Real-Time Deep Learning-Based Malware Detection Using Static and Dynamic Features

Radu Ştefan Mihalache, Dragoş Teodor Gavriluţ and Dan Gabriel Anton
Bitdefender Laboratory, Faculty of Computer Science, "Al.I. Cuza" University, Iaşi, Romania

Keywords: Malware, Threat Detection, Neural Network, Static Features, Dynamic Features.

Abstract: Cyber-security industry has been the home of various machine learning approaches meant to be more proactive when it comes to new threats. In time, as security solutions matured, so did the way in which artificial intelligence algorithms are being used for specific contexts. In particular, static and dynamic analysis of a threat determines certain characteristics of an artificial intelligence algorithm (such as inference speed, memory usage) used for threat detection. While from a product point of view, static and dynamic analysis of a threat target separate product features such as protection for static analysis and detection for dynamic analysis, the feature sets derived from analyzing threats in those two scenarios (static and dynamic analysis) are complementary and could improve the accuracy of a model if used together. The current paper focuses on a multi-layered approach that takes into consideration both static and dynamic analysis of a threat.

1 INTRODUCTION

The cyber-security industry has evolved along with the constant increase of attacks that led to more than 1.2 billion[1] known malware in 2023. Nowadays, most of the cyber-security solutions rely on different machine learning algorithms for threat detection.

At the same time, cyber-security solutions evolved in an attempt to cover various needs from both consumer and business markets. One of the major differences in these cases is that while consumers are more focused on protection (the role of a security solution being to make sure that nothing bad happens to a system), the business solution also focuses on providing visibility around an attack. Most of the requirements that appear in the business market are a direct result of several compliance rules that enterprises have to obey. For example, if a cyber-security attack succeeds on a bank, the bank is required to start an internal investigation that analyzes logs and additional attack artifacts to better understand the impact of that attack.

With this, requirements for a new type of machine learning algorithm emerge, that focuses on the behavior of an attack and uses logs or asynchronously obtained events to create features that are further used by the algorithm. These algorithms cannot block a threat but can provide late triggers about an undergoing attack. These triggers are often referred to as detection capabilities as they do not provide any protection.

Models designs for detection don't necessarily need to have a low false positive rate as they can not block anything. As such they are usually allowed a certain level of false positives if the detection rate is increased. Another important observation is that the output of such models is usually analyzed by a SOC (Security Operation Center) team that determines if an attack is real or not. This behavior led to a phenomenon called *alert fatigue*, meaning that alerts from these models accumulate to a point where they are hard to be analyzed by security officers.

Our paper focuses on a method that combines two types of models: based on statically extracted features and models designed for detection, with the purpose of reducing the *alert fatigue* phenomena while preserving a high detection rate. For this purpose we have used multiple malicious files that were analyzed from a dual perspective: **a static analysis** perspective where we extract features based on meta information we can extract from the malicious files and **a dynamic analysis** perspective where we extract features that reflect the malware behavior at runtime.

The rest of the paper is organized as follows: section 2 reveals similar research, section 3 describes the current cyber-security landscape and the problem we are tackling, section 4 presents our approach on building a neural network that uses both statically and

[1] https://www.av-test.org/en/statistics/malware/

Mihalache, R., Gavriluţ, D. and Anton, D.
Real-Time Deep Learning-Based Malware Detection Using Static and Dynamic Features.
DOI: 10.5220/0012316800003636

In *Proceedings of the 16th International Conference on Agents and Artificial Intelligence (ICAART 2024) - Volume 3*, pages 226-234
ISBN: 978-989-758-680-4; ISSN: 2184-433X

dynamically extracted features, section 5 shows the databases used in our experiment and several results and finally, section 6 draws several conclusions on the practical aspects of our proposal.

2 RELATED WORK

Malware detection has been one field that attracted a lot of attention from researchers who used various machine learning methods. There are two main approaches that are usually employed when it comes to deciding what features will be used in the process of training and evaluating machine learning models.

The first approach involves extracting **static** features from the files, without executing them. Thus, this approach usually consumes fewer resources and reaches high speed and potentially high accuracy.

