Automatic Vulnerability Detection through Machine Learning

1st Hristo Chaparov
Department of Computer Science
Technical University of Munich
Munich, Germany
ge65puh@mytum.de

***Abstract*—Implementing a code with a certain purpose nowadays is nothing but a basic task. It becomes much more complicated if it has to be optimized or if the weaknesses of the script have to be revealed. The machine learning is one of the fastest progressing scopes of the artificial intelligence and can spare a lot of time and financial losses of the companies if the vulnerability detection relies on it. First of all, in this work is observed the work-flow of the machine learning. Than are examined 2 possible approaches to the topic - a source-based and a built-based one on a C and C++ code. After that it is gone through the whole process of both approaches (feature extraction, labeling and a model building). For the model construction and later in the evaluation is paid attention to the following models: word2vec proceeded by the TextCNN convolution layer, pooling layer and the TextCNN fully connected layers, a word2vec proceeded by the TextCNN convolution layer, pooling layer and the extra-trees classifier, a bag of words proceeded through an extra-tree classifier and last but not least, for the build based model - an average vector, based on an average use-def matrix and opcode vectors. At the end, the benchmark evaluates the performance of the models through calculating the false positive rate and the true positive rate after data is provided by packages of C and C++ on Debian (Linux distribution) and Git repositories. The best performance is shown by the model using TextCNN and an Extra-tree classifier.**

I. Introduction

21^{th} century is called the age of information technologies. However, so much open sourced information and fast technological progress is accompanied by many anonymous faces, which try to steal these inventions or try to harm their owners through hacking them. That is why the high levels of security of each implemented software are always prioritized by the product owner or investor. Therefor, many tools were created the last years in order to detect the vulnerabilities of a code. According to the Organization of Internet Safety (OIS) software vulnerability means “a flaw within a software system that can cause it to work contrary to its documented design and could be exploited to cause the system to violate its documented security policy” [1]. Moreover, ”The consequences of a class of system failures, commonly known as software vulnerabilities, violate security policies. They can cause the loss of information, and reduce the value or usefulness of the system” [2]. In each case no matter in what form they are, the software weaknesses can have a crucial impact not only on one person, but also on a whole economy. In 2020 are revealed the statistics that cyberattacks cost on average 200 000$ to companies of all sizes. Furthermore, 43% of the attacks are directed to small companies, from which only 14% have sufficient web defense to protect them [3]. Additionally, in 2003 the cost of software weaknesses on U.S software developers and users in the financial services sector equals to 3342$ millions. When it comes to aerospace losses, the software failures kill hundreds of people, lead to data loss and costs billions [4]. To be avoided or at least to be decreased the negative influence to some extend, the software developers look for solutions in the machine learning scope.

src

3.3 billion dewer src

II. Related work

In this paper are implemented and compared source-based and build-based machine learning models. Both are appropriate approaches for automatic weaknesses detection. There are also other tools for insecurity detection. For instance, GenProg is a tool, which collects foreign data from the source code of other users and checks all of them which fits best as a solution of the existing problem. A certain amount of tests are run on the candidate for fixing everything inappropriate and if they are all passed, the code is fixed [5]. Another useful tool is Prophet. Its software learns probabilistic models and uses them to repair the code. Weaknesses are detected thanks to an algorithm which iterates through the code and creates a list, where all of them are ranked. After that tries to replace the buggy parts with a certain amount of code abstracts accessed from open sourced repositories, runs a couple of tests in order to check if the chosen abstract works correctly and if it fits right to the other parts of the code. If everything runs appropriately the solution is suggested to the user [6]. The big advantage of the machine learning methodology over them is that it learns not only from foreign data used as an input, but also from its own mistakes, which decreases the probability of leaving unrevealed ones. In this work are observed the stages of creating a working machine learning model, its workflow, used for finding vulnerabilities in C and C++ code and at the end an evaluation of the efficiency and comparison of all methodologies, which are used for the implementation.

where does this comes from

III. Machine learning

A. Definition

In order to be reached a greater depth in the scope of automatic vulnerability detection, it has to be clear what the fundamentals of the machine learning are and what kind of processes and goals it hides beneath its surface. An abstract definition of machine learning is a computer system, which finds automatic solutions on a variety of problems based on already collected data. The bigger the data set is, the probability for mistakes of the model is smaller, which makes its usage very reliable.

explain the goal of machine learning
Decission making like a human

One of the most spread machine learning methods are the supervised ones. Often the trained data is saved in pairs (x, y) where x plays the role of an input query and y represents the prediction made based on the provided input. The inputs may be vectors, images and etc., while the output also depends on the essence of the problem. The most developed ones are the binary classification problem, where y is categorized as one out of two possible labels (for example, if it has to be determined if an object is "handmade" or "made by a machine"), multi-class classification, in which y obtains one out of three or more possible labels, multi-label classification, in which to y can be assigned multiple labels, ranking problem, in which y provides a new, suitable order of a set and a general structured prediction problem, where y predicts structured objects, which consist of elements corresponding to certain limitations and requirements [7].

