

A Categorical Data Approach for Anomaly Detection in WebAssembly Applications

Tiago Heinrich¹^a, Newton C. Will²^b, Rafael R. Obelheiro³^c and Carlos A. Maziero¹^d

¹Computer Science Department, Federal University of Paraná, Curitiba, 81530-015, Brazil

²Computer Science Department, Federal University of Technology, Paraná, Dois Vizinhos, 85660-000, Brazil

³Computer Science Department, State University of Santa Catarina, Joinville, 89219-710, Brazil

Keywords: WebAssembly, WASI Interface, Intrusion Detection, Web Services, Security.

Abstract: The security of Web Services for users and developers is essential; since WebAssembly is a new format that has gained attention in this type of environment over the years, new measures for security are important. However, intrusion detection solutions for WebAssembly applications are generally limited to static binary analysis. We present a novel approach for dynamic WebAssembly intrusion detection, using data categorization and machine learning. Our proposal analyses communication data extracted from the WebAssembly sandbox, with the goal of better capturing the applications' behavior. Our approach was validated using two strategies, online and offline, to assess the effectiveness of categorical data for intrusion detection. The obtained results show that both strategies are feasible for WebAssembly intrusion detection, with a high detection rate and low false negative and false positive rates.

1 INTRODUCTION

WebAssembly, also known as Wasm, is a bytecode-like format that aims to enable the portability of other languages to the Web environment (Battagline, 2021).


It runs in an isolated environment provided by a sandbox and is conceived to easily interact with other languages, through a shared-memory interaction model called *exposed functions*. WebAssembly helps developers with the implementation of Web services, providing a performance gain and code size reduction. WasmEdge and Atmo are popular examples of Web services applications developed using WebAssembly (WasmEdge, 2023; Connor, 2023). WebAssembly had a quick adoption and is supported by all the mainstream Web browsers. Its current stable version is 1.0, and it is being actively developed to add new features.


Wasm applications are executed inside a sandbox environment that offers a layer of isolation between the application and the underlying environment. However, security issues exist and can affect Web users (Kim et al., 2022). Attackers are also exploring Wasm


features to make cryptojacking and to obfuscate malicious applications (Musch et al., 2019; Romano et al., 2022).


Proposals for security improvement developed in the latest years consider the classification of Wasm binaries (Romano and Wang, 2020), hardening the sandbox by the use of Trusted Execution Environments (TEEs) (Qiang et al., 2018; Ménétrey et al., 2021), using fuzzing techniques to find flaws in software implementation (Liang et al., 2018), and dynamic analysis tools that extract information from binaries to find vulnerabilities (Lehmann and Pradel, 2019; Brito et al., 2022). However, to the best of our knowledge, solutions that explore the interactions between the application and the sandbox environment to detect anomalous application behavior were not proposed yet.

The use of data extracted from Inter-Process Communication (IPC) for anomaly detection is not new (Forrest et al., 1996; Castanhel et al., 2021; Lemos et al., 2022). This approach allows to observe the operations being executed by the application, capturing interaction patterns that can be used by Machine Learning (ML) models to identify malicious behavior. Our proposal explores the interactions between the WebAssembly application and its runtime for intrusion detection.

^a <https://orcid.org/0000-0002-8017-1293>

^b <https://orcid.org/0000-0003-2976-4533>

^c <https://orcid.org/0000-0002-4014-6691>

^d <https://orcid.org/0000-0003-2592-3664>

The approach proposed here consists in extracting data from the interactions between a Wasm application and its runtime and classifying such data using a categorical model. This pre-processed data is then used to train ML models to capture the application behavior. Finally, the trained models are used for intrusion detection. We explore two detection strategies (online and offline) with different models of ML for the detection of threats.

To the best of our knowledge, we present the first proposal to use dynamic interaction data for intrusion detection in the WebAssembly environment. Our main contributions are:

- An intrusion detection solution for WebAssembly based on interaction data;
- A comparison between online and offline approaches for the proposed intrusion detection;
- A generic categorical data model that explores interaction data to detect anomalies; this model is generic enough to be applied to other situations, like OS system calls and Android binder calls; and
- An overview on the use of categorical data in security approaches that explore machine learning solutions.