(Ahmadi et al., 2016) proposed a malware family classification system, using a wide range of **static** features extracted from the original PE executable files that were not unpacked or deobfuscated. By combining the most relevant feature categories and feeding them to a XGBoost-based classifier, their model reached an accuracy of around 99.8% on the Microsoft Malware Challenge dataset (Ronen et al., 2018) of 20000 malware samples.

Other approaches demonstrated that minimal knowledge is needed for extracting relevant **static** features from executable files. For instance, a study demonstrated that effective malware detection can be obtained using the information in the first 300 bytes from the PE header of executable files as input (Raff et al., 2017b). The same year, a more comprehensive study (Raff et al., 2017a) presented MalConv, a deep convolutional neural network model which directly uses the raw byte representation of executable (limited to the first 2 MB) as input, without any intelligent identification of specialized structures or specific executable or malware content. The model showed good results, achieving 94% accuracy after training on a large dataset of 2 million PE files.

In a more recently conducted study (Zhao et al., 2023), the authors researched a different method, converting the bytecode extracted from the original files into color images and using them as input features for training an AlexNet convolutional neural network (CNN). The results were promising, the accuracy of their model reaching more than 99% on two rather small public malware datasets from Google Code Jam and Microsoft of around 10000 samples.

However, using **static** features alone might bring some limitations in real-world malware detection scenarios where advanced obfuscation, packing or encryption are being used for creating malicious files. In a recent study (Aghakhani et al., 2020) this aspect was investigated and, using a dataset of almost 400 thousand files, it was demonstrated that using static information exclusively is not indicative of the actual behavior of the classified files and a substantial number of false positives on packed benign files occur.

The other main approach would be to extract **dynamic** features that describe the behavior of the malware during execution or partially retain information regarding the said behavior.

One method is to include dynamic runtime opcodes as input features, allowing the behavior of executables to be captured. An extensive study (Carlin et al., 2019) showed that this approach can accurately detect malware, even on a continuously growing and updatable dataset that requires retraining. The authors compared 23 machine learning algorithms and concluded that their method worked best using the Random Forest model.

In a recent study (Zhang et al., 2023), the authors proposed another method of combining the API call sequences-based dynamic features with the semantic information of functions, bringing more context to the actual performed action by the API call. Compared to existing similar experiments that only used API call information, their solution shows improvements of 3% to 5% in detection accuracy.

(Ijaz et al., 2019) compare several methods based on machine learning for detecting Windows OS executables. They use a small set of files of only 39000 malicious binaries and 10000 benign ones, from which they statically extract a small set of 92 features from the PE headers using the PEFILE tool. They also dynamically extract 2300 features from a small part of the files from the execution in Cuckoo Sandbox. Their detection measurements are made using either the **static** features or the **dynamic** features separately. Also, using a sandbox for the training and evaluation part when using the dynamic features brings in a series of disadvantages, because it does not provide a form of real-time protection for the new malicious files that would need to be evaluated.

In one of the first such approaches, (Santos et al., 2013) present a **hybrid** malware detection system that combines both **static** and **dynamic** features. The small dataset they use consists of 1000 malware and 1000 legitimate files, from which they extract two-byte opcodes, perform feature selection using Information Gain and select the first 1000 as the static features that will be used. The dynamic characteristics are extracted by monitoring the behavior of the programs in a controlled sandboxed environment.

Another different **hybrid** approach that uses both static and dynamic features for malware detection is proposed in (Zhou, 2019). The authors use a sandbox for recording API call sequences from the execution of 90000 malicious and benign files and they extract dynamic features out of a trained RNN model that is fed with the recorded sequences. The static and dynamic features are then combined into custom images that will be used in the training and validation phase of a CNN model. Both studies demonstrate how combining both static and dynamic characteristics brings improvements in detection rates.

Compared to our approach, these two studies have a few limitations, one related to the small number of files used in the dataset and another related to the usage of a sandbox for extracting dynamic features during execution. Thus, their systems could provide only offline detection and classification mechanisms and are no practical solutions for a product which must provide real-time protection against malware.

3 PROBLEM/SECURITY LANDSCAPE

The cat-and-mouse game has been a constant of the cyber-security ecosystem for decades; malicious actors create a new threat, cyber-security solutions adapt then the new threats adapt to the new cyber-security changes and the cycle goes on. And while this type of change is inevitable, there were other (more business-related) changes that a security product suffered during the years.

