Experimental Evaluation of Description Logic Concept Learning Algorithms for Static Malware Detection

Peter Švec[1], Štefan Balogh[1] and Martin Homola[2]
[1]*Institute of Computer Science and Mathematics, Faculty of Electrical Engineering and Information Technology, Slovak University of Technology, Ilkovičova 3, Bratislava, Slovakia, Slovak Republic*
[2]*Department of Applied Informatics, Faculty of Mathematics, Physics and Informatics, Comenius University, Mlynská Dolina, Bratislava, Slovakia, Slovak Republic*

Keywords: Malware Detection, Ontology, Description Logics, Machine Learning, Concept Learning.

Abstract: In this paper, we propose a novel approach for malware detection by using description logics learning algorithms. Over the last years, there has been a huge growth in the number of detected malware, leading to over a million unique samples observed per day. Although traditional machine learning approaches seem to be ideal for the malware detection task, we see very few of them deployed in real world solutions. Our proof-of-concept solution performs learning task from semantic input data and provides fully explainable results together with a higher robustness against adversarial attacks. Experimental results show that our solution is suitable for malware detection and we can achieve higher detection rates with additional improvements, such as enhancing the ontology with a larger amount of expert knowledge.

1 INTRODUCTION

In recent years, machine learning algorithms have been increasingly applied to solve various problems in the application domains such as image classification or natural language processing. With an alarming rate of unique malware samples observed in the wild every day, machine learning approaches are considered to be the final solution to malware detection instead of traditional signature based algorithms.

In fact, there has been a lot of research focused on using machine learning algorithms in malware detection, utilizing static features, dynamic features or their combinations. Hassen et al. (2017) used the random forest algorithm with static features extracted from disassembled malicious binaries. Kilgallon et al. (2017), on the other hand, used dynamic features in the form of an API calls made by the monitored binary and applied various machine learning algorithms such as SVM, decision trees or neural networks. Íncer Romeo et al. (2018) introduced an adversarially robust classifier based on monotonic static features and gradient boosting decision trees. Raff et al. (2017) proposed a slightly different approach, where they used whole binaries as an input into convolutional neural networks. An interesting approach proposed by Nataraj et al. (2011), transformed a malicious binary into a gray scale image and adopted traditional algorithms used in computer vision for malware classification.

Despite the potential of machine learning algorithms and high accuracies reported in the literature, we see few, if any, deployed in the real world systems (Smith et al., 2020). Separating malicious and benign binaries is a hard problem, therefore we argue that it is most likely impossible to create a deployable classifier without enhancing our detection mechanisms with a large amount of expert knowledge. Previously mentioned approaches used either feature engineering techniques, to craft a vector representation, suitable for machine learning algorithms or used whole binaries (and their transformations). Traditional vector representations often lack semantics and a lot of valuable information is lost during the transformation process. Solutions that are not using feature engineering and apply the whole binary in an appropriate form as an input, often rely on the fact that machine learning algorithms should automatically learn the features. However, executable files lack proximity relationships and continuity that are present in other domains (Smith et al., 2020). Hence, in the image classification domain, if two neighboring pixels have similar values, the algorithm would assume a proximal relationship. In binaries, on the other hand, in-

Švec, P., Balogh, Š. and Homola, M.
Experimental Evaluation of Description Logic Concept Learning Algorithms for Static Malware Detection.
DOI: 10.5220/0010429707920799
In *Proceedings of the 7th International Conference on Information Systems Security and Privacy (ICISSP 2021)*, pages 792-799
ISBN: 978-989-758-491-6

structions can jump to a different location in the code section and the values next to each other can have significantly different meaning. Saad et al. (2019) identified a few challenges that limit the success of machine learning based classifiers. That is: using larger amount of smaller classifiers (specialized for different malicious behaviors) instead of a single classifier, interpretable results and robustness against adversarial attacks (Kolosnjaji et al., 2018).

In this paper, we propose a proof-of-concept malware detection technique based on description logics learning algorithms. Compared to traditional machine learning techniques, our solution works on a semantic input space and implicitly provides fully interpretable decisions. Our experimental results show that description logics learning is in fact suitable for malware detection systems and can achieve even higher accuracies with larger and richer ontologies, while maintaining the explainability and to some degree robustness against adversarial examples.

