
T1140	0,007188	0,018721	0,023742	0,02521
T1573.002	0,007987	0,00624	0,007597	0,005042
T1573.001	0,03115	0,017941	0,010446	0,013445
T1041	0,007188	0,00546	0,011396	0,018487
T1008	0,010383	0,00468	0,004748	0,001681
T1083	0,043131	0,029641	0,025641	0,035294
T1564.001	0,001597	0,00468	0,006648	0,011765
T1562.001	0,005591	0,00624	0,006648	0,010084
T1070.004	0,025559	0,029641	0,02754	0,030252
T1105	0,039137	0,047582	0,040836	0,038655
T1056.001	0,025559	0,015601	0,020893	0,015126
T1036.004	0,003994	0,00624	0,004748	0,006723
T1036.005	0,011981	0,0078	0,007597	0,011765
T1112	0,01278	0,014041	0,013295	0,011765
T1106	0,00639	0,00624	0,008547	0,023529
T1095	0,01278	0,00312	0,001899	0,008403
T1027	0,02476	0,031201	0,035138	0,040336

Figure 9: Extract from the relative frequencies of a technique per year, dyed. As darker the red as higher the probability of this technique in that corresponding year.

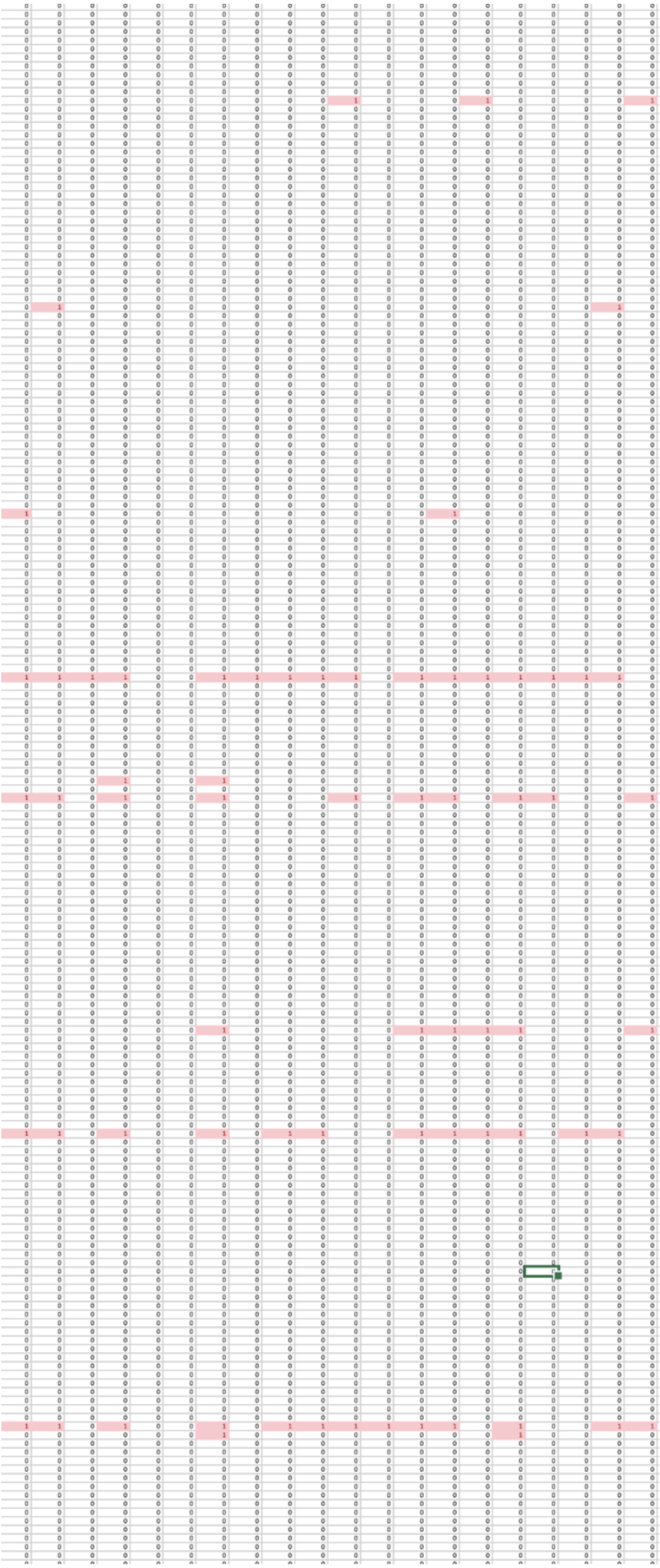


Figure 10: Sample of 20 generated software by prediction from VAR(3) model. Ones marked with red.

Erklärung

Ich versichere wahrheitsgemäß, die Arbeit selbstständig verfasst, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde sowie die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet zu haben.

Karlsruhe, July 16, 2021

E. Goebel

Tobias Budig Elisabeth Goebel Jurek Muff Tessa Fischer