One such important change was the split between types of users: enterprise and consumer. While consumer users are more interested in protection (the security solution is perceived as a tool that quietly ensures that everything is secured), the business environment comes with several different challenges. When a breach happens, there is a need (sometimes driven by compliance regulations) to understand exactly what endpoints were affected, what kind of data was exfiltrated, when the attack started or what set of measures would reduce the chance for a similar attack to happen in the future.

While these differences relate mostly to a product feature (centralized dashboard, reports for enterprise environment and automated flows for consumer), there are several differences that regard threat detection as well. As a result, threat detection differences can be classified in regards to:

- **Static Detection** - usually associated with pre-execution / on-access scanners. The main characteristic of this type of detection is that it takes a file as an input (but a file that was not executed yet) and analyzes it (in terms of its content). Security products refer to this type of detection as **protection** as that file is not executed yet and detecting it at this point blocks the attack and keeps the user protected. This is heavily used in consumer products where the expectation is to block everything and keep the user protected.
- **Dynamic Detection** - usually associated with post-execution scanners. It implies that the file is allowed to run, while at the same time its actions are monitored. This type of detection is better at identifying behavior and intent, but it's less resilient in terms of protection (once some data is copied to an external site, even if we record the event, we cannot un-copy it). Enterprise solutions use this method as part of EDR/XDR products to record data related to an attack and automatically create a root cause report.

From a detection point of view (and in particular, if we refer to machine learning models) there are several distinct features that each of these two detection methods (static and dynamic) have:

- **Static Detection** methods are usually used in the pre-execution phase. This actually means that for example, before a file gets executed, its content is scanned. From a technical perspective, this is achieved via a kernel mode driver that stops the execution until the result from the scanner is available. While this method ensures protection (nothing gets executed unless it was scanned), it also imposes certain limitations. If for example, the duration of a scan is one second for each file, the entire operating system will be heavily slowed down. As such, models that are used in this phase have to be fast (fewer neurons or other forms of more classical machine learning approaches such as binary decision trees, random forests, etc). It's also important to notice that features used in these methods are extracted directly from the file content (strings, section information, disassembly listings, imports and exports, etc) and don't reflect the behavior of a sample but rather a probability of something being malicious.
- **Dynamic Detection** on the other hand is used with events that are recorded asynchronously. As such, the performance impact is reduced and assuming storage space is not an issue, larger models (e.g. neural networks with multiple hidden layers) can be used. It is also worth mentioning that the input to these models reflects behavior (system events, APIs that are being used, etc).

As a general concept, a security solution uses these two forms of detection sequentially (first a file

Table 1: Static vs dynamic detection.

	Static Detection	Dynamic Detection
Susceptible to packers and obfuscation techniques	Yes	No
Behavior false positive (use of startup registry keys)	No	Yes
Susceptible to dynamic detection evasion techniques (behave differently if monitored)	No	Yes

is scanned with the static scanner, then if nothing is found it gets executed and the dynamic scanner analyzes it). However, each one of these two solutions comes with several limitations in terms of detection - as shown in table 1.

Since these evasion techniques are often used by advanced malware in different stages of an attack, using just one of these detection methods or both but sequentially may be ineffective.

4 SOLUTION

In order to address the problem of advanced malware detection, we set on developing a neural network that classifies a program as benign (labeled 0) or malicious (labeled 1) by analyzing static features of the program's file as well as dynamic features of the process runtime behavior. Our solution targets PE files.

We chose to use deep learning because it allowed us to apply transfer learning and also because it provided better results than other modeling approaches (described in section 5.2).

To aid this approach, we designed the neural network with two branches that process static and dynamic features separately. Results from these two branches are concatenated and then fed to layers responsible for correlating static with dynamic data. The diagram in figure 1 displays this architecture.

There are 2779 static features and 508 dynamic features. These are fed through separate branches of the neural network. A total of 576 values result by concatenating the outputs of the static and dynamic feature branches. These 576 values are fed through fully connected layers of decreasing sizes (576, 384, 192). These were chosen by experimenting with neural layer sizes that are multiples of powers of 2.