This paper is organized as follows. In Section 2, we describe concept learning algorithms in description logics. In Section 3 we discuss our ontology we used during the learning process. Section 4 describes the malware dataset we used in our experiments and its transformation to a semantic dataset annotated by an ontology. Section 5 is devoted to evaluation of our results. Finally, in Section 6 we summarize the results and ideas which we aim to further investigate in the future.

2 DESCRIPTION LOGICS LEARNING

In this section we describe basics of description logics and knowledge bases. Next, we discuss the fundamentals of a concept learning problem. Lastly, we briefly describe our approach and compare the workflow with a traditional machine learning approach.

2.1 Description Logics

Generally, description logics (DLs) are a family of formal knowledge representation languages used mainly for expressing structured knowledge about a specific domain (Baader et al., 2003). DLs are based on three disjoint sets of basic elements: *concepts* $N_C = \{A, B, ...\}$, referring to classes of entities; *roles* $N_R = \{R, S, ...\}$, denoting a binary relationships between individuals of a domain; *individuals* $N_I = \{a, b, ...\}$ referring to instances of concepts. More complex concepts can be built by using a set of various operators such as conjunction, disjunction, etc. The set of provided operators determines the expressiveness of a particular language.

In DLs, the specific domain knowledge is modeled using terminological axioms (TBbox), assertional axioms (ABox) and relational axioms (RBox). TBox represents intensional knowledge between the concepts such as $A \sqsubseteq B$ (A is subsumed by B). ABox contains assertions about named individuals. Such facts are called *concept assertions* such as $A(a)$ (the individual a belongs to the class A) and *role assertions* $R(a,b)$ (the individual a is in the relation that is denoted by R to the individual b). RBox refers to additional properties of roles such as $R \sqsubseteq S$ (R is a subrole of S) and in our work it is considered to be a part of a TBox.

Hence, a DLs knowledge base $\mathcal{K}$ is defined as a couple $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, where $\mathcal{T}$ is the TBox and $\mathcal{A}$ refers to the ABox.

2.2 Concept Learning Problem

Concept learning can be defined as follows (Lehmann and Hitzler, 2010). Suppose we have a knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, a target concept C and a training set $T = Ps \cup Ns$, where Ps are positive examples and Ns are negative examples. The goal of the learning algorithm is to find a concept description D, which approximates C, such that:

$$\forall a \in Ps : \mathcal{K} \models D(a) \qquad (1)$$

$$\forall b \in Ns : \mathcal{K} \models \neg D(b) \qquad (2)$$

Generally, the concept learning algorithms can be defined as a search process in the space of all the possible concepts. These algorithms use a so called *refinement operator*, which returns a set of more specific concepts (in case of a downward operator) from a given concept and heuristics to control how the search tree is traversed.

For our experiments we investigated two learning algorithms: *OCEL* (OWL Class Expression Learner) and *CELOE* (Class Expression Learning for Ontology Engineering). Both are implemented in *DL-Learner*, which is considered to be the state of the art framework for supervised machine learning in DL (Bühmann et al., 2018).

OCEL is the most basic concept learning algorithm. It uses the ρ refinement operator (Lehmann, 2010). The Algorithm provides various techniques to cope with redundancy and infinity by revisiting nodes in the search tree several times and performing additional refinements.

CELOE is a variation of the *OCEL* algorithm. It uses the same refinement operator, but different heuristics that are biased towards shorter concepts. As the algorithm was designed mainly for ontology engineering, shorter concepts are usually more useful for enhancing knowledge bases.

Except from previously mentioned algorithms, there are more approaches that are yet to be fully supported by the *DL-Learner* framework. Tran et al. (2012) proposed the *PARCEL* algorithm; parallel implementation of a concept learning algorithm. Hua and Hein (2019) presented an interesting hill climbing heuristics for concept learning. Rizzo et al. (2016) proposed new DL concept learning algorithms inspired by the traditional decision trees that can also handle uncertainty in the dataset. There are also similar systems to *DL-Learner*, such as *DL-FOIL* (Fanizzi et al., 2008) and its variation for fuzzy DLs (Straccia and Mucci, 2015).

2.3 Approach

We can see our approach in Figure 1. Compared to traditional machine learning approaches, first step is the same; that is preparing the dataset. It is important to have a quality dataset, which is sufficiently representative. Specifically in our case, this step includes gathering the benign/malicious binary samples.

