Attack Forecast and Prediction

Florian Klaus Kaiser[a], Tobias Budig[a], Elisabeth Goebel[a], Tessa Fischer[a], Jurek Muff[a], Marcus Wiens[a] and Frank Schultmann[a,b]

[a] *Karlsruhe Institute of Technology (KIT), Kaiserstraße 12, 76131 Karlsruhe, Germany*
[b] *Adelaide Business School, University of Adelaide, 12/10 Pulteney Street, Adelaide, Australia*

Abstract

Cyber-security has emerged as one of the most pressing issues for society with actors trying to use offensive capabilities and those who try to leverage on defensive capabilities to secure their assets or knowledge. However, in cyber-space attackers oftentimes have a significant first mover advantage leading to a dynamic cat and mouse game with defenders. Cyber Threat Intelligence (CTI) on past attacks bears potentials that can be used by means of predictive analytics to minimize the attackers first mover advantage. Yet, attack prediction is not an established means and automation levels are low.

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 three modules representing different analytical procedures which are clustering, time series analysis, and genetic algorithms. *AFP* identifies trends in the usage of attack techniques and crafts forecasts and predictions on future malware and the attack techniques used. We rely on time sorting to generate subgraphs of MITRE ATT&CK and evaluate the accuracy of predictions generated by *AFP* based on these. Results of an experiment performed on the basis of 493 different malware, validate the utility of using *AFP* for attack prediction. *AFP* reaches for each module an F-score which is higher than an extrapolation of observed probabilities (baseline) with an F-score of up to 0.83 for a single module. It can hence be considered an effective means for predicting future attack patterns and help security professionals with preparing for future attacks.

Keywords

Attack Prediction, Cyber Threat Intelligence, Genetic Algorithms

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 that the damage related to worldwide cyber-crime was $3 trillion in 2015. This number is expected to increase by 15% every year, reaching $10.5 trillion annually by 2025 [1] [2]. Consistently, individuals, businesses and governments are becoming increasingly concerned about the costs and threats presented by cyber-crime, espionage, and cyber-warfare [3]. It is hence expected that the worldwide information security market will reach $170.4 billion in 2022 [4].

C&ESAR'21: Computer Electronics Security Application Rendezvous, November 16-17, 2021, Rennes, France
florian-klaus.kaiser@kit.edu (F. K. Kaiser); tobias.budig@student.kit.edu (T. Budig); elisabeth.goebel@student.kit.edu (E. Goebel); tessa.fischer@student.kit.edu (T. Fischer); jurek.muff@student.kit.edu (J. Muff); marcus.wiens@kit.edu (M. Wiens); frank.schultmann@kit.edu (F. Schultmann)

Attackers' strategies quickly develop and are subject to dynamic innovations. Consequently, 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 an arms race between security professionals, and threat actors." [5] Thereby the attacker seems 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 analyzing cyber-attacks that happened in the past and predicting patterns of attacks for the future means an improvement of cyber-security in its ability to enhance one's position in the arms race between adversaries and defenders.

Therefore, predictive analysis could lead to an advantage for organisations to properly allocate their scarce defence resources. Although predicting attacks is not a new procedure, automating attack forecasting and predictions were options hardly used in the past. Rather, attack predictions were largely based on subjective perceptions of experienced experts from the cyber-threat landscape. Yet, experienced experts are rare and their time is even scarcer.

Automation of attack forecasting and prediction would substantially decrease attackers first mover advantage by decreasing biases in predictions and minimise experts' time spending on generating forecasts.

1.2. Problem statement

As described, predicting future malware and their functioning is of especial interest. We thereby consider a malware as a unique combination of attack techniques (software vector). Technically predicting future malware is the prediction of (co-)occurrences of techniques in future malware respectively predicting new malware nodes within the cyber-attack knowledge graph. These predictions can increase cyber-security maturity. Currently, predictions mainly focus on short term. However, medium respectively long term predictions are of high importance for security professionals. Consistently, research and development of defensive measures take a lot of time, whereas better prediction can reduce the time lead over attackers. Note that security spending is an investment in the future security of a company and should hence follow the dynamics of attacks guaranteeing the feasibility of defensive means. Yet, as medium and long term predictions are scarce, research and development of defensive measures are subject to high risk of obsolescence.