In order to address the dying ReLU problem, we experimented with SELU activation. On some layers, this activation function improved the model's performance. SELU is a variant of ReLU that makes possible to compute non-zero gradients on negative values.

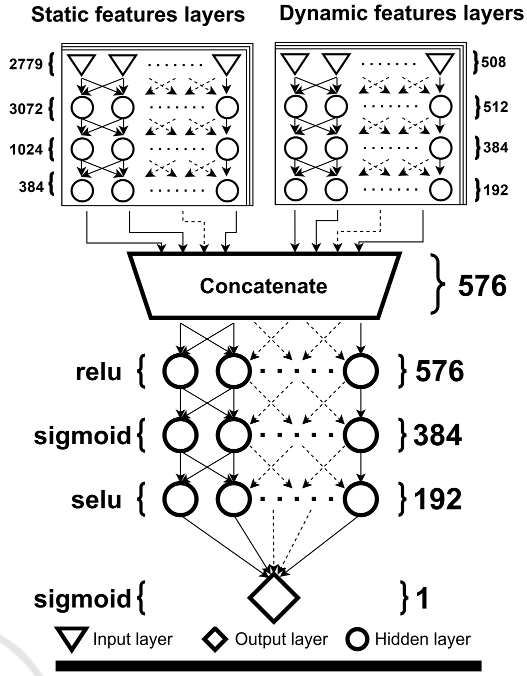

Figure 1: Proposed solution.

By having different sections of the neural network that process static and dynamic features, we are able to train neurons in these two sections separately. This makes it possible to overcome a situation where for training purposes, there is a lack of dynamic data but an abundance of static data (as we will show later on, this is the case).

An important point to make is that our solution could perform well even against fileless malware. Fileless malware infects legitimate programs already present on the endpoint. Our solution could ascertain the expected behavior of a legitimate program based on the static analysis of the executable file and compare it with the actual runtime behavior. Thus, if unexpected suspicious actions occur that are indicative of a fileless malware attack, a detection may be raised with greater accuracy.

4.1 Static Component

The branch that analyzes static features is made up of the first 4 layers of the neural network displayed in figure 2. This neural network was trained as a self-contained model and used to initialize weights and biases in the static feature branch of the final model.

The neural network in figure 2 takes as input static features of a PE file. These features consist of both boolean and numeric values. In order to extract static features we used an AntiMalware engine that analyzes the structure and contents of the file.

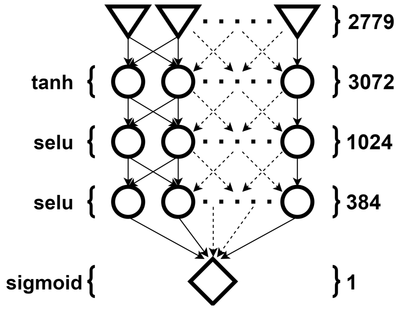

Figure 2: Layers for static features.

Examples of static features that were utilized are found in table 2. Some of the features listed below may be represented as fixed-length collections of numeric or boolean values (for example, the APIs histogram).

Table 2: Examples of static features.

File header characteristics
Byte histogram
Used APIs histogram
Specific library imports
Byte patterns specific to known behavior (for example, decoding memory section with XOR)
Known packer type used by file

4.2 Dynamic Component

Similar to the static component, the branch that analyzes dynamic features is made up of the first 4 layers of the neural network displayed in figure 3. This neural network was trained as a self-contained model and then used to initialize weights and biases in the dynamic feature branch of the final model.

The neural network in figure 3 takes input dynamic features extracted based on the process behavior at runtime. In order to obtain these features we used an EDR security solution (Endpoint Detection and Response) which is installed on the machine and it extracts dynamic features in real-time by monitoring the process at runtime and the entire system at large. The EDR solution correlates a sequence of events representing actions done by the process. Multiple sensors installed on the machine monitor operations and send events to the correlation component. Some of the used sensors are API hooks, network probes and Event Tracing for Windows.

Examples of examined events and the correlation logic used for extracting dynamic features are found in table 3. The resulting dynamic features consist of both boolean and numeric values.