One of the most important step in our workflow is ontology modelling. As we will see in the next sections, designing a rich ontology that is suitable for DL concept learning algorithms is a crucial step. More details are provided in Section 3.

When we have our dataset prepared and the final ontology is designed, binary samples from the dataset have to be mapped in the ontology to create the final *knowledge base*. Usually the data is in the different format, so the samples have to be mapped to the correct format used by the ontology.

Training phase is again similar to traditional machine learning approaches. We can divide our dataset into the training and the testing set. The training set is used by a DL concept learning algorithm, which incrementally produces better and better class descriptions until a sufficient accuracy is achieved.

The final output is in the form of a class expression (or alternatively a set of the most successful class expressions), that can differentiate between malicious and benign samples with a reasonably high accuracy. Using the final class expressions on previously unseen samples is similar as in the previous steps. We simply map the samples into ontology, creating a small *knowledge base* and applying the class expressions. If the sample is satisfied by the class expression, we consider it to be a malware. Otherwise, we label it as a benign application.

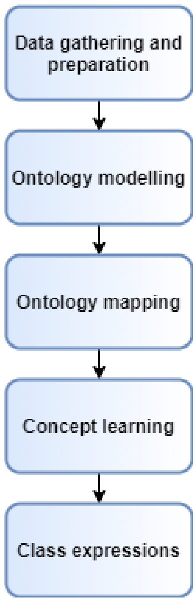


Figure 1: Description logics learning workflow.

3 ONTOLOGY DESIGN

As we mentioned earlier, designing an ontology is a crucial step. During our experiments we designed many models, starting from an overly generalized ontology, representing all aspects of a PE file with high amount of classes, object properties and data properties. However, we found out that these kind of ontologies are too complicated for DL concept learning algorithms and we simply cannot expect to find a sufficient concept description in a reasonable time. Inspired by the work of Oyama et al. (2019), we reduced the complexity and added more granularity into PE imports, creating a subclasses that group together various imports that are used in a similar context. Some examples include process manipulation functions (e.g. *CreateProcess*, *ExitProcess*) or file manipulation functions (e.g. *ReadFile*, *WriteFile*). With these modifications, the results got better, nonetheless we also tried different approaches. We came to the conclusion that we need to inject more semantics and *expert knowledge* into the ontology, as we still worked with features that are not sufficiently discriminative between malicious and benign behaviour.

Our final ontology can be seen in Figure 2. It features 15 classes, 9 object properties and 3 data properties. We decided to include the following main concepts:

- **Debug.** This feature represents the fact that the binary has debugging symbols. In many cases, malware authors are stripping debugging symbols from their executables, as they are used mainly during the development and their presence in the deployed binary significantly helps with reverse engineering.

- **TLS.** *Thread-local storage* is a special section in PE files that enables malware authors to run code stealthily before the original entry point.
- **Signature.** Official benign applications use signatures to prove their non-maliciousness to the operating system. Although signatures used to be present solely in benign applications, in recent years the malware authors are slowly finding their ways on how to sign their binaries.
- **Section.** We also defined the concept of sections in our ontology. This includes three additional concepts representing facts that section has high value of entropy, nonstandard section name (i.e. name that is not usually generated by the most common compilers) or has a read/write/execute permissions enabled. These concepts are present in an ontology mainly for detecting packed binaries. Packers are popular tools that enable to hide the original functionality of a binary (Marpaung et al., 2012).
- **Malicious Behavior.** We decided to group the imported functions based on a malicious behavior they are used for. In total, we defined 6 distinct concepts. For example, *anti-debug* represents functions that are used for hindering the dynamic analysis, such as *IsDebuggerPresent* or *OutputDebugString* or *load_API*, for dynamically loaded API functions (common for packers) such as *LoadLibrary* or *GetProcAddress* (Sikorski and Honig, 2012).

It is important to note that the presented ontology is only a proof-of-concept model. Additional concepts can be added or existing concepts can be extended to add more granularity into the ontology. For instance we can define an *injection* concept and model various injection techniques such as *DLL injection*, *Reflective DLL injection* or *Process Hollowing* as its subclasses (Mohanta and Saldanha, 2020).