The predictions of supervised learning systems are based on mapping, which can be represented as a function $f(x)$ with input x. There is a big diversity of mappings, for example neural networks, decision forests, logistic regression, decision trees and etc [7], [8].

Also exist a lot of different learning algorithms. For instance, Boosting is an algorithm which combines a couple of weak learners (algorithms, which have really low percentage of accurate prediction, but have better percentage than a random guessing algorithm) in order to combine and convert them to a single strong learner. Similarly, the Multiple kernel learning algorithm in Fig. 1 uses a number of kernels, which are already part of existing methods and uses their functions. Thanks to their similarities in different parameters, they can be combined and instead of creating a new kernel, the same efficiency can be reached with the combination of the existing ones [7].

B. Stages in Machine learning

- Stage 1: Data collection
  Collecting data is essential nowadays. In the century of "Big Data" the leading corporations save their data in order to have enough of it for their machine learning projects, while smaller companies may use public available data sets to develop their projects in the scope. There are two types of data - structured and unstructured.

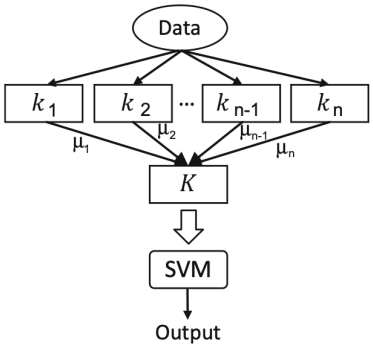

Fig. 1. Multiple kernel learning, (REBAI, Ilyes; BENAYED, Yassine; MAHDI, Walid. Deep multilayer multiple kernel learning. Neural Computing and Applications, 2016, 27.8: 2305-2314.)

The structured one follows a fixed format, mostly an organized table. The header row of such table contains information about the type of the data in a certain box and also its constraints. The unstructured one is in most cases in unordered text format [9].

Even though there are so many data sources, if it can not be found a suitable one and there is some available data, can be used data augmentation, which transforms the data in such way, that the essential information remains unchanged and the data samples increase their amount [9].

Moreover, additional data may be created with the support of synthetic data generators. The first type of generator uses a distribution model such as the Monte Carlo method to generate data. The other one uses deep learning techniques and neural networks for the same purpose [9].

A third way to fill in the needed data is through transfer learning. This means that while looking for a solution of another problem set, may be reached a progress and be found a solution of the current one [9].

- Stage 2: Data preparation
  The stage of preparing data aims to convert the already collected data to an appropriate format in order to be generated a predictive model. Such step in the process is required, because: "Machine learning algorithms require data to be numbers, some of the algorithms impose requirements on the data, statistical noise and errors in the data may need to be corrected and complex nonlinear relationships may be teased out of the data" [10].
  Changes, which have to me made are influenced by the type of the data, the algorithms used in the model and the purposes of the project. Nevertheless there are some standardized steps, which should always be taken:
  -"Data Cleaning: Identifying and correcting mistakes or

list

errors in the data" [10].
-"Feature Selection: Identifying those input variables that are most relevant to the task" [10].
-"Data Transforms: Changing the scale or distribution of variables" [10].
-"Feature Engineering: Deriving new variables from available data" [10].
-"Dimensionality Reduction: Creating compact projections of the data" [10].

Furthermore, on this stage has to be determined which data is going to be used for testing the model and which one for training it as it is shown in Fig. 2.

- Stage 3: Model selection and training
  In order to be made a right decision about that which machine learning model is most suitable to be used, it is needed a detailed knowledge about the core of the data. There are 3 types of machine learning models - a supervised, an unsupervised and a reinforcement one.

  The supervised machine learning is algorithms, in which the model is trained through a provided labeled data and samples with given input and output . The labeled data consists of attributes, which may have either a numerical value or a categorical value. Furthermore, the supervised learning may be proceeded in two different ways. The first one is through classification, where a labeled data is being classified. On the other side there is a regressive supervised learning, which determines the trends of one or more variables on another variables, based on the labeled data [11].

  The unsupervised machine learning is algorithms, in which the provided data is not labeled and are provided samples with only inputs. The model is constructed based on found patterns and trends. Most frequently is used clustering [11].

  The above mentioned algorithms may be clearly distinguished from each other in Fig. 3.

  The third type is reinforcement. in this case the model is trained through experience. It is not determined which actions should be taken, but all possibilities are examined and is decided which one brings most positives. That is why there are two types of reinforcement- the trial-and-error search and the delayed reward [12].