CTI (e.g. as it is provided by MITRE ATT&CK)[1] contains information with the potential of making predictions more accurate representing a great opportunity for ensuring cyber-security. However, currently, predictions on future developments of attacks are rare and oftentimes are largely based on experiences of some analysts rather than CTI. This low level of automation

[1]For an in depth discussion on CTI and the relation to MITRE ATT&CK refer to section 2

regarding the prediction of new attacks causes many problems for cyber-security. First, experts were distracted from their operations and generating predictions means extra workload for them. Second, even the best and experienced experts perceive cyber-attacks only from a limited and in this way subjective perspective. This is, human crafted predictions are prone to biases.

1.3. Research question and course of the study

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

i Are there any patterns or trends in the MITRE ATT&CK that can be used for crafting medium to long term predictions on the threat landscape?
ii How do different algorithms perform to predict future malware in a medium to long term?

To address these questions, we collect data about observed (”historic”) malware respectively CTI by scraping information about software and their used techniques from the MITRE ATT&CK database.

In a next step, we conduct a simple statistical analysis to identify the probability of techniques used by a software and the distribution of the number of techniques per software. Building on these results, there is a possibility to provide insights into the most important 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. We then predict new malware as a future combination of techniques. Thereafter, we compare approaches to predict impending attacks by fine-tuning the model by propagating trends.

Here we propose, on the one hand, to use genetic algorithms generating malware predictions [7]. On the other hand, we make a dimensional reduction, followed by clustering approaches i.e. hierarchical clustering. A time-series and regression analysis is conducted to identify trends inside these clusters and craft predictions on new malware [8] [9].

To evaluate our algorithms we craft subgraphs from knowledge graphs using time sorting and run the analysis on real data (knowledge at a specific point in time). We evaluate our predictions with walk-forward validation.

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

2. State of research and related work

2.1. Cyber-situational awareness

Situational awareness describes the “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 near future.”[10] In the context of cyber-security, situational awareness can 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 joint 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 subcategories: (1) technical, (2) operational, (3) tactical, and (4) strategic threat intelligence [13]. It includes tactics, techniques, and attack patterns (TTPs), indicators of compromise (IOCs), tools, threat actors, date of discovery, and other information. Information extracted from various sources of CTI is leveraged by many analysts to increase the efficiency 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 a high importance of sharing CTI among different stakeholders to enable a holistic perspective [12]. Hence, there were great 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 [2], which is also utilised by MITRE ATT&CK[3].

MITRE ATT&CK is a globally accessible curated knowledge base and a model about adversarial behaviour in cyber-attacks based on real-world observations. The MITRE Corporation created ATT&CK out of the need to categorize and structure data of 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 improving 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 an opponent will carry out an attack tactic. Each technique is associated with one or more tactics. Tactics can be understood as phases of an attack and are therefore consistent to the cyber kill chain [14]. These tactics answer the question 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 softwares. MITRE ATT&CK consists of three parts: The Enterprise version focuses on adversarial behaviour against enterprises, MITRE ICS version focuses consists of attacks within industrial control systems and the mobile version focuses on attacks against mobile devices.

Furthermore, beside CTI for attacker modelling, there is also information on defender modelling including information on system vulnerabilities. This is for example the National Vulnerability Database (NVD)[4] or the Common Vulnerabilities and Exposures (CVE) database[5].

[2] https://www.mitre.org/capabilities/cybersecurity/overview/cybersecurity-blog/stix-20-finish-line
[3] https://attack.mitre.org/
[4] https://nvd.nist.gov/
[5] https://cve.mitre.org/

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 showed how discrete or continuous models and machine learning methods could be applied to the cyber-security sector in recent years. Husák et. al [9] 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 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]. Within their work, they 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 predict subsequent and linked attack techniques given some currently observed data. 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 can be beneficial for improving cyber-security and setting efficient defensive measures at 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. Furthermore, 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 inter alia 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 NVD has only low predictive power. This might be as there is a close link between the occurrence of new attacks and the exploitation of vulnerabilities. Furthermore, Ozment [19] highlighted that there is not enough information in freely accessible vulnerability databases including NVD. This is, CTI about attacks might have a 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. Methodology

3.1. Hierarchical clustering

Al-Shaer et al [23] proposed agglomerative hierarchical clustering as the most effective means of investigating associations within MITRE ATT&CK. Inspired by their approach we propose the use of hierarchical clustering for finding these associations and predicting yet not existing respectively unobserved links in the usage of attack techniques for crafting predictions on new attacks (new malwares). We thereby calculate for each software the distance to every other software and cluster the most similar ones. The distance between softwares is calculated based on the phi-coefficient. We define the phi-coefficient between software i and software j ($r_\phi(S_i, S_j)$) as follows. For doing so, we formalize each software as a binary vector each bit denotes the utilization or non-utilization of a specific technique.