Modelling an ontology is the most important step in our approach. While creating an appropriate model requires a lot of effort, well designed ontology should be sufficiently universal for a specific domain. In our case, the presented model is general for *static malware features* and various datasets can be easily mapped to the ontology. Additionally, when new trends emerge in the malware domain, ontology can be easily extented by introducing new concepts.

4 DATASET

For our experimental purposes we have decided to use data from the *EMBER* dataset (Anderson and Roth, 2018). This dataset is a collection of statically extracted features from approximately 1.1 million benign/malicious Windows executables. During the last few years *EMBER* has emerged as one of the most popular datasets and various research has been done regarding the feature usefulness (Oyama et al., 2019).

Each sample in the dataset consists of a single JSON file describing various features of a *Portable Executable* (PE) file. Features are organized in the following categories:

- **General File Information.** These set of features are dedicated to a more general file information such as virtual/raw size, number of imported/exported functions, presence of a signature, etc.
- **Header Information.** This category contains various information from the *Common Object File Format* (COFF) and *Optional* headers such as target machine, timestamps, linker version, etc.
- **Section Information**. Various properties of the sections contained in the executable. These properties include section name, entropy or virtual sizes.
- **Imported Functions.** This list contains imported library functions organized by the library.
- **Exported Functions.** Similar to previous section, this category includes list of exported functions. These are usually included only in dynamic link libraries.

Despite the large amount of different features for each file, we have decided to use only a fraction of them, based on our *expert knowledge* and their usefulness in terms of malicious/benign file separation and their suitability for DL representation (more on this in Section 3). Traditional machine learning approaches that use *EMBER* dataset, usually vectorize the whole feature space, which leads to almost trivial adversarial examples generation simply by modifying/appending a few bytes in an executable.

The malware classification problem is known for its class imbalance issue. Huge case study was made by Li et al. (2017), where they showed that distribution of benign/malicious files in the real world follows approximately 80:1 ratio. However, it is still an open research question on how to correctly prepare the datasets in terms of their malware/benign binaries distribution. We have decided to prepare 4 training sets and 3 testing sets of various distributions. Properties of datasets that we used can be seen in Table 1.

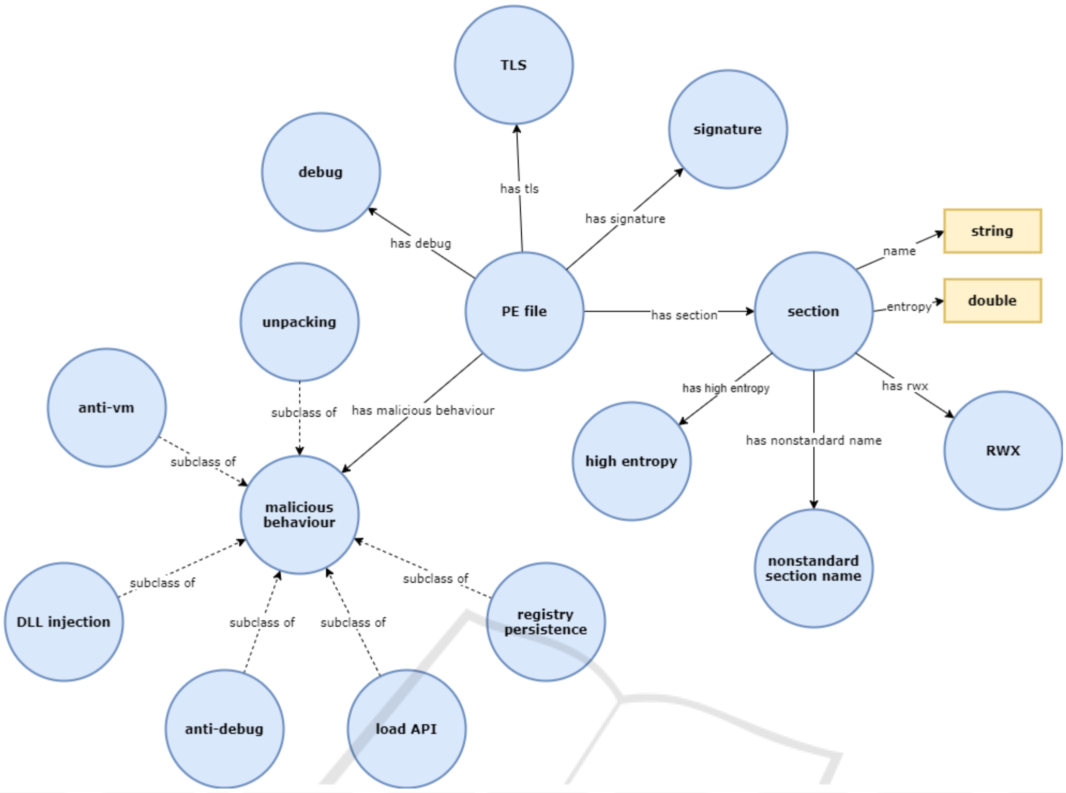


Figure 2: Ontology for static malware features.