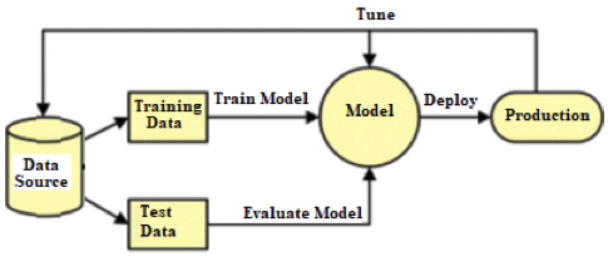

Fig. 2. Machine learning workflow, (Supervised machine learning, unsupervised machine learning, and Deep Learning 2022,[accessed 29 June 2022])

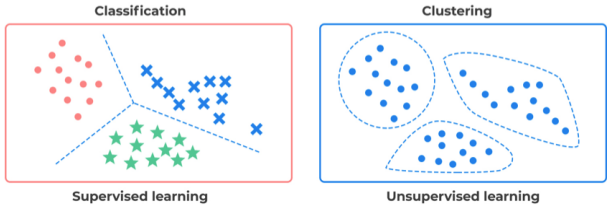

Fig. 3. Supervised learning and Unsupervised learning, (MAHESH, Batta. Machine learning algorithms-a review. International Journal of Science and Research (IJSR).[Internet], 2020, 9: 381-386.)

- Stage 4: Model evaluation
  Cross validation:
  This is a resampling method, used to check how reliable the different kinds of models are, when working with unseen data. For this purpose, approximately 80% of the data is distributed to the training data, while the rest is used for testing. There are two types of cross validation techniques. The first one is called exhaustive and it tests all possible combinations of thr provided data sets. The other one is called non-exhaustive, because it only checks random combinations of the given testing and training data sets [13], [14].

- Stage 5: Performance optimization:
  The machine learning optimization has direct impact on the performance of the hyper parameters of the model. There right set up may be crucial for cases like the following ones [15]:
  - "reduce the human effort necessary for applying machine learning. This is particularly important in the context of AutoML."
  - "improve the performance of machine learning algorithms (by tailoring them to the problem at hand); this has led to new state-of-the-art performances for important machine learning benchmarks in several studie."
  - "improve the reproducibility and fairness of scientific studies. Automated HPO is clearly more reproducible than manual search. It facilitates fair comparisons since different methods can only be compared fairly if they all receive the same level of tuning for the problem at hand."

- Stage 6: Prediction:
  Deployment is a process, in which the working predictive model is attached to a suitable production environment. In order to achieve a successful deployment, the software has to be portable ("Portability is a measure of the ease with which a program can be transferred from one

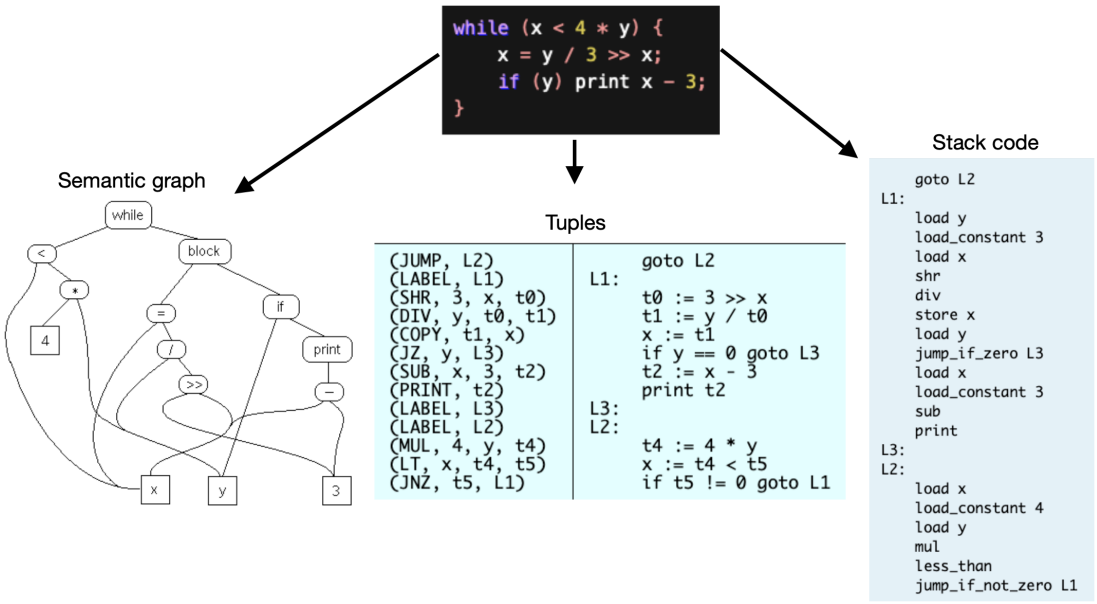

Fig. 4. Examples for IR structures, (Intermediate representations,[accessed 28 June 2022])

environment to another" [16]) and has a good scalability ("means not just the ability to operate, but to operate efficiently and with adequate quality of service, over the given range of configurations" [17]). Furthermore, the main methods of deployment are One-off (upload a model only once), Batch (always upload a model, if there is an updated version of it and is not required a real time prediction) and Real-time (if predictions in real time are required).
It depends on the use purpose of the model, which is the most suitable one. In other words, the prediction is the last stage of the machine learning workflow, where the model works with actual unseen data [13].