The remainder of this paper is structured as follows: Section 2 presents the background; Section 3 presents related works; Section 4 discusses the proposal; Section 5 presents the evaluation; and Section 6 concludes the paper.

2 BACKGROUND

This Section presents a brief overview about the concepts used in this work. It contextualizes the WebAssembly environment, the use of categorical data approaches, and intrusion detection using Machine Learning (ML).

2.1 WebAssembly

WebAssembly (also known as Wasm) is a bytecode binary format as a target. It presents a performance gain and reduced size in comparison with Web languages. Programs in WebAssembly can be written using WebAssembly Text (WAT), a textual programming format that is compiled into the bytecode. Besides that, developers can use languages such as Rust, C, and C++, which are not natively supported in the Web environment, but can also be compiled into Wasm binaries. This allows to easily port code developed in such languages to the Web environment. Such features

open new opportunities and easiness for developers (Hoffman, 2019).

The WebAssembly binary contains a module that encapsulates metadata (like the the WebAssembly version), functions, variables, information about exported functions that can be reached by other applications, or even imported resources (Battagline, 2021). A WebAssembly application executes inside a sandbox environment, with external resources being accessible through an Application Programming Interface (API) (Kim et al., 2022). This design choice allows applications to run on a wide range of platforms, reduces the size of messages exchanged with the server to retrieve content, and offers more security for the environment when running unknown binaries (Battagline, 2021).

Figure 1 presents the WebAssembly environment, with three of the most popular ways that WebAssembly applications can be used as a service. The three options presented from the left to the right are, (i) a runtime inside a host (ii) a container environment, and (iii) inside the Web browser. The application interacts with the underlying environment through the use of the WebAssembly System Interface (WASI) calls that passes through the WASI API. The interface defines rules and controls the interactions that are performed between the application and the environment. This layer allows the execution of native operations without breaking the security levels already defined for WebAssembly applications (Battagline, 2021).

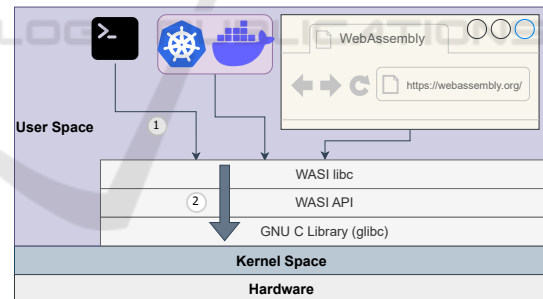


Figure 1: WebAssembly architecture overview.

The interaction with the API is made through WASI calls, which have behavior similar to system calls, but in a more abstract layer, closer to the application. They allow a WebAssembly application running inside the sandbox to access resources provided by the underlying environment. Currently, 45 distinct WASI calls are supported (more calls will be added in future versions) (BytecodeAlliance, 2021a).

Despite the similarity between WASI calls and system calls, WASI calls are made specifically for the WebAssembly environment, not having a 1:1 binding to system calls. This allows to offer WASI calls more abstract and better suited to build Web applications, in

opposition to system calls, which should be closer to kernel resources.

2.2 Categorical Data

Categorical data or categorical variables are defined as a data type that can represent a known number of variables by a limited number of categories (Powers and Xie, 2008). Quantitative and qualitative variables can usually be represented by categories (except continuous variables). The use of data categorization as a measurement strategy is popular in the social sciences field, representing data types such as age, gender, and location.

Categorical representation in specific situations can be a better fit to represent data that would be oversimplified by numerical representations. However, both representations are not that different, the key difference is that the probability distribution of the first one is associated with categories instead of numerical data (CHARU, 2017).

The use of categorical data representation also limits the strategies that can be applied for the data evaluation. Statistical algorithms, linear models, proximity-based algorithms, and density-based models are examples of techniques that should be adapted to correctly work with categorical data (CHARU, 2017).