$$(\exists has\_tls.\{pe\_tls\}) \sqcup (\geq 3 has\_malicious\_behaviour.\neg load\_api) \tag{3}$$

$$(\exists has\_malicious\_behaviour.(dll\_injection \sqcup registry\_persistence)) \sqcup (\exists has\_tls.\{pe\_tls\}) \tag{4}$$

Figure 3.

5 RESULTS

This section is devoted for discussion about our experimental results.

As we mentioned in previous sections, we evaluated our approach with 4 training and 3 testing sets (with various benign/malware samples distribution), using two concept learning algorithms that is *OCEL* and *CELOE*.

Results from the training phase can be seen in Figure 4. Although we selected a time window of 10 minutes, all the tests were run for exactly 1 hour. This is mainly because we noticed the largest gains in terms of accuracy, during the first 4 minutes. The training accuracy continued to grow, however we observed a radical slowdown after the initial 0 to 4 minutes. This trend was observed in all tests. We can immediately see that in all cases, the training accuracy starting value was based on the percentage of malware samples we used in a particular dataset. The reason behind this fact is that we defined malware binaries as positive examples and the most simple concept that can describe these examples with some value of accuracy is the top concept $\top$ (which is usually the root node in the traversal tree). Hence, if we use a dataset with 10% of positive examples, $\top$ concept can differentiate between positive and negative examples with an accuracy of 10%. We can see that the *OCEL* algorithm achieved higher training accuracies compared to *CELOE*. This is mainly because the *CELOE* algorithm heuristics implicitly prefer shorter class descriptions that should be not suitable in malware application domain as we need more complex concepts. Although we can see that the accuracies are much closer to each other with an increasing percentage of malware samples in the dataset. It is also important to note that while we used datatype properties in the ontology, we disabled numeric and string constructors

Table 1: Properties of datasets used during the experiments.

Name	Malware	Benign	Class assertions	Object property assertions	Data property assertions
Testing set $\mathcal{T}_1$	500	500	5674	7600	9324
Testing set $\mathcal{T}_2$	200	800	5796	7244	9598
Testing set $\mathcal{T}_3$	800	200	5693	8023	9362
Training set $\mathcal{A}$	1000	9000	57 068	69 730	94 112
Training set $\mathcal{B}$	2000	8000	57 030	71 434	94 036
Training set $\mathcal{C}$	3000	7000	57 082	72 902	94 140
Training set $\mathcal{D}$	6000	4000	57 791	78 391	95 558

Table 2: Accuracies detected for various class expressions.

	$\mathcal{A}(OCEL)$	$\mathcal{A}(CELOE)$	$\mathcal{B}(OCEL)$	$\mathcal{B}(CELOE)$	$\mathcal{C}(OCEL)$	$\mathcal{C}(CELOE)$	$\mathcal{D}(OCEL)$	$\mathcal{D}(CELOE)$
$\mathcal{T}_1$	66.20%	55.90%	66.20%	55.90%	66.60%	60.30%	66.40%	60.30%
$\mathcal{T}_2$	**82.40%**	40.10%	**82.40%**	40.10%	76.50%	54.40%	**72.40%**	54.40%
$\mathcal{T}_3$	54.80%	**72.27%**	54.80%	**72.27%**	**79.70%**	**67.8%**	60.10%	**67.80%**

for the refinement operator ρ, since they proved to be very problematic during the learning process and instead we defined concepts such as *high entropy* and *nonstandard section name*. Despite that we kept that information in the ontology for more explainability.