C. The impact of Machine learning

The machine learning is an automated process, which has almost nothing in common with the influence of a human being on the way how the machine makes its conclusions. It learns from the received input and based on the new data it develops itself. This combination guarantees a limited risk of mistakes making, isolation of the personal preferences and bigger probability to be noticed new unobserved before vulnerabilities.

IV. Automatic vulnerability detection

A. Idea

In order to examine how the machine learning process may be applied to an existing Candr C++ code and detect its weaknesses, are observed two different approaches. The data sets of both of them have nothing in common, because that is how at the end can be realistically evaluated which model is trained more efficiently [19].

The first approach is based on the features extracted from the intermediate representation (IR) during the compilation of the code. An intermediate representation can be one or more "states", which a code adopt on its way to become a target language from the source language. There are many types of IR such as structured, flat (tuple-based) and flat (stack-based). Fig. 4 shows an example for each of the above mentioned structures. Moreover IRs are divided in three levels: high, medium and low. The higher the level, the more similar is the intermediate representation to the source code [18]. The second one proceeds direct on the source code.

part of compile strucutre

The build-based model receives parts of the code which were already compiled by the compiler, which knows the principles of the language. This may be observed as an advantage, but also can become a drawback, because some key parts of the code, which are meaningless for the compiler and at the same time very useful for the vulnerability recognition may be lost after the compilation [19].

On the other side, in the second approach, the model has zero knowledge about the semantics of the language which is checked. It has to find dependencies between variables. This is called statistical correlation. There are three levels of correlation which are vital for the process of learning from unknown data - positive correlation (the variables change in the same direction [20]), negative correlation (the variables change in the opposite direction [27]) and neutral correlation (there is no relationship in the change of the variables [27]). Moreover, techniques of the natural language processing may be used. NLP is a combination between linguistics and computer science. Its goal is to separate the language in simpler understandable for the machine parts and acquire the logic of

which vars

the language [19].
The combination of multiple approaches increases the chances for greater number of found vulnerabilities.

B. Feature extraction

The needed input (data) which is required is already available, which means that it has to be checked if it is too big, if there is redundancy and etc. If the data is too much and redundant, it is converted into a smaller set of features consisting of the whole useful information of the initial input. This process is called feature extraction and makes the machine learning implementations work more efficiently [22].

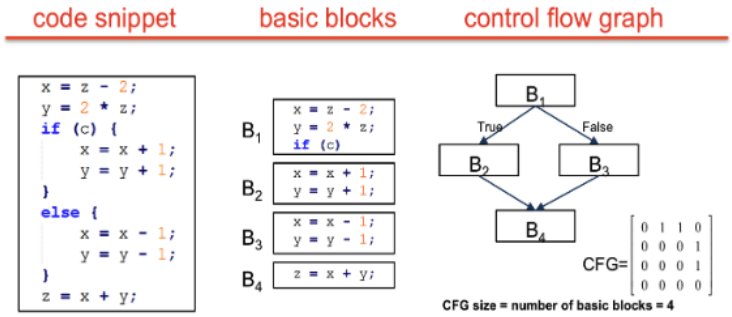

Fig. 5. CFG, (HARER, Jacob A., et al. Automated software vulnerability detection with machine learning. arXiv preprint arXiv:1803.04497, 2018.)

- Build-based:
  To begin with, the build-based implementation is extracted by the tools Clang and LLVM. Clang is a C and C++ code compiler with specifications similar to the GCC compiler and has additionally a couple of code analysis tools. Meanwhile,the LLVM tool takes the already created IRs and compiles them into a binary code. LLVM's IR contains most of the common structures in the different languages and thanks to this it can develop "coroutines and interfacing with C libraries" [23].

  During the build process features are extracted into function level and high-level.

  Firstly, at the function level is extracted the control flow graph of a function. For clarity a couple of terms has to be examined:
  "A directed graph, G, can be denoted by G = (B,E) where B is the set of nodes (blocks) (bl,b2,...,b n) in the graph and E is the set of directed edges ((bi,bj), (bk,b),...). Each directed edge is represented by an ordered pair (bi,b j) of nodes (not necessarily distinct) which indicate that a directed edge goes from node b. to node b" [24]. Moreover "A basic block is a linear sequence of program instructions having one entry point (the first instruction executed) and one exit point (the last instruction executed). It may of course have many predecessors and many successors and may even be its own successor. Program entry blocks might not have predecessors that are in the program; program terminating blocks never have successors in the program" [24].

  CFG ?