Categorical data analysis is a widely explored field. (Bay and Pazzani, 1999) present a strategy for mining categorical data, proposing a methodology for the identification of contrasting groups. Also, (Das et al., 2008) proposes a range of solutions focusing on pattern detection; anomaly detection is covered by (Liao and Vemuri, 2002; Taha and Hadi, 2019), and outlier detection for categorical data by (Wu and Wang, 2011; Li et al., 2018).

In our proposal, the use of categorical representations for the data allows us to (i) evaluate the use of this data representation for intrusion detection solutions, and (ii) have a better understanding of categorical data representations for machine learning strategies. Data categorization is important because it allows using a representation that can highlight new patterns for the samples by the use of categories or groups.

2.3 Intrusion Detection Using Machine Learning

Intrusion Detection System (IDS) is a mechanism that aims to warn about activities that may put the system security at risk (Axelsson, 2000). The mechanism can monitor the network, hosts, and applications, looking for signs of attacks or intrusions (Stallings et al., 2012). When an evidence of an attack is observed, an alert

is issued for further analysis, and in some cases, the system itself can respond and take countermeasures.

An IDS tries to distinguish different behaviors and patterns to identify unwanted actions. Two strategies are popularly used to identify threats (Lam, 2005): *Signature-based*, which uses a database of known attack samples to identify unwanted behavior; these samples are signatures of threats or known attack behavior. On the other hand, an *anomaly-based* IDS focuses on using statistical models to characterize the normal/usual activity in the system/environment. Thus, an anomaly portrays some activity that deviates from the defined normal model (Debar et al., 1999; Yassin et al., 2013).

Over the years, the adoption of Machine Learning (ML) solutions to improve security strategy increased. Intrusion Detection Systems (IDSs) can be improved by the use of models that allow an application to learn the patterns of attacks by itself. ML models are capable of creating a general representation of the environment, identifying key differences between types of attacks, and, for some specific models, continuously learning during the time of evaluation (Mishra et al., 2018; Ceschin et al., 2020).

3 RELATED WORK

There are some few research works targeting WebAssembly binary code analysis. Most of them focus on feature extraction from the binary code to identify vulnerabilities and other problems.

Wasabi is a framework for dynamic analysis of WebAssembly binaries (Lehmann and Pradel, 2019). It allows the analysis of instructions, call graphs, taint and tracing. The information is collected during the execution and is not limited to WebAssembly applications. An API allows new evaluation systems to be implemented and enables the extraction of information from the framework.

The paper (Romano and Wang, 2020) presents WASim, a tool that performs the classification of WebAssembly binaries. A dataset for model training and evaluation was built using binaries extracted from the top 1 million sites in the Alexa ranking. Their prediction strategy achieved an accuracy of 91.6%.

Wasmati is a tool for static analysis of WebAssembly binaries (Brito et al., 2022), aiming at identifying code vulnerabilities. Property graphs are generated for each application based on 10 queries, and a set of datasets was scanned, looking for vulnerabilities.

Another strategy for static analysis of WebAssembly binaries is presented by (Stiévenart and De Roover, 2020). Flows of information are extracted for the bi-

nary code, allowing the evaluation of function calls in WebAssembly.

4 PROPOSAL

Our proposal consists of using data extracted from WASI calls issued by the application during its execution, to detect anomalies. The extracted data is classified using a categorical data strategy and then used to train a machine learning model for detecting such threats.

Two strategies to evaluate the proposal were defined. The first strategy consists of an offline evaluation, that is aimed at the classification of WebAssembly binaries. The second strategy is an online evaluation, that aims to simulate a real case of intrusion detection, where the application would be running. Both solutions are aimed at identifying intrusion detection in WebAssembly applications.

The categorization strategy we propose is presented in Section 4.1, the dataset built for the evaluation is described in Section 4.2, and the proposal evaluation itself is presented in Section 5.

4.1 Data Categorization

Instead of directly using WASI calls as features for the machine learning modes, our proposal is based on the data categorization. A variety of approaches can be applied to the data, as the categorization allows to highlight new characteristics from the dataset. For intrusion detection, the security risk associated to each call can be highlighted through the use of categorization.