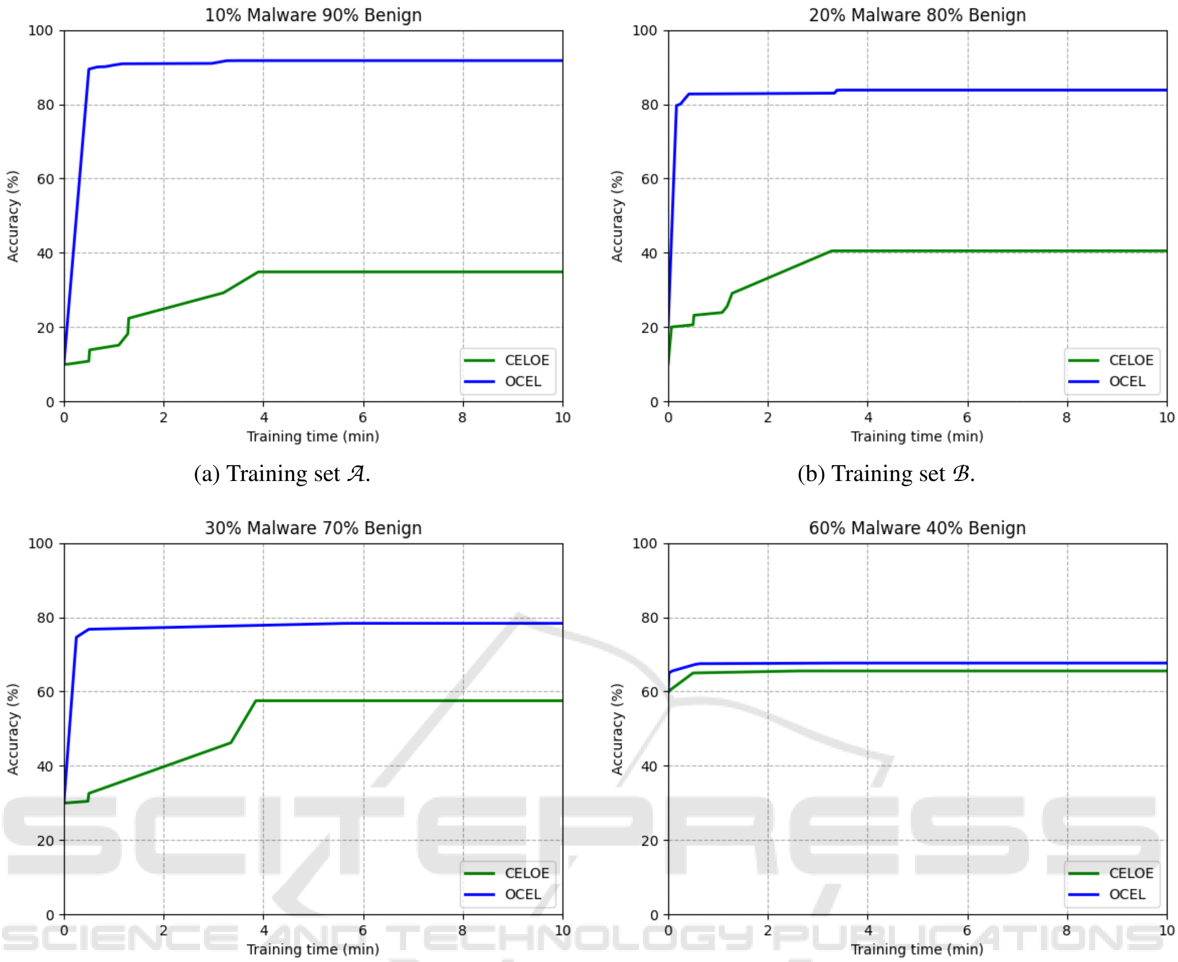
We can see some of the most successful class descriptions in (3) and (4). Class description (3) can be directly interpreted as follows: *a binary is malicious if the TLS section is present or if it has at least three malicious behaviors, except dynamic API loading*. Second class expression (4) can be interpreted similarly: *binary is malicious if it has a malicious behavior that is either DLL injection or registry persistence or if the TLS section is present*. So we can see that from the limited training dataset we used, concept learning algorithms considered concepts such as *TLS* or *malicious_behaviour* important in terms of malware/benign separation.