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

where n_{11} describes the number of techniques that are used by both softwares, n_{00} the number of techniques not used by both softwares and n_{10} respectively n_{01} the number of techniques used by either software i or software j. Furthermore, $n_{1\cdot}$ respectively $n_{\cdot 1}$ represent the total number of techniques used by each software and $n_{\cdot 0}$ respectively $n_{0\cdot}$ the number of unused techniques by software i or j. For clustering we utilized Ward's linkage [24] as it showed the best results.

3.2. Principal component analysis

To identify hidden factors inside the data, principal component analysis (PCA) is a feasible means [25]. It transforms the original data points to a new orthogonal basis. These basis vectors are sorted by the ratio of variance they cover and then interpreted as underlying factors.

3.3. Time series analysis

The time-dependent structure of the attack data implies a time series analysis and forecasting. We propose to use vector autoregression (VAR) [26]. One software is represented by a binary vector of techniques with a date. Each technique can be interpreted as a time series in itself. VAR then 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, we cannot perform a VAR model fitting directly mainly because multiple software have the same time-stamp. Instead, we introduced two different procedures. First, we did ordinary VAR with one software per point in time. Second, we aggregated all software per year and analyzed it to identify trends based on a yearly basis.

Within the first procedure, 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 is the training set for a VAR(n) time series analysis with a lag of n. We performed six regressions with lags $n \in \{1, \dots, 5, 15\}$.

Predictions for future software can be made by the forecast method. It generates vectors of float values. We transformed the prediction to software vectors by setting values above 0.5 to one and the rest to zero. We compare the first prediction for evaluation as further steps of prediction generate the same software structure.

The second procedure involved aggregating all software per year. Next, all values are normalised by dividing them by the number of software in the respective year. This table of relative frequencies of techniques per year gets the subject of analysis. A time-series analysis like in the first procedure is hereby impossible due to the small sample size. However, trends can be identified analyzing the frequencies of techniques used in softwares.

3.4. Genetic algorithm

Genetic algorithms (GA) are a class of algorithms based on the biological process of evolution [27]. Yet, the core idea is transferable to many other research areas. Central for this class of analytical procedures is that the evolution of an initial population respectively the development from a given state is an iterative process, improving their fitness.

Figure 1 gives the basic structure of the GA according to Höschel et al. [28].

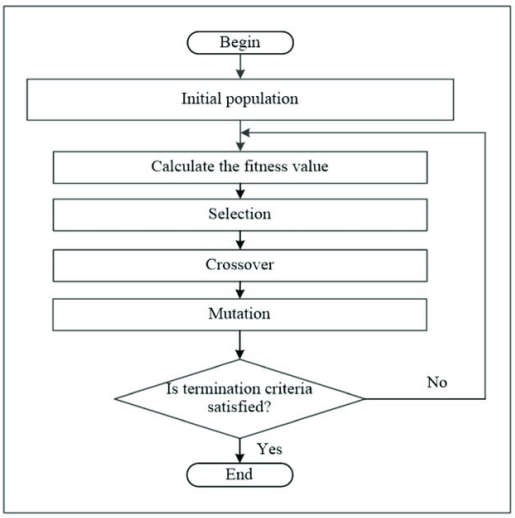

Figure 1: Flowchart of a standard genetic algorithm (GA)

3.4.1. Initialisation

AFP interprets the observed software as the underlying population where each particular technique is interpreted as a determinant of the fitness.

For the initial population, we use the observed probability of the utilization of a technique. For lowering the computational complexity, the size of the initial population could hereby be lowered. In this sense, the run-time can be decreased without significantly deterioration of the accuracy.