Categorical data are variables that are measured by categorizing a limited set of “values” (Powers and Xie, 2008). This is a basic data type used for analysis (CHARU, 2017). Discrete, ordinal, or nominal variables can be represented as categories. For security and machine learning, this concept is applied with limitations. The application of data categorization strategies requires understanding the data to group the variables in such a way that key points are represented in the categories and information on their respective members is not lost (Markman and Ross, 2003).

The system calls taxonomy proposed by (Bernaschi et al., 2002) is a pioneer work considering system call grouping. Through a categorization strategy they define two groups for the classification of system calls. The first group aims to classify system calls according to their functionality, and the second group defines a threat level for each system call. The threat levels are 1 (enable full system compromise), 2

(allow a denial-of-service attack against the system), 3 (enable subversion of the calling process), and 4 (harmless calls). This classification is hierarchical, as it assumes that a system call classified at level i can also perpetrate an attack at threat level j for $i < j$. The first issue is treating threat levels as an hierarchy, with threats at lower levels subsuming threats at higher levels, which may not always be true. A counter example would be the `fork` system call, which can cause a denial-of-service (DoS) on the system (and is thus classified at threat level 2) but cannot subvert a process (threat level 3).

In our proposal, we aim to remove hierarchical problems from the classification defined by (Bernaschi et al., 2002) and propose a categorical definition that is flexible for using with other data types. A range of communication mechanisms are used in the Operating System (OS), such as system calls (Galvin et al., 2003), binder calls (Lemos et al., 2023), or IPC, and IDSs analyzing them can benefit from exploring categorization.

Each WASI call has specific information related to its operation, functionality, and security. Each interaction (or operation) that an application performs with the system can be categorized. Also, each interaction also has a respective risk associated to it. Table 1 presents two possible categorizations based on the operation that the application generates when interacting with the system. Two categories are presented, a level for the type of operation and a risk level for the operations.

The five risk levels presented aim to group interactions (that in our proposal are WASI calls) that alone offer some kind of risk to the system if used by a malicious application. When considering the defined risk group (high or low), we tend to directly relate it to the type of operation that the interaction performs in the system. The first three risk levels (A, B, and C), in the *high risk* group, are interactions that can be used by an attacker. Such risk levels define different behavior of operations that malicious applications have on the environment.

We also propose a classification based on the functions performed by each interaction with the runtime (also WASI calls). Such classes are shown in Table 2. This classification is an expansion of categorizations already proposed, such as (Bernaschi et al., 2002) and (Galvin et al., 2003). We added to it the device manipulation (10) and the removed/debug (9) interaction groups, since several changes were observed in the system APIs.

We assume that an application making calls classified in class A more frequently tends to offer more risk than an application that only uses calls presented in class D. This classification is not used alone in the

Table 1: Operation Classification.

Risk Level	Risk Group	Type of operation
A	High	These calls alone offer system risk/control
B		Calls that may offer some risk to the system
C		Calls that are dangerous when much repeated
D	Low	Harmless calls
E		Unused/deprecated calls

Table 2: Functionalities Classification.

Group	Functionalities
1	File manipulation
2	Process control
3	Module management
4	Memory management
5	Time operation
6	Communication
7	System information
8	Reserved
9	Not implemented/removed/debug
10	Device manipulation

intrusion detection process, but together with the classification of functionalities. In this way, an application that makes calls classified as A-2 (a process control call that offers a high risk) offers more risk than if it makes calls classified as C-5 (time-related calls that offer some risk). This distinction contributes to the training stage of the ML models, as it is possible to emphasize calls that pose more risk.

Table 3 presents the proposed data categorization for the WASI calls. WebAssembly is a format in active development, thus we are using the calls supported for version 1.0. In future versions (`snapshots`) new calls will be introduced, but this proposal already has the required categories for the correct classification of such calls.

The advantages of the categorization proposed here are (i) it is flexible enough to be applied to other data types; (ii) it reduces the volume of data used for training and identification; (iii) it allows a better understanding of the behaviors behind application interactions; and (iv) it allows to relate different types of interactions.