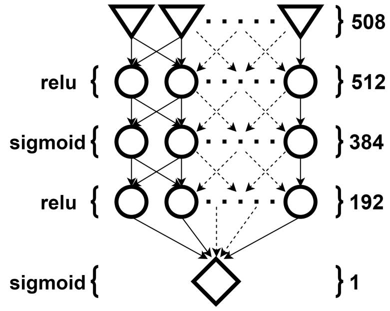

Figure 3: Layers for dynamic features.

Table 3: Examples of runtime events and dynamic features.

Event name	Example of dynamic feature
Process create	A process with suspicious command line arguments was created
Process inject	Code was injected into a running process
Process load module	A suspicious module was loaded through DLL Search Order Hijacking
Process API call	A process called an asymmetric encryption API
Windows Management Instrumentation operation	A process performed a suspicious operation through Windows Management Instrumentation
Registry value write	A value was written in a registry startup key
Network connect	A process made an HTTP connection to an untrusted domain
File create	A file was created that has a name specific to a ransomware note
File delete	Multiple user files were deleted - possible destruction of data

4.3 Evaluation Triggers

The proposed neural network is meant to be integrated into an EDR security product where performance impact is of the essence. This requires an efficient evaluation mechanism for the neural network.

Dynamic features are extracted as the program is running. The evaluation mechanism must decide when to perform a neural network inference in order to detect a malware program as early in its execution as possible. In order not to affect performance, inference should be performed only a limited number of times per period and only when it is relevant.

The diagram in figure 4 displays the evaluation mechanism we have chosen for EXE files. The mechanism for DLL files is similar but evaluation starts after a process has loaded the DLL module.

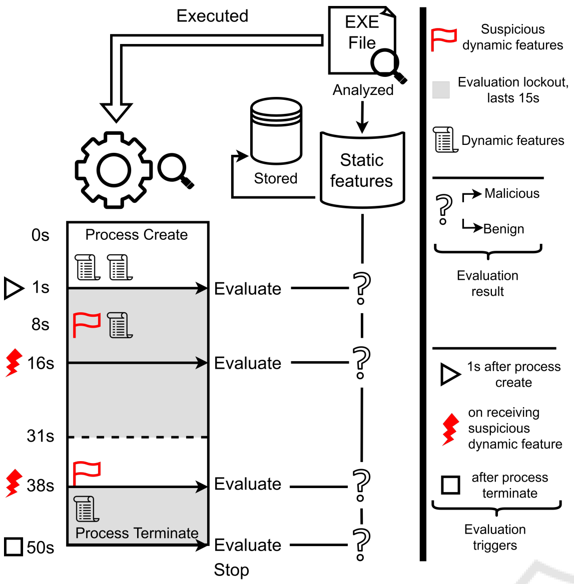

Figure 4: Evaluation triggers and steps.

As shown above, for evaluation to be triggered, static features of the EXE file must be available and a process needs to have started execution based on said EXE file. If those conditions are met, evaluation is triggered in the following cases: one second into the process execution, when a suspicious dynamic feature is received and after the process is terminated.

After each time an evaluation is performed, an evaluation lockout period is put in place for said process in order to prevent a flood scenario that would impact performance. The evaluation lockout period lasts 15 seconds.

The evaluation triggers and overall logic is similar for DLL file analysis.

5 RESULTS

5.1 Database

We used a private dataset containing 5455942 PE files, of those 4068535 are labeled benign and 1387407 are labeled malicious. The reason for the dataset containing 75% benign files is to reflect that on a typical computer, there are many more benign files than malicious ones. As such, this ratio helps us avoid a model prone to false positive alerts.

The static features representing the PE file structure were extracted using a component of an AntiMalware solution. Examples of static features are found in table 2.

There were more than 150000 static features extracted for each file. Much like in (Dahl et al., 2013) we had to reduce the number of features that would be used for the model based on static features. The total uncompressed size of the static data was 5.5TB. In order to perform feature selection on such a large quantity of data in a reasonable amount of time we had to do the selection in multiple steps. These consecutive steps are described by the function *Select*.