Then, we applied the most successful class descriptions learned from various training sets to 3 different testing sets, which contained samples that were not used during the training phase. We can see the results in Table 2. As expected, the *OCEL* algorithm performed much better almost in all cases. Although at first glance the results may seem average, especially compared with the state of the art machine learning classifiers, we consider them to be very promising. It has to be noted that the experiments were done with limited dataset and more importantly, limited ontology. We believe that with richer and more complex ontology we can achieve similar results as current classifiers, while offering additional properties. As previously mentioned, one of the properties is full explainability. Another property is higher robustness against adversarial attacks. Current machine learning classifiers are vulnerable against various forms of attacks, especially in the static detection that includes appending bytes to binaries, adding imports or perturbating unused sections (Suciu et al., 2019). These attacks would simply not work even for our smaller class expressions. In order to evade our class expression based classifier, the attacker would need to get rid of the TLS section or rewrite the original code, so that it uses dynamic loading (however this behavior may be also learned as dangerous in case of previously mentioned possible improvements). So there are definitely methods of evasion, however, they come at a higher cost.

6 CONCLUSION AND FUTURE WORK

Our work has led us to the conclusion that DL concept learning algorithms are in fact suitable for malware detection. Further possible improvements have to be investigated in the future research, which include:

- **Ontology Enhancement.** As we mentioned in previous sections, in order to achieve higher accuracy and robustness, we need to inject more *expert knowledge*. This includes specifying various additional malicious properties as concepts or expanding different concepts into subclasses.
- **Larger Datasets.** Since our work is mainly proof-of-concept, we used relatively small datasets for training and testing. This could be the main reason why various concepts included in our ontology were ignored in the final class expressions. Using larger amount of samples could improve DL heuristics and lead the algorithm to find additional malicious patterns, hence producing more



(a) Training set $\mathcal{A}$. (b) Training set $\mathcal{B}$.

(c) Training set $\mathcal{C}$. (d) Training set $\mathcal{D}$.

Figure 4: Training results for various datasets.

complex and robust class expressions.

- **Parallelism.** In this work we explored two different DL concept learning algorithms, that is *OCEL* and *CELOE*. However, both algorithms suffer in terms of performance, as they are both implemented as single threaded applications. There are two parallel algorithms available in *DL-Learner*, such as *PCELOE* or *PARCEL*, although currently, they are not fully supported. Since the DL concept learning is a search problem, utilizing more threads would potentially result in traversing more concepts, hence leading to better class descriptions in the same time.

- **Fuzzy Ontology.** Some concepts in our ontology, such as *high entropy*, were specified based on a threshold value. This kind of concepts are however more suitable for representation in fuzzy logics (supported to some degree by *DL-Learner*). Similar fuzzy functions could be also applied to additional features present in an *EMBER* dataset, such as number of exports, amount of strings recognized as registry values, etc.

In this paper we investigated static malware features, although presented approach could be in similar manner applied to a dynamic features or even various system events (Balogh and Mojžiš, 2019). Lot of research focus on static malware detection, however, it is still questionable, if various static binary properties provide enough information for malware/benign separation, due to the large amount of different obfuscation techniques. An interesting approach was proposed by Biondi et al. (2019), where they used static features and machine learning only for detecting packed binaries, as the packers are mainly used in malicious samples. Various DL concept learning algorithms could be potentially also applied for this kind of objective.

ACKNOWLEDGEMENTS

This research was sponsored by Slovak Republic under grants VEGA 1/0159/17 and APVV-19-0220 and by the EU H2020 programme under Contract no. 952215 (TAILOR).

REFERENCES

Anderson, H. S. and Roth, P. (2018). Ember: an open dataset for training static pe malware machine learning models. *arXiv preprint arXiv:1804.04637*.

Baader, F., Calvanese, D., McGuinness, D., Patel-Schneider, P., Nardi, D., et al. (2003). *The description logic handbook: Theory, implementation and applications*. Cambridge university press.

Balogh, Š. and Mojžiš, J. (2019). New direction for malware detection using system features. In *2019 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, volume 1, pages 176–183. IEEE.

Biondi, F., Enescu, M. A., Given-Wilson, T., Legay, A., Noureddine, L., and Verma, V. (2019). Effective, efficient, and robust packing detection and classification. *Computers & Security*, 85:436–451.

Bühmann, L., Lehmann, J., Westphal, P., and Bin, S. (2018). Dl-learner structured machine learning on semantic web data. In *Companion Proceedings of the The Web Conference 2018*, pages 467–471.