4.2 Data Approach

The dataset used for the training and testing of our proposal was created by us from different data sources, allowing us to build a dataset that represents a variety of behavior found in the WebAssembly environment. Overall, we have 18 types of attack samples present, with a total of 377 samples to form the anomaly class,

and we have 263 samples for the normality class. The data samples came from a range of data sources like test suits, benchmarks, single WebAssembly modules, and applications (Stiévenart, 2023; Denis, 2023; WebAssembly, 2023; Beyer, 2023).

Our goal is to have a varied set of samples that represents the environment. The variety of the data samples allows us to better assess how the data categorization proposed is able to highlight the functionality and operations behind each interaction of the application with the underlying environment. The samples are also helpful to demonstrate that the proposal categorization is generic and may be applied to other data sources.

For the extraction of the WASI calls in each binary sample, we used the *Wasmtime* runtime CLI (BytecodeAlliance, 2021b), which executes standalone. It provides functionality to generate detailed traces of the WebAssembly application being run, without instrumenting or modifying the application.

5 EVALUATION

To evaluate the proposal, we built the classification strategy, collected data to conduct the experiments, and train/test the machine learning models. This allows us to evaluate the efficiency of our intrusion detection solution, based on the categorization of calls from the WebAssembly environment.

Two types of strategies are possible for intrusion detection: an *online evaluation* is made in real-time and uses the data generated during the runtime of an application to identify threats. On the other hand, an *offline evaluation* is not made in real-time and is quite useful in a controlled environment, enabling the study of the entire application and its interactions.

We defined two groups of tests with the goal of evaluating both scenarios, in how effectively the use of calls from the WebAssembly environment is for intrusion detection solutions. The first scenario considers an offline approach and uses all the interactions from the environment. The second scenario used a fixed size sliding window, defining a partial view of the environment, and simulating an online approach

Table 3: WASI Calls classification.

#	#	WASI Calls
A	1	<code>path_link, path_rename, path_symlink, path_unlink_file, path_remove_directory, path_filestat_set_times, args_get, environ_get</code>
B	1	<code>fd_fdstat_set_flags, fd_tell, fd_seek, path_create_directory, fd_pread, fd_pwrite, sock_recv, fd_read, sock_send, fd_write, fd_filestat_get, path_create_directory</code>
	2	<code>fd_advise, sched_yield</code>
C	1	<code>fd_renumber, fd_allocate, path_open, random_get</code>
	6	<code>sock_recv, sock_send, proc_raise</code>
D	1	<code>fd_close, path_readlink, path_filestat_get, args_sizes_get, environ_sizes_get, fd_prestat_get, fd_prestat_dir_name, fd_readdir</code>
	5	<code>clock_res_get, clock_time_get</code>
	7	<code>sock_shutdown</code>
	9	<code>fd_filestat_set_times</code>
	10	<code>fd_datasync, fd_sync, fd_fdstat_get</code>
E	-	

for intrusion detection solutions.

In both scenarios, the interactions from the environment were treated equally. Using the same classification process for the calls of the WebAssembly environment. This way, we are able to evaluate the effectiveness of the use of this type of interaction for intrusion detection in WebAssembly and discuss how this type of solution based on the use of categorical data may be used in other solutions.

Overall, we selected six ML models for the experiment, to study the impact of the proposal in different model approaches. These models are from a range of supervised learning algorithms, encompassing decision trees classifier, ensemble methods, and neural network models. The models were used with their default parameters.

Section 5.1 presents the offline evaluation, discussing the detection of each model. Section 5.2 presents the online evaluation. Both strategies are discussed in-depth, with details on how the machine learning models behave. Section 5.3 presents a discussion of how the use of calls (communication between the application and the runtime) can be used for intrusion detection, and what we learnt from our evaluation of such data.

5.1 Offline Evaluation

In the offline evaluation, all the information collected during an application execution is used by the models during the classification phase. Table 4 presents the results obtained for each model. The dataset was split in half for the training and test phases. The overall result is promising for the strategy, with all models achieving an *f1score* above 80%.