3.4.2. Fitness function

The definition of the fitness function is essential for each GA. Within *AFP* we propose to take three terms into account for describing the fitness function which are the probability term (*PT*), the correlation term (*CT*) and the adjustment term (*AT*).

PT considers the frequency of the utilization of a technique in every observed software as an indicator for their fitness. By doing so, we 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 softwares with n potential techniques with the vector q.

PT can then be described as follows.

$$\mathrm{PT} = q \cdot P \in \mathbb{R}^m \tag{2}$$

As some techniques are more likely to appear together or conversely, some techniques are more likely to appear separately, coocurences seem to have an additional information value. This coocurrence of specific techniques can be interpreted as an indicator that they support each other respectively increase the fitness or are even necessary prerequisites. We start with a given generated software vector $p \in \mathbb{R}^n$ with n techniques and the technique correlation matrix $C \in \mathbb{R}^{n \times n}$.

We then substract the unit matrix to remove the self-correlation.

$$\hat{C} = C - E \tag{3}$$

Furthermore, we multiply p by $\hat{C}$ to identify the correlation value for each technique in the generated software. The resulting vector a with

$$a = \hat{C} \cdot p \in \mathbb{R}^n \tag{4}$$

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

$$k = p^T \cdot a \in \mathbb{R} \tag{5}$$

Lastly, this metric is normalized.

$$\hat{k} = \frac{k}{|p|} \tag{6}$$

To do this computation for m populations in parallel, we extend the ideas mentioned above as follows.

$$A = \hat{C} \cdot P \in \mathbb{R}^{n \times m} \tag{7}$$

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.

$$K = P^T \cdot A \in \mathbb{R}^{m \times m} \tag{8}$$

CT is then described by the diagonal elements of the matrix $\mathrm{CT} = \mathrm{diag}(K)$.

We furthermore observe that the number of techniques used within a software is relatively stable. Hence, *AT* is included to account for potential "degenerations" of the predicted software i.e. overfull software vectors. *AT* thereby takes a key role for determination of the accepted false positive and false negative rates and hence for the accuracy of the algorithm. For this purpose, the mean number of techniques in each software within MITRE ATTA&CK is calculated. First, we calculate the occurrence $o \in \mathbb{R}^n$ of different techniques for each predicted software from $P \in \mathbb{R}^{n \times m}$ in parallel.

$$o = P \cdot \mathbb{1} \in \mathbb{R}^n \tag{9}$$

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

In the next step, the difference between the vector o and the mean value μ is calculated by subtracting μ from all the values in o, which leads to the difference term (*DT*).

$$\mathrm{DT} = o - \mu \cdot \mathbb{1} \in \mathbb{R}^n \tag{10}$$

AT penalizes deviations from the mean number of techniques used. We propose to rely on the following formulation where *PST* penalizes positive deviations from the mean and *BST* remunerates negative deviations. *PST* can thereby be described as follows, where *PF* denotes the penalty factor.

$$\mathrm{PST} = \frac{d^2}{\mathrm{PF}} \in \mathbb{R}^n \tag{11}$$

Furthermore, *BST* can analogously be described as follows where *BF* is the "bonus factor" for deviations.

$$\mathrm{BST} = \frac{d}{\mathrm{BF}} \in \mathbb{R}^n \tag{12}$$

Since the differences are negative and the adjustment term *AT* is subtracted in the final fitness function, the bonus has a positive effect on the fitness. *AT* thereby works against the tendency of *GA* to produce overfull software matrices for the predicted softwares. This behaviour happens because it would increase the fitness scores since they are partly based on the occurrence probabilities, and thus more techniques used would lead to higher fitness scores without *AT* favoring the degeneration of software.

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

$$\mathrm{AT} = \mathrm{PST} + \mathrm{BST} \in \mathbb{R}^n \tag{13}$$

The entire fitness function *FT* consists of the three main terms described above.

$$\mathrm{FT} = (\lambda \cdot \mathrm{PT} + (1 - \lambda) \cdot \mathrm{CT}) - \mathrm{AT}) \tag{14}$$

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.

3.4.3. Selection

The selection function selects from the set of softwares those that should be used within the predictive approach to will most likely be reused and recombinated to generate novel software or in other words that are used by the GA to craft predictions. There are different selection methods. We implemented a roulette wheel selection and a simple tournament selection. 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 so called child in the new generation to be recombinated using the crossover methods described in section 3.4.4. This procedure is repeated until an entirely new child generation has been generated.