  And from the above mentioned definitions it can be constructed a definition for the CFG that it is a directed graph, consisting of basic blocks instead of nodes and with edges playing the role of control flow paths [24]. So, if a code block is taken into account, it is divided into basic blocks. The sequence of the blocks is already defined, therefor it can be constructed a control flow graph as in Fig. 5 [19]. This is useful to be done so, because the model learns about risky execution principles.

  Secondly, the high-level view concentrates its attention on the instruction-level. From each basic block are extracted a use-def matrix and an opcode vector. The use-def matrix is a matrix, which defines, in which block a variable is define and used. For instance, if a variable is defined in instruction l and used in instruction m, then is $(a_{ml}) = 1$, where $(a_{ml}) \in \mathbb{R}^{p \times q}$.
  Moreover, each basic block receives an opcode vector. It has different sections such as conditional, binary, memory address and etc. If the instruction belongs to one of the given categories, it is assigned to 1.
  Using both structures for observing the basic blocks themselves may be useful in order to be identified weaknesses on instruction-level. [19]

- Source-based:
  In order to be made conclusions based on uncompiled code, has to be implemented a C and C++ lexer, which divides the different semantics of the language into groups. That is why it can be said that each word or symbol of the code is distributed in a different token. In the code exist the following parsing standards: "comments (which are currently removed), string literals, single character literals, multiple charac-ter literals, numbers, operators, pre-compiler directives,and names. Names are further sub-categorized into key-words (such asfor,if, etc.), system function calls (suchasmalloc), types (such asbool,int, etc.), or vari-able names. Variable names are all mapped to the samegeneric identifier, but each unique variable name withina single function gets a separate index (for the purposesof keeping track of where variables re-appear)" [25]. "This mapping changes from function-to-function based on the positioning which the identifier first appears. Unique string liter-als, on the other hand, are all mapped to the same singleidentifier. Finally, integer literals are read digit-by-digit,while float literals are mapped to a common token" [19].

  From this point, when there are already elements to

work with, they have to be represented in a numerical way. That is why the following 2 representations are used:

The first one is by creating bag-of-words vector. From the name of the model origins its logic and way of proceeding. It ignores the order of the tokens and pays attention only to that how many times a certain token has been used [26]. In our case, it is constructed a vector with number of elements equal to the amount of different tokens in the code.It may be created a separate vector for each function so. The logic is examined in Fig.6.

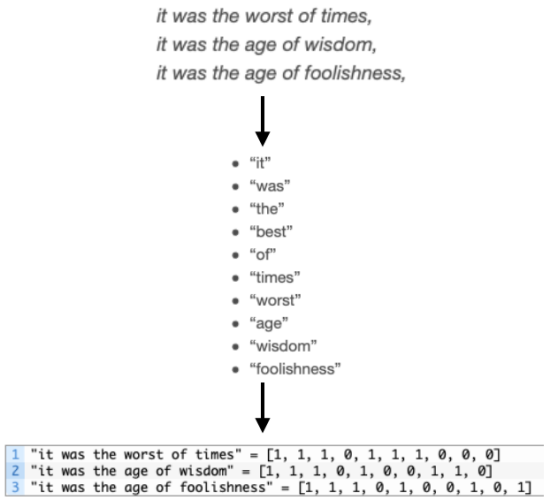

Fig. 6. Bag-of-words model, (Brownlee, A gentle introduction to the bag-of-words model 2019, [accessed on 30 June 2022])

The second one, which is used, is a Word2vec algorithm. This is one of the most efficient ways to be created an embedding of a word (mapping a word to a vector), using a big amount of data. It examines the context in which 2 words are used and if it is similar, then both words are going to be embedded similarly. In order to be able to construct vectors from word, in the essentials of the algorithm lay 2 learning models - CBOW and Continuous skip-gram model. CBOW predicts an unknown word based on the other words. On the other side, skip-gram model predicts the surrounding words of the target one, based on it. Combined both methods create a word vector [28]. A graphical example for the word2vec algorithm is provided in Fig. 7. It can be noticed that the tokens, which are from the same group are stacked near each other. This confirms the correctness of the process. [19]

C. Data labeling

To begin with, both CBOW and Continuous skip-gram model are unsupervised methods, which means that Word2vec is also unsupervised. Therefor, no labeling is needed during the source-based feature extraction.

Meanwhile, during the build-based feature extraction are generated labels for every function through a Clang static analyzer. If a weakness in the code is detected, the function gets labeled. Warnings which do not relate to vulnerability problems are excepted. After that each function is determined either as a good one or as a problematic one. Therefor, if a function is noticed by the machine learning detection model and is also marked as a problematic one by the analyzer, then it is a marked as true positive. If it is detected by the model, but not from the analyzer - it is false positive [19].

example

D. Building a model

- Build-based model:
  The data is extracted, but it still remains the big problem that having a feature vector as a result is impossible at the moment, because the CFG matrix is at functional level, while the opcode vectors and the use-def matrix are at a basic block level. They also have different size compared to the CFG and the basic block. A working solution is to be implemented an average use-def matrix and op-vec for each function [19]. After receiving a feature vector as a result it has to be put into a machine learning model. For this case the random forest model is very appropriate. The random forest prediction is based on the decisions of the decision trees, from which it is built. Each decision tree provides as an output its own result. The random forest classifier takes into account all of them and gives as an output the most popular prediction between its decision trees. [29]