The **precision** shows the false positive impact in the models. Although we obtained high precision val-

ues, our models are still impacted by a small fraction of the samples being misclassified and generating false positives. SGB is the exception in all metrics, having a low performance overall. The **recall** doesn't depend on precision and is influenced by false negative samples, which do not impact our models. The **f1score** is based on precision and recall. This metric enables the description of the positive classes, describing the adequacy of the models to classify the normal behavior and the detection of the intrusion classes. In these three metrics, we are not considering the impact of true negative samples.

The **accuracy** and **balanced accuracy** (BAC) metrics show a better understanding of the true positive and true negative classifications. Despite not considering the false negative/positive classes, we notice that our results are similar to the previous results found for the *f1score*, showing that our models are being able to classify correctly most of the samples. The similarity between accuracy and BAC results also shows that the dataset used for the training and test is balanced.

The lower the **Brier score** is, the most calibrated the models are to make the classification. This evaluation strategy enables the measurement between the predicted probability of a sample and the achieved result. Only two models present a brier score higher than 2% and only one of such models present a poor overall result.

The *Stochastic Gradient Descent* (SGD) is a linear classifier, which in our case is being impacted by a variety of samples found in the dataset. With a high number of classes that come from a variety of samples, the linear model is not being able to correctly distinguish between the two groups, representing a model that is not adequate for intrusion detection with this type of data.

The similarity between *Decision Tree* and *Ad-*

Table 4: Performance of the models for the offline strategy.

Classifier	Precision	Recall	F1Score	Accuracy	BAC	Brier Score
XGBoost	98.36%	100%	99.18%	99.06%	98.91%	0.94%
Decision Tree	98.38%	100%	99.18%	99.06%	98.91%	0.94%
Nu-Support Vector	92.86%	100%	96.30%	95.61%	94.89%	4.39%
MLP	96.81%	100%	98.38%	98.12%	97.81%	1.88%
AdaBoost	98.38%	100%	99.18%	99.06%	98.91%	0.94%
SGD	73.09%	89.56%	80.49%	75.24%	72.88%	24.76%

aBoost results (over even *XGBoost*), is due to the proximity of the strategies used in such models (Hastie et al., 2009; Molnar, 2020). They are also popular for intrusion detection and malware detection, enabling users to study the decision made by the classifier.

Overall, our results are promising for the proposal for intrusion detection. Most of the evaluated models were able to classify correctly both classes, with few samples being misclassified. The strategy for offline detection is quite adequate using the classification and calls between the WebAssembly application and its runtime.

5.2 Real-Time Evaluation

For the online proposal, we have the objective in identify threats during execution time. A new model was trained and tested. However, we simulated a limited vision for the models using a fixed size sliding window, using two strategies: *non-overlapping* sliding windows (*abc, def, ghi, ...*) and *overlapping* sliding windows (*abc, bcd, cde, ...*).

With a sliding window we can understand the impact of our proposal in a real-time intrusion detection strategy where only small portions of the execution interactions are available, since the application is still running. In comparison with the previous solution, the same categorical classification is used, with the only difference being the use of the sliding window.

The sliding window size was defined as three (3), based on previous research using this value, the size of the traced applications, and characteristics observed from WebAssembly applications (Liu et al., 2018; Castanhel et al., 2020). WebAssembly applications are limited to a module, a limited number of types exist in the format, and its design limits the number of operations available. In consequence, the size of an application is quite smaller than what would be found in other languages, and it generates a smaller amount of calls to the runtime. Such reasons led to choose a small window size.

Table 5 presents the results of the online experiments with a non-overlapping sliding window. Overall, the restriction on the information amount provided to

the models impacts the detection efficacy. However, the precision was reduced in four of the classifiers and the recall reduced for all of them, directly affecting the f1score. The f1score and accuracy for these models indicates a reduction in the detection rate, leading to an increase of false positives and false negatives. This behavior was expected because we are trying to detect threats with only partial information being provided to the classifiers.

Two of our classifiers in Table 5 achieve a better result for precision than the offline strategy, showing a reduction in the number of false positives. However, the same models have an increase in the number of false negatives (as the recall shows a reduction in comparison with Table 4), and the accuracy describes an impact in both classes (negative/positive detection rate are lower than the previous solution).