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 normalized so that the sum of the resulting fitness values is one.

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_i} \tag{15}$$

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 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 software depends on the individuals' fitness relative to the rest of the population. Our implementation allows to individually specify the number of softwares that should be selected for crossover.

3.4.4. Crossover

The crossover operator is the implementation of recombination in the GA. Pairs of softwares (also called parents) are crossed by exchanging segments of the respective bit strings between the two parent softwares. This creates new softwares (children) based on the genetic makeup of the two parents. The number of crossover points in the software vector is usually one or two, implemented using one-point crossover and two-point crossover [29]. However, research has shown that often a more significant number of crossover points can be beneficial [30] [31].

For one-point crossover, two selected softwares are passed and then cut at a random location in the software vector. Two new offsprings are then generated by rejoining the two parent softwares at their intersections. Correspondingly, the two-point crossover method, softwares are cut at two points and then rejoined by combining the respective regions of the parents software vectors to create two new offspring.

Uniform crossover, produces on average ($\frac{L}{2}$) 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 by two parent softwares $A^{(t)}$ 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 software vector. 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].

$$u \leq p_s \tag{16}$$

$$c_i^{t+1} = b_i^t \tag{17}$$

$$d_i^{t+1} = a_i^t \tag{18}$$

This process is repeated up to the length L of the *parent's* software vector. If $u > p_s$ the bits of the parent softwares are copied to the children without flipping them.

3.4.5. Mutation

The mutation function intends to flip bits and thus prevent particular parts of the software vector from being identical in different predicted softwares. This is also to prevent the search for a solution only in a subspace of the original search space. Furthermore, while crossover accounts for the reutilization of software and the recombination of different parts of a software, mutations account for entirely new coding (new developments). Including both, GA seem to come close to real software development processes. Mutations increase the exploratory power of the GA. The probability that a bit mutates is set by a parameter p_m and is usually relatively low.

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.

3.4.6. Optimizing hyper-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 optimized regarding better predictive power. A global optimization routine like simulated annealing [33] is utilized to produce the results stated in section 4.3. While running the optimizer, 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 optimizer is the negative mean of 30 F_1 scores of generated software by the GA.

For comparison, we also tested a local optimization routine. However, the tested limited memory Broyden– Fletcher–Goldfarb–Shanno algorithm (L-BFGS) [34] converged to the start values, and stuck in a local minimum.

3.5. Generative adversial network

Generative Adversarial Networks (GAN) are a specific type of neural networks (NN) frequently used in deep learning 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 (so called generator), in our case software, out of noise input. The other discovers fake data (so called discriminator) [35]. 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.

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 (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 Adam optimizer [36].

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 real samples for this batch. Next, the generator creates new software, which the discriminator evaluates. The generator is trained to fool the discriminator. In this setup, we trained batch-wise with the size of 64 and 2000 iterations.

4. Evaluation

4.1. Database

For evaluation of the proposed methods, we rely on CTI that is presented in MITRE ATT&CK, which is information on 493 malware and 552 unique techniques and sub-techniques these malware rely on.

To be able to train and evaluate *AFP*, we split all available data on the ratio of 80:20. We therefore sort all data by the date of discovery. 80 % of the malware (the malware discovered first - respectively the oldest malware) is taken for training purposes and 20 % (newest malware) for testing. This results in an evaluation period of approximately one year.

Note the imperfections within the database due to unequal time stamping. This is due to a change in the data structure and represents a burden for evaluation as well as for prediction.

Furthermore, we performed a preprocessing i.e. cleaning of the data. We thereby removed *degenerated* 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.

4.2. Experimental setup

4.2.1. Evaluation metrics

We measure the performance of the predictions as the harmonic mean of precision p and recall r - commonly known as the F-score. This means, that for evaluation we measure the gap between the actually observed softwares in the one year time period (Y) and those that were predicted with the help of *AFP*.

$$F = 2 \cdot \frac{p \cdot r}{p + r} \tag{19}$$

Furthermore, we benchmark each module of *AFP* with a simple simulation of attack evolution (as a basic approach of attack prediction) based on the extrapolation of observed probabilities for the utilization of each technique. For generating the baseline, we generated 100 softwares by random picking of techniques. This basic simulation led to an F-score of 0.42. We take the F-score of this extrapolation as a benchmark.