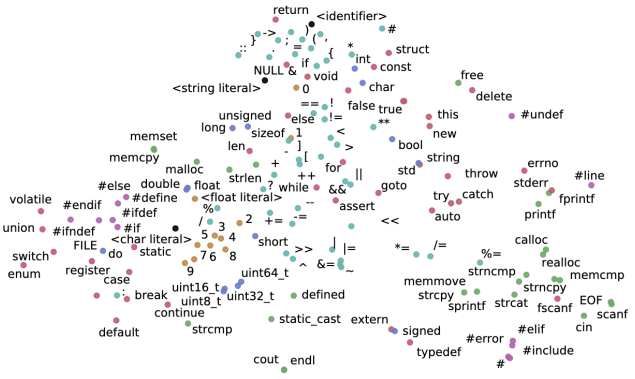

Fig. 7. Word2vec graphical representation, (HARER, Jacob A., et al. Automated software vulnerability detection with machine learning. arXiv preprint arXiv:1803.04497, 2018)

- Source-based model:
  For the source-based model there are a couple of different models, because its input (feature vector) may be represented as bag-of-words or a sequence of word2vec vectors. The models, which are used are an extremely random trees classifier and a TextCNN model [19].
  The extremely random trees classifier is similar to

what is TExtCNN

random forest. It consists of multiple trees but uses no bootstrapping. Moreover compared to the random forest, it makes a random split instead of a best split.
On the other side, the TextCNN is mainly used for text classification. In order to be reached a feature extraction, the matrix received as an input goes through a convolution, a pooling layer and a full connected layer. The convolution layer has a filter (Kernel), which is smaller matrix than the one serving as an input, and goes through its elements to generate a vector. Than the vector goes through the pooling layer, where everything associated with redundancy is removed. At the end comes the fullyconnected layer, where a Softmax function is used, in order to be given a result between 0 and 1 [30].
To sum up, in the source-based model are evaluated the following 3 models [19]:

- Word2vec + TextCNN convolution layer+ TextCNN pooling layer + TextCNN fully connected layers
- Word2vec + TextCNN convolution layer+ TextCNN pooling layer + Extra-trees classifier
- Bag of words + Extra-trees classifier

- Source-based and build-based model:
The combined model of the above mentioned approaches concatenates the build-based vector and the source-based bag of words vector. [19]

V. Evaluation

In the evaluation are examined all of the above mentioned models for the source-based and the building-based models and based on the results are compared the different methods.

A. Provided data

In order to be evaluated all of the varieties of the models, is needed data(in the current case this means C and C++ code). The biggest sources of such kind of data are the packages of Debian Linux distribution and the Git repositories in Github. The training set is used for supervised learning and trains the models to predict if the data is ”buggy” or ”good”. The provided data is separated in three groups - training (data used for the training of the models), validating (data needed for an appropriate set up of the hyperparameters of the model) and testing (data used for testing the model). When the input to the source-based model is provided, it becomes obvious that the data is a lot more than the data of the others. Behind this there is a logical reasoning - the source-based code does not require the code to build successfully. For the build-based models the data set is separated to the Debian one and Git one. The data set for them is split like this, because of the possibility of function duplication. If the same data is in the training data set and in the testing or validating data set, the performance of the model may be influenced together with its set ups. To be avoided such an obstacle, if the source-based data contains a function, which is lexed the same way as another function, then the function must be removed. When it comes to the build-based data, there is the additional condition that if two functions have the same build feature vector, then this function is also removed. This method is the most accurate one in terms of save duplicate removal [19].

B. Benchmarking

In order to evaluate the models, are examined their precision-recall (PR) curves and receiver operating characteristic (ROC) curves. What both AUC methods do, is that they decide to what extent the classification models are correct, but from different points of view. The PR AUC determines the correlation between the recall on the X-axes and the precision on the Y- axes, which means that it pays attention to the positive predictive value and the true positive rate, while ROC AUC examines the correlation between the true positive rate and the false positive rate. The PR AUC gives more preference to the positive class and the ROC AUC is preferred if the data in both negative and positive class is separated equally. So the more points of view there are according to the evaluation, the greater accuracy may be reached during the analysis of the results. Moreover, the closer is the result to 1, the more accurate is the model [31].

- Build-based model:

Dataset	ROC AUC	P-R AUC
Debian	0.76	0.21
Github	0.74	0.22

Fig. 8. Build-based model evaluation, (HARER, Jacob A., et al. Automated software vulnerability detection with machine learning. arXiv preprint arXiv:1803.04497, 2018.)