<u>function Select</u> $(D, limit, S, c)$;
Input : D dataset
$limit$ the maximum amount of data able to be processed
S selection methods
c maximum length of set A containing relevant features

Output: Set A with the most relevant features

$A = all\_features(D)$;
do
  largest n s.t. $n * len(A) < limit$;
  d = n_random_instances_with_features(D, n, A);
  $k = max(c, len(A)/2)$;
  $A = \{\}$;
  for *select_K_best* **in** S **do**
    $A = A \cup$ select_k_best(d, k);
  end
while $len(A) > c$;

Algorithm 1: Progressively select the best features of the dataset.

The methods that we used for selecting the best static features are the maximum information gain and the importance given by a random forest classifier trained on the dataset.

While there are many sandbox tools that emulate the execution of a PE file, nowadays there are malware programs that employ anti-emulation techniques. Considering that, we decided to run the files on virtual machines with an EDR sensor installed. The execution of the file produces a sequence of events that are correlated by the EDR sensor through various heuristics. These heuristics provide behavior flags representing the dynamic features of a file's execution. Examples of events and dynamic features extracted by the EDR sensor are found in table 3.

To minimize the time required for extracting dynamic features, we developed a system that uses multiple virtual machines to run PE files in parallel.

As displayed in figure 5, each file is run on a vir-

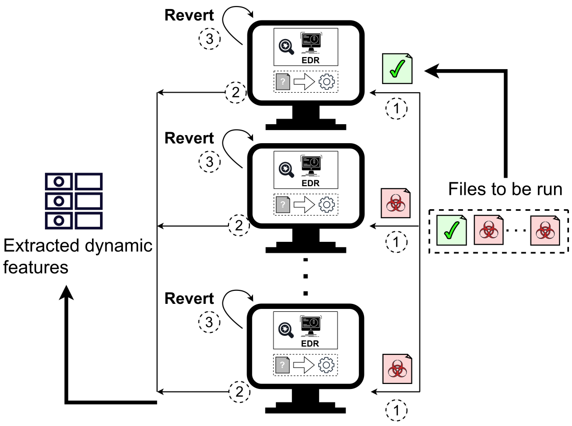

Figure 5: Sample running and dynamic feature extraction system.

tual machine for at most 3 minutes, the EDR sensor extracts dynamic features, then the virtual machine is reverted to an earlier snapshot in order to mitigate any damage done by a malware program. Even with 10 virtual machines running samples in parallel, it would have taken more than 3 years to extract dynamic features for all files. Because of that, we selected in a random uniform manner 13814 files. For 13217 of those we were able to extract dynamic features, 9978 labeled benign and 3239 labeled malicious, thus maintaining the original ratio.

To clean up the dataset, we performed an analysis on samples with static and dynamic data.

C - set of samples correctly classified by either the model using static features or the model using dynamic features

M - set of samples incorrectly classified by both the model using static features and the model using dynamic features

$N_{m,K}$ - set of the K closest neighbors of m from C based on Hamming Distance

$$\Delta_K(m) = \sum_{n \in N_{m,K}} \frac{1}{d_H(m,n)+\varepsilon} \cdot \begin{cases} \gamma, & y_m = y_n \\ 1-\gamma, & y_m \neq y_n \end{cases}$$

$m \in M$; d_H - Hamming Distance; $\gamma = 0.85$
y_m - label of m; y_n - label of n

We chose $\gamma = 0.85$ to minimize the chances of removing a correctly labeled sample from the dataset.

Based on these statistics, we removed 212 samples we found to be possible mislabels. This represents 1.6% of the dataset containing dynamic features and 0.00003% of the dataset containing static features.

In order to visualize the extracted data, we performed a PCA analysis. Figure 6 displays the PCA analysis for static (left side) and dynamic (right side) data.

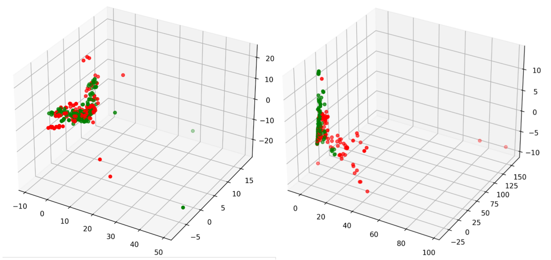

Figure 6: Static data PCA and Dynamic data PCA.

5.2 Training & Inference

The training process was performed separately for the model that uses static features and the model that uses dynamic features. These two neural networks serve as the building blocks to the final model that makes use of transfer learning to better correlate static and dynamic features.