4.2.2. Outline of the experiment

In order to evaluate each implemented module of *AFP*, we perform a Walk forward evaluation with a time horizon of one year. For every prediction we craft by using *AFP* relying on the training data set, we test whether there is a real attack within the evaluation data set. We consider the highest parallelism/ correspondence of a predicted software and an observed software as the adequate measure defining the prediction accuracy, where each observed software (software from the evaluation dataset) can only be chosen once. In doing so, we evaluate the extent to which *AFP* is feasible to predict the development of new attacks within the time horizon of one year which we define as a medium to long term horizon for attack predictions.

We thereby consider the four strategies for crafting attack predictions representing the different modules implemented within *AFP*.

Algorithm 1 describes the schema of the experiment. Line 1 calls the selected procedure for crafting predictions implemented within *AFP*. Line 2 to 4 calculates the precision and recall of each prediction for a single prediction and an observed attack. In Line 5, the evaluation metric is calculated for the prediction crafted and line 6 calculates the evaluation metric of the *AFP* on Y.

Algorithm 1 Evaluation procedure

Input: $DatasetX, DatasetY, AFP\_approach$
Output: $software\_vector, evaluation\_metric$

1: $Predictions \leftarrow GenPredictions(AFP\_approach, DatasetX)$
2: **for** Pred ∈Predictions **do**
3: $precision \leftarrow Get\_precision(Pred, DatasetX, DatasetY)$
4: $recall \leftarrow Get\_recall(Pred, DatasetX, DatasetY)$
5: $evaluation\_metric(Pred) \leftarrow Get\_metric(precision, recall)$
6: $evaluation\_metric \leftarrow Mean(evaluation\_metric(Pred))$
7: **return** $evaluation\_metric, evaluation\_metric(Pred)$

Algorithm 2 shows the procedure of *AFP*.

VAR is called and executed with line 1 Line 3 calls and executes the GA, parameters are set, and the respective selection, crossover, fitness and mutation functions are called, compare Figure 1. Figure 2 shows the best fitness score in each generation and the course of the GA. Last, in line 6 the GAN is called.

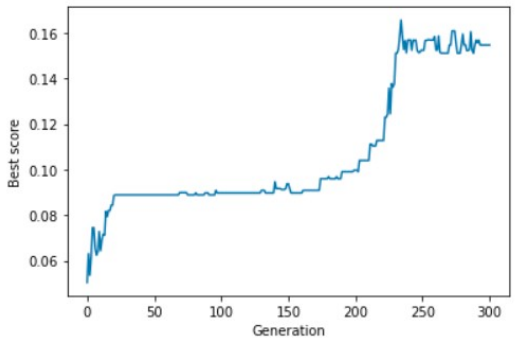

Figure 2: Evolution of the fitness scores in different generations of the GA

Algorithm 2 GenPredictions

Input: $DatasetX, num\_predictions, AFP\_approach$
Output: $Predictions$

1: **if** AFP_approach = VAR **then**
2: $Predicitions \leftarrow GenPredictions\_VAR(num\_predictions, DatasetX)$
3: **else if** AFP_approach = GA **then**
4: $Predicitions \leftarrow GenPredictions\_GA(num\_predictions, DatasetX)$
5: **else**
6: $Predicitions \leftarrow GenPredictions\_GAN(num\_predictions, DatasetX)$
7: **return** $Predictions$

After initializing the start population , the algorithm executes the GA up to a predefined

number of generations. Another essential part of the 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. However, there is no general best replacement strategy, as the choice depends on many problem-specific factors [37]. 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.

4.3. Results

In a first step we applied, hierarchical clustering sorting the software based on the similarity or dissimilarity to one another. The resulting dendrogram is presented in figure 3. It becomes evident, that there are clear clusters that can be extracted from the dataset (types of malware). This could enable prediction approaches within clusters.