In Fig.8 are provided the results of the benchmarking of the build-based model, where the inputs of Debian and Github are compared. It can be noticed, that in the case of using ROC AUC the Debian data set has a little advantage, while in the case or PR AUC usage, the advantage is on the side of the Github data set. To summarize, there is no impressive dominance of either of the both data sets.

- Source-based model:

Model	ROC AUC	P-R AUC
BoW + ET	0.85	0.44
word2vec + CNN	**0.87**	0.45
CNN feat. + ET	**0.87**	**0.49**

Fig. 9. Source-based model evaluation ,(HARER, Jacob A., et al. Automated software vulnerability detection with machine learning. arXiv preprint arXiv:1803.04497, 2018.)

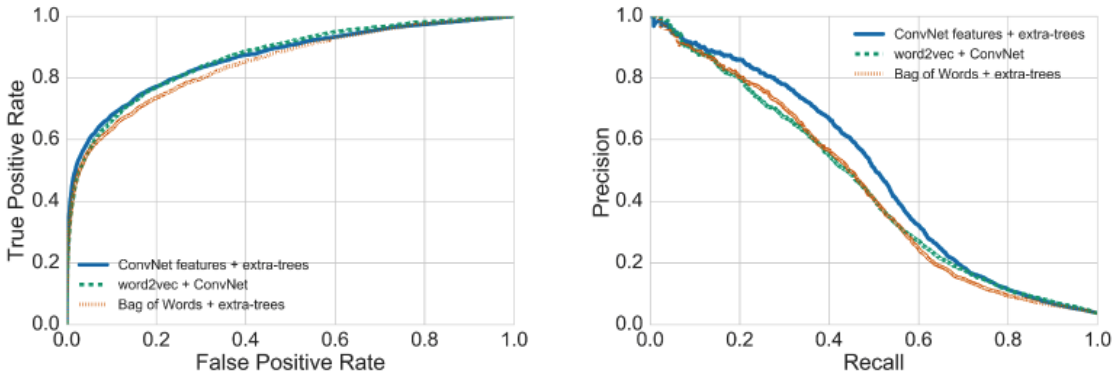

Fig. 10. Banchmarking rates of source-based model, (HARER, Jacob A., et al. Automated software vulnerability detection with machine learning. arXiv preprint arXiv:1803.04497, 2018.)

In Fig.9 are provided the results of the benchmarking of three different source-based models. It can be noticed that in both AUC testing methods, the TextCNN convolution layer combined with an extra-tree classifier gives back the most accurate output. The second best result is shown by the model which, has a word2vec as an output and is manipulated with fully connected TextCNN layers. It has rates similar to those of the model mentioned above and even has the same rates in ROC AUC testing. In this case of label imbalance in the code, the more objective representation is the one of ROC AUC, because of its lack of sensitivity to the label imbalance [19].The rates can be represented graphically as in Fig. 10.

- Comparison:
  To sum up, even if both models are tested with the same data set, the statistics stay similar to those on Fig. 8 and Fig. 9, but it should be emphasized on the fact that the source-based model shows always better results than the build-based model for both statistical approaches. This means that during the IRs' manipulation an important part of the code, which acknowledges insecurity is lost.

VI. Discussion

The vulnerability detection has a big impact not only on the private business, but also on the whole economy. Many private and public projects may be compromised fatally after being hacked or because of human mistake. With the information above, it can be concluded that the machine learning can support security of a code comparably successfully. The weaknesses may be brought to a minimum, but still 100% accuracy looks like an unreachable goal.

VII. Conclusion

In this work has been examined the process of automatic vulnerability detection through machine learning, which analyses a C and C++ code. Each phase of the workflow is separated in two parts. The first one describes the build-based approach, while the other one pays attention to the source-based one. The benchmark shows an accuracy of the source-based model close to 90%, which makes it the most trustworthy one. Furthermore, it should be marked that a weakness of the models is that they are not fully automated. They have to be set up by a data specialist, who has to understand in details the core of the structures and the data. This part may also be automated in order to be decreased the probability for human mistake or not optimal set up of the model.

REFERENCES

[1] AMANKWAH, Richard; KUDJO, Patrick Kwaku; ANTWI, Samuel Yeboah. Evaluation of software vulnerability detection methods and tools: a review. International Journal of Computer Applications, 2017, 169.8: 22-27.

[2] KRSUL, Ivan Victor. Software vulnerability analysis. Purdue University, 1998.

[3] STEINBERG, Scott, 2020, Cyberattacks now cost companies $200,000 on average, putting many out of business. CNBC [online]. 9 March 2020. [Accessed 5 July 2022]. Available from: https://www.cnbc.com/2019/10/13/cyberattacks-cost-small-companies-200k-putting-many-out-of-business.html

[4] PLANNING, Strategic. The economic impacts of inadequate infrastructure for software testing. National Institute of Standards and Technology, 2002, 1.