In order to obtain the best results, the values of both static and dynamic features in datasets were normalized such that the standard deviation $\sigma = 1$ and the mean $\mu = 0$.

The neural network for static features was trained for 30 epochs with a variable learning rate starting from $lr = 0.001$ and then decreasing from epoch 5 by $lr = lr \cdot e^{-0.16}$. The loss function used was binary cross entropy. To measure the model's performance we used K-fold cross-validation with 5 folds.

Table 4: Performance of the model using static features.

Metric	Value	STD Dev
FP Rate	0.368%	0.00086
TP Rate	86.746%	0.00195
F1 Score	92.234%	0.00022
Accuracy	96.346%	0.00016

The neural network using dynamic features was trained on 30 epochs with a variable learning rate similar to the model using static features and binary cross entropy loss function. Its performance was measured using K-fold cross-validation with 5 folds.

Table 5: Performance of the model using dynamic features.

Metric	Value	STD Dev
FP Rate	0.716%	0.00155
TP Rate	81.52%	0.01331
F1 Score	88.52%	0.00940
Accuracy	94.832%	0.00476

We wanted to see if we could obtain better performance by training other machine learning algorithms on the dynamic features. We tested the accuracy of both Random forest and XGBoost with 100 estimators against our result obtained with a neural network. As it turns out, Random forest algorithm obtained 93.171% and XGBoost 93.394%, both less than 94.832% that was obtained by our neural network.

The final neural network uses both static and dynamic features and it processes them on two different branches. Neurons on these branches load weights and biases from the previous two models that are already trained, thus making use of transfer learning. At first, only neurons processing the concatenated results from the first two branches are trained, but after a certain epoch and learning rate, all neurons are trained.

We used this scheme in order to make the model learn based on knowledge already gained by the previous two models. In this way, we are able to take full advantage of the large number of instances with static data while also using the statistically representative number of instances with dynamic data. Furthermore, by using this training scheme we were able to reduce overfitting and obtain a considerable performance improvement.

In order to find the best combination of hyperparameters, we used a Bayes search algorithm that tries to minimize a function we chose $f(model) = 1 - F1Score(model)$. Figure 7 displays the value of this function at the number of attempts made by the Bayes search algorithm.

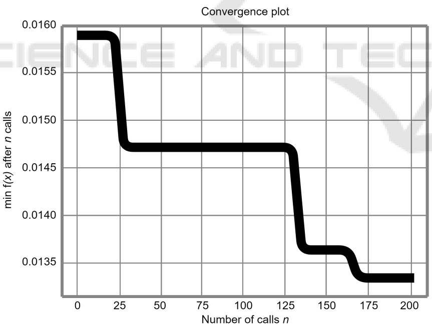

Figure 7: Bayes search graphic.

The neural network that uses both static and dynamic features was trained on 25 epochs with a variable learning rate that decreases starting from epoch 10. When $epoch > 21$ and $lr < 0.00025$ all layers of the network are trained. The loss function used was binary cross entropy. To measure the model's performance we used K-fold cross-validation with 5 folds.

In order to thoroughly analyze the performance of the model that uses static and dynamic features, we also considered the Confusion Matrix for threshold=0.5 and the ROC AUC. The total ROC Area Under the Curve is 0.99896.

Table 6: Performance of the model using static and dynamic features.

Metric	Value	STD Dev
FP Rate	0.26%	0.00147
TP Rate	97.822%	0.00326
F1 Score	98.47%	0.00389
Accuracy	99.302%	0.00143

Table 7: Confusion Matrix of the model using static and dynamic features.

		Predicted	
		Clean	Malware
Ground truth	Clean	10532	34
	Malware	60	3188

Table 8: Analysis of correctly classified samples by combined model.

Element		Accuracy
Correctly classified by analyzing static data	$S \cap C$	98.704%
Improvement by analyzing dynamic data	$(D \setminus S) \cap C$	0.6%
Improvement by correlating both static and dynamic data	$C \setminus (S \cup D)$	0.0144%
Total correctly classified samples by combined model	C	99.319%

6 CONCLUSIONS

Based on the results, we can confidently say that the model using combined features is best suited for practical application. By having the highest TP rate and the lowest FP rate, the combined model is able to raise alerts that offer a high degree of visibility while keeping alert fatigue at a minimum. This offers a way for cyber-security analysts to quickly identify an advanced threat without having to shift through large amounts of false positive alerts.