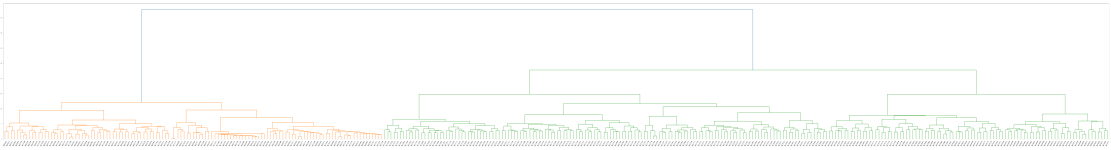

Figure 3: Plot of agglomerative hierarchical clustering of software attacks

Thereafter, we used the data of the clusters to apply PCA. In this case, shown in fig. 4, it was not possible to recognise a structure of temporal developments, and the data points were distributed randomly.

Table 1 presents the results for $n \in \{1, \dots, 5, 15\}$ lags for the VAR(n) time series model. 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. VAR shows with a lag of five a maximal F-score of 0.83 and minimal F-score of 0.4. VAR hence shows in this implementation an improved predictions on future malware for nearly every prediction with only insignificant deterioration even for the worst prediction crafted by the VAR module compared to the baseline.

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, it could be used as a heuristic.

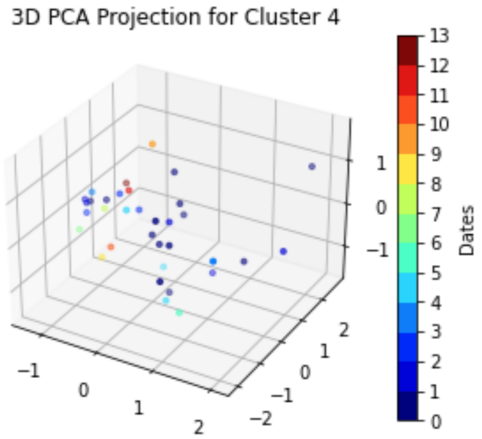

Figure 4: Software clusters with 3 dimensions after performing PCA, colored with reference to the respective timestamps

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

Table 1
Comparison of results of VAR(n) first prediction for different lag sizes.

Figure 5 gives an overview on the performance of the different modules of *AFP*. It shows a clear superiority of *AFP* compared with the baseline (simulation based on the extrapolation of observations).

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.

GA performed best of all 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 optimized parameters (see section 3.4.6), 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, a slightly worse 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

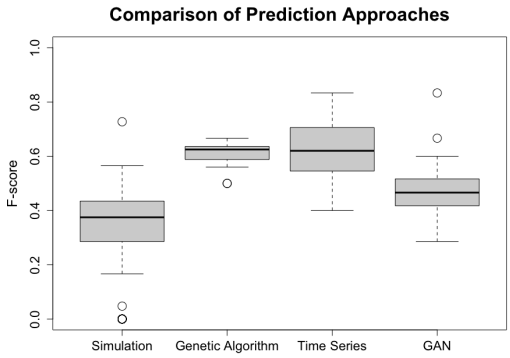

Figure 5: Boxplot showing the comparison in terms of the F-score results of different prediction approaches. We used the simulation as a baseline and ran the Time Series with VAR(5).

F-values of 0.52 - 0.8 ensuring better estimations than the extrapolative approach (benchmark) for every prediction. 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 similar for several runs of the GA.

4.4. Discussion

With *AFP* we present an automated attack predictor able to forecast new malware developments, where each malware is described as a software vector defining the techniques that are used by the novel malware. We showed and validated the predictive power of GA, VAR and GAN based on a Walk forward validation of predictions. *AFP* can therefore be considered as an efficient means for predicting software developments within a time horizon of a year. It needs to be noted, that predictions are not exact rather, real software deviates slightly from most predictions. Yet, predictions show considerable descriptive power. Furthermore, *AFP* helps to increase cyber-security maturity by significantly improving medium to long term predictive capabilities. Furthermore, *AFP* is completely automated saving (time) resources of security professionals and security operation centers. The results of *AFP* can be used to prepare proactively for future attacks and improving investment decisions/ research spending of defenders. This is, defenders can better prepare for attacks utilizing specific attack techniques by implementing adequate counter measures. Furthermore, *AFP* identifies trends in malware development respectively within the utilization of techniques of new malicious software. Security professionals can rely on this information and develop security measures for the most probable predictions even before they where observed *in the wild*. *AFP* can hence also be considered to be an enabler for engineering secure systems within a dynamically changing threat landscape.