[5] Genprog,. squaresLab@CMU [online], [Accessed 5 July 2022]. Available from: https://squareslab.github.io/genprog-code/

[6] LONG, Fan; RINARD, Martin. Automatic patch generation by learning correct code. In: Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. 2016. p. 298-312.

[7] JORDAN, Michael I.; MITCHELL, Tom M. Machine learning: Trends, perspectives, and prospects. Science, 2015, 349.6245: 255-260.

[8] HASTIE, Trevor, et al. The elements of statistical learning: data mining, inference, and prediction. New York: springer, 2009.

[9] ZUBCHENKO, Alexander, 2021, Data collection for Machine Learning: The complete guide. Waverley [online]. 28 September 2021. [Accessed 28 June 2022]. Available from: https://waverleysoftware.com/blog/data-collection-for-machine-learning-guide/

[10] BROWNLEE, Jason. Data preparation for machine learning: data cleaning, feature selection, and data transforms in Python. Machine Learning Mastery, 2020.

[11] SARAVANAN, R.; SUJATHA, Pothula. A state of art techniques on machine learning algorithms: a perspective of supervised learning approaches in data classification. In: 2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS). IEEE, 2018. p. 945-949.

[12] SUTTON, Richard S.; BARTO, Andrew G. Reinforcement learning: An introduction. MIT press, 2018.

[13] SELDON, 2021, How to build a machine learning model. Seldon [online]. 11 September 2021. [Accessed 28 June 2022]. Available from: https://www.seldon.io/how-to-build-a-machine-learning-model

[14] BERRAR, Daniel. Cross-Validation. 2019.,[accessed 27 June 2022]

[15] HUTTER, Frank; KOTTHOFF, Lars; VANSCHOREN, Joaquin. Automated machine learning: methods, systems, challenges. Springer Nature, 2019.

[16] POOLE, Peter C.; WAITE, William M. Portability and adaptabilty. Software Engineering, 1975, 183-277.

[17] JOGALEKAR, Prasad; WOODSIDE, Murray. Evaluating the scalability of distributed systems. IEEE Transactions on parallel and distributed systems, 2000, 11.6: 589-603.

[18] Intermediate representations, [Accessed 28 June 2022]. Available from: https://cs.lmu.edu/ ray/notes/ir/

[19] HARER, Jacob A., et al. Automated software vulnerability detection with machine learning. arXiv preprint arXiv:1803.04497, 2018.

[20] HAYES, Adam, 2022, What is positive correlation? Investopedia [online]. 8 February 2022. [Accessed 29 June 2022]. Available from: https://www.investopedia.com/terms/p/positive-correlation.asp

[21] BROWNLEE, Jason, 2020, How to calculate correlation between variables in Python. Machine Learning Mastery [online]. 20 August 2020. [Accessed 29 June 2022]. Available from: https://machinelearningmastery.com/how-to-use-correlation-to-understand-the-relationship-between-variables/

[22] Feature extraction, [Accessed 30 June 2022]. Available from: https://www.mathworks.com/discovery/feature-extraction.html

[23] YEGULALP, Serdar, 2020, What is LLVM? the power behind swift, rust, Clang, and more. InfoWorld [online]. 11 March 2020. [Accessed 30 June 2022]. Available from: https://www.infoworld.com/article/3247799/what-is-llvm-the-power-behind-swift-rust-clang-and-more.html

[24] ALLEN, Frances E. Control flow analysis. ACM Sigplan Notices, 1970, 5.7: 1-19.

[25] GUPTA, R., PAL, S., KANADE, A.,ANDSHEVADE, S. Deepfix:Fixing common c language errors by deep learning. InAAAI(2017), pp. 1345–1351.

[26] ZHANG, Yin; JIN, Rong; ZHOU, Zhi-Hua. Understanding bag-of-words model: a statistical framework. International journal of machine learning and cybernetics, 2010, 1.1: 43-52.

[27] MA, Long; ZHANG, Yanqing. Using Word2Vec to process big text data. In: 2015 IEEE International Conference on Big Data (Big Data). IEEE, 2015. p. 2895-2897.

[28] BIAU, Gérard; SCORNET, Erwan. A random forest guided tour. Test, 2016, 25.2: 197-227.

[29] SONG, Peng; GENG, Chaoyang; LI, Zhijie. Research on text classification based on convolutional neural network. In: 2019 International Conference on Computer Network, Electronic and Automation (ICCNEA). IEEE, 2019. p. 229-232.

[30] ALBAWI, Saad; MOHAMMED, Tareq Abed; AL-ZAWI, Saad. Understanding of a convolutional neural network. In: 2017 international conference on engineering and technology (ICET). Ieee, 2017. p. 1-6.

[31] CZAKON, Jakub, 2021, F1 score vs ROC AUC vs Accuracy Vs PR AUC: Which evaluation metric should you choose? neptune.ai [online]. 31 December 2021. [Accessed 5 July 2022]. Available from: https://neptune.ai/blog/f1-score-accuracy-roc-auc-pr-auc