Taking a closer look, we found that the increase in accuracy was not just comprised of samples correctly identified by either the model using just static features or the model using dynamic features. The model using combined features correctly classified samples that neither of the two previous models did. This provides a way to counteract advanced attacks better.

We tested each model on the dataset containing both static and dynamic features.

Let S, D, and C be the sets of samples correctly labeled by each model (S for the model using static data, D for the model using dynamic data and C for

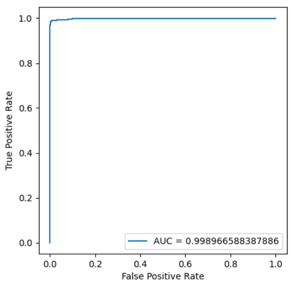

Figure 8: ROC AUC of the model using static and dynamic features.

the model using combined data).

In table 8 we show how using static and dynamic data contributes to the final model performance. Keep in mind that, as accuracy approaches 100%, even small improvements are significant, especially in the field of malware detection.

One disadvantage our solution currently has is that it was trained using just the initial access stage of an attack. While from a protection perspective, this is desired, the EDR philosophy is to provide visibility. In the next iteration, we plan to train the model using events from multiple steps of an attack, from initial access to exfiltration.

REFERENCES

Aghakhani, H., Gritti, F., Mecca, F., Lindorfer, M., Ortolani, S., Balzarotti, D., Vigna, G., and Kruegel, C. (2020). When malware is packin' heat; limits of machine learning classifiers based on static analysis features. *Proceedings 2020 Network and Distributed System Security Symposium.*

Ahmadi, M., Ulyanov, D., Semenov, S., Trofimov, M., and Giacinto, G. (2016). Novel feature extraction, selection and fusion for effective malware family classification.

Carlin, D., Okane, P., and Sezer, S. (2019). A cost analysis of machine learning using dynamic runtime opcodes for malware detection. *Computers & Security*, 85.

Dahl, G. E., Stokes, J. W., Deng, L., and Yu, D. (2013). Large-scale malware classification using random projections and neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3422–3426.

Ijaz, M., Durad, M. H., and Ismail, M. (2019). Static and dynamic malware analysis using machine learning. In *2019 16th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, pages 687–691.

Raff, E., Barker, J., Sylvester, J., Brandon, R., Catanzaro, B., and Nicholas, C. K. (2017a). Malware detection by eating a whole exe. In *AAAI Workshops*.

Raff, E., Sylvester, J., and Nicholas, C. (2017b). Learning the pe header, malware detection with minimal domain knowledge. pages 121–132.

Ronen, R., Radu, M., Feuerstein, C., Yom-Tov, E., and Ahmadi, M. (2018). Microsoft malware classification challenge. *CoRR*, abs/1802.10135.

Santos, I., Devesa, J., Brezo, F., Nieves, J., and Bringas, P. G. (2013). Opem: A static-dynamic approach for machine-learning-based malware detection. In Herrero, Á., Snášel, V., Abraham, A., Zelinka, I., Baruque, B., Quintián, H., Calvo, J. L., Sedano, J., and Corchado, E., editors, *International Joint Conference CISIS'12-ICEUTE´12-SOCO´12 Special Sessions*, pages 271–280, Berlin, Heidelberg. Springer Berlin Heidelberg.

Zhang, S., Wu, J., Zhang, M., and Yang, W. (2023). Dynamic malware analysis based on api sequence semantic fusion. *Applied Sciences*, 13:6526.

Zhao, Z., Zhao, D., Yang, S., and Xu, L. (2023). Image-based malware classification method with the alexnet convolutional neural network model. *Security and Communication Networks*, 2023:1–15.

Zhou, H. (2019). Malware detection with neural network using combined features. In Yun, X., Wen, W., Lang, B., Yan, H., Ding, L., Li, J., and Zhou, Y., editors, *Cyber Security*, pages 96–106, Singapore. Springer Singapore.