Especially GA seems to be suitable for predicting the development of new malware in the medium to long term (respectively as evaluated in this work on the basis of one year). This is as the GA shows very robust results with little variance in the predictive power as well as the

highest mean F-score. Furthermore VAR showed high predictive power. Yet, it suffers from high variance in precision of predictions. Although GAN promises high potential for medium and long term attack prediction, it showed the worst results of the implemented algorithms yet delivering significant improvements compared to the extrapolation of observations suffering most from the limitations of low availability of data on past attacks. This is as *AFP* implements in its current version a GAN that is trained on a relatively small dataset (X). Yet, the training set of the GAN increases with every newly detected malware. This is, we expect the accuracy of the GAN module to increase significantly with the size of CTI utilized.

Likewise, PCA suffers from the lack of data respectively the structure of the database it was extracted. In the future, when data with different time stamps are available, it will most likely become possible to gain deeper insights with this approach.

5. Conclusion, impact & future work

Improving defensive capabilities in cyber-space for improving cyber-security is one of the key challenges that need to be solved to enable resilient societies and modern life, which is increasingly penetrated by information technology.

Understanding the past and predicting the future is an approach being sought in the course of time to develop new security profiles and software to help protect socially sensitive data and critical infrastructure from attackers. Predicting future cyber-attacks can help businesses, individuals and society. Minimising attackers first mover advantage therefore need to be a focal point of research. Yet, it is barely considered or largely based on subjective opinions and biased by individual perspectives of experts.

In this work, we present *AFP* taking advantage of CTI for automatically crafting predictions on future malware and their attack techniques used. In doing so, *AFP* leverages on CTI to infer patterns within time series of attacks, making it possible to gain insights to attack evolution and development as well as deriving further relevant information (e.g. the popularity of specific attack techniques) and crafts forecasts based on this information. *AFP* thereby predicts new attacks as a binary vector of techniques and provides hence actionable insights on the probable development of the cyber-threat landscape. In this way, analysts, researchers and security managers can prepare proactively for threats that are likely to occur in the future. Additionally, cyber-risk managers can perform intelligent and proactive 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 the 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.

Within our work we identified clusters and patterns that can be used for crafting medium to long term predictions on future attack development empowering security analysts in classifying upcoming software, discovery and reaction to trends. Furthermore, these trends promise useful guidance in strategic actions taken by defenders e.g. security investment decisions. With

AFP, we automated the process of crafting attack medium to long term attack predictions and forecasts. The different graph analytical approaches employed by *AFP* show high potential for attack prediction. Yet, GA and VAR show the most promising results.

Although reaching reasonable good results in medium to long term attack prediction, *AFP* suffers a lack of data. Hence, it seems to be of utmost importance to gain and share CTI.

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. Spring, Automating Reasoning of MITRE ATT and CK for Predicting Cyber Attack Techniques using Statistical Machine Learning, Technical Report, Carnegie Mellon University Software Engineering Institute Pittsburgh United ..., 2019.
[24] J. H. Ward Jr, Hierarchical grouping to optimize an objective function, Journal of the American statistical association 58 (1963) 236–244.
[25] J. Shlens, A tutorial on principal component analysis, arXiv preprint arXiv:1404.1100 (2014).
[26] H. Lütkepohl, Introduction to multiple time series analysis, Springer Science & Business Media, 2013.
[27] D. Whitley, A genetic algorithm tutorial, Statistics and computing 4 (1994) 65–85.
[28] K. Höschel, V. Lakshminarayanan, Genetic algorithms for lens design: a review, Journal of Optics 48 (2019) 134–144.
[29] 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.
[30] G. Syswerda, Uniform crossover in genetic algorithms, in: Proceedings of the third international conference on Genetic algorithms, Morgan Kaufmann Publishers, 1989, pp. 2–9.
[31] 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.
[32] T. Dominik-Gwiazda, Genetic algorithms reference, volume i, crossover for single-objective numerical optimization problems, 2006.
[33] C. Tsallis, D. A. Stariolo, Generalized simulated annealing, Physica A: Statistical Mechanics and its Applications 233 (1996) 395–406.
[34] R. Pytlak, Conjugate gradient algorithms in nonconvex optimization, volume 89, Springer Science & Business Media, 2008.
[35] 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).
[36] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).
[37] 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.