Fanizzi, N., d'Amato, C., and Esposito, F. (2008). Dl-foil concept learning in description logics. In *International Conference on Inductive Logic Programming*, pages 107–121. Springer.

Hassen, M., Carvalho, M. M., and Chan, P. K. (2017). Malware classification using static analysis based features. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–7. IEEE.

Hua, Y. and Hein, B. (2019). Rapid restart hill climbing for learning description logic concepts. In *International Conference on Inductive Logic Programming*, pages 46–61. Springer.

Íncer Romeo, Í., Theodorides, M., Afroz, S., and Wagner, D. (2018). Adversarially robust malware detection using monotonic classification. In *Proceedings of the Fourth ACM International Workshop on Security and Privacy Analytics*, pages 54–63.

Kilgallon, S., De La Rosa, L., and Cavazos, J. (2017). Improving the effectiveness and efficiency of dynamic malware analysis with machine learning. In *2017 Resilience Week (RWS)*, pages 30–36. IEEE.

Kolosnjaji, B., Demontis, A., Biggio, B., Maiorca, D., Giacinto, G., Eckert, C., and Roli, F. (2018). Adversarial malware binaries: Evading deep learning for malware detection in executables. In *2018 26th European Signal Processing Conference (EUSIPCO)*, pages 533–537. IEEE.

Lehmann, J. (2010). *Learning OWL class expressions*, volume 22. IOS Press.

Lehmann, J. and Hitzler, P. (2010). Concept learning in description logics using refinement operators. *Machine Learning*, 78(1-2):203.

Li, B., Roundy, K., Gates, C., and Vorobeychik, Y. (2017). Large-scale identification of malicious singleton files. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, pages 227–238.

Marpaung, J. A., Sain, M., and Lee, H.-J. (2012). Survey on malware evasion techniques: State of the art and challenges. In *2012 14th International Conference on Advanced Communication Technology (ICACT)*, pages 744–749. IEEE.

Mohanta, A. and Saldanha, A. (2020). Code injection, process hollowing, and api hooking. In *Malware Analysis and Detection Engineering*, pages 267–329. Springer.

Nataraj, L., Karthikeyan, S., Jacob, G., and Manjunath, B. S. (2011). Malware images: visualization and automatic classification. In *Proceedings of the 8th international symposium on visualization for cyber security*, pages 1–7.

Oyama, Y., Miyashita, T., and Kokubo, H. (2019). Identifying useful features for malware detection in the ember dataset. In *2019 Seventh International Symposium on Computing and Networking Workshops (CANDARW)*, pages 360–366. IEEE.

Raff, E., Barker, J., Sylvester, J., Brandon, R., Catanzaro, B., and Nicholas, C. (2017). Malware detection by eating a whole exe. *arXiv preprint arXiv:1710.09435*.

Rizzo, G., Fanizzi, N., Lehmann, J., and Bühmann, L. (2016). Integrating new refinement operators in terminological decision trees learning. In *European Knowledge Acquisition Workshop*, pages 511–526. Springer.

Saad, S., Briguglio, W., and Elmiligi, H. (2019). The curious case of machine learning in malware detection. *arXiv preprint arXiv:1905.07573*.

Sikorski, M. and Honig, A. (2012). *Practical malware analysis: the hands-on guide to dissecting malicious software*. no starch press.

Smith, M. R., Johnson, N. T., Ingram, J. B., Carbajal, A. J., Haus, B. I., Domschot, E., Ramyaa, R., Lamb, C. C., Verzi, S. J., and Kegelmeyer, W. P. (2020). Mind the gap: On bridging the semantic gap between machine learning and malware analysis. In *Proceedings of the 13th ACM Workshop on Artificial Intelligence and Security*, pages 49–60.

Straccia, U. and Mucci, M. (2015). pfoil-dl: Learning (fuzzy) el concept descriptions from crisp owl data using a probabilistic ensemble estimation. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 345–352.

Suciu, O., Coull, S. E., and Johns, J. (2019). Exploring adversarial examples in malware detection. In *2019 IEEE Security and Privacy Workshops (SPW)*, pages 8–14. IEEE.

Tran, A. C., Dietrich, J., Guesgen, H. W., and Marsland, S. (2012). An approach to parallel class expression learning. In *International Workshop on Rules and Rule Markup Languages for the Semantic Web*, pages 302–316. Springer.