With BAC and Brier score it is clear that the models were impacted and had an increase in the number of misclassified samples. However, Table 5 shows a small reduction in comparison with the offline approach; we consider such results acceptable. The classifiers are still able to detect threats despite misclassifying some samples.

Considering an online (i.e. real-time) intrusion detection, the f1score metric above 93% obtained in four of the classifiers is considered an adequate result. The low score obtained by the SGD classifier was expected; as presented in Table 4, such linear model seems not well-suited to this kind of data.

Table 6 presents the results for the overlapping sliding window with overlap 1 (*abc, cde, efg, ...*). We included these results as it is quite popular when using sliding windows to consider overlapping to simulate a real-world case and to increase the amount of data, since more windows are created. We notice an increase in the number of false positives, despite the reduction in the number of false negatives. The models had an inferior performance as demonstrated by the f1score result. The similar values of accuracy and BAC between Tables 5 and 6 are not enough to suggest that the use of overlapping is a good strategy when using categorical data.

Giving a deeper look at the data, we can better

Table 5: Performance of the models for the real-time strategy (non-overlapping).

Classifier	Precision	Recall	F1Score	Accuracy	BAC	Brier Score
XGBoost	94.69%	92.88%	93.78%	94.90%	94.60%	5.10%
Decision Tree	99.86%	92.88%	93.86%	94.97%	94.66%	5.03%
Nu-Support Vector	93.70%	92.88%	93.29%	94.46%	94.23%	5.54%
MLP	93.7%	92.88%	93.29%	94.46%	94.23%	5.54%
AdaBoost	88.20%	96.01%	91.94%	93.03%	93.46%	6.97%
SGD	32.81%	68.58%	44.38%	28.83%	34.66%	71.17%

Table 6: Performance of the models for the real-time strategy (overlapping).

Classifier	Precision	Recall	F1Score	Accuracy	BAC	Brier Score
XGBoost	82.44%	100%	90.37%	92.25%	93.91%	7.75
Decision Tree	82.44%	100%	90.37%	92.25%	93.91%	7.75
Nu-Support Vector	72.17%	100%	83.83%	85.96%	88.97%	14.04
MLP	81.47%	99.84%	89.73%	91.68%	93.43%	8.32
AdaBoost	82.54%	100%	90.44%	92.30%	93.95%	7.7
SGD	71.97%	99.84%	83.65%	85.80%	88.80%	14.2

understand why the overlapping approach is not ideal for this type of data. Since our interactions that are WASI calls are being categorized, when the overlapping strategy is adopted we duplicated some of the information. In this case, we are adding more categories to the window generated from the trace of the application, because we have an overlap of one (that adds a category in each window).

In all the categories, most groups present a similar number of calls, except for three classes. Two of them are from the high-risk level and present growth in the number of calls for C1 (87.4%) and B1 (23.1%). Class D5 also presented a growth of 37.2%. The increase in the false positive rate of our models is directly associated with the increase in the number of calls in the categories with high risk. As we are categorizing the information used, in the case of overlapping we are also adding information to the dataset that is incorrect. Since we are duplicating small portions of the data, we are also adding information related to operations, functionality, and security of the application that in the reality do not exist. For example, with a growth of 87.4%, we are also telling the models that the risk is higher in comparison with the previous experiment (Table 5), which is not exact, because the risk is the same.

The impact in machine learning models also exists because of the categorization, we are specifying distinct features that with an overlapping strategy are misleading the models. For this reason, we cannot recommend the use of categorization and overlapping in the same data.

5.3 Use of Calls for Intrusion Detection

The use of communication data for intrusion detection is not new (Forrest et al., 1996; Lemos et al., 2023). However, an approach that considers this type of interaction in the WebAssembly sandbox is a novelty. Considering such data as categorical is also a new strategy for intrusion detection in this context. Our results highlight how the use of this type of strategy is able to represent properly the characteristics found in the applications on the dataset.

The categorization of variables enables the user to highlight key factors in the data, that otherwise may be difficult to identify by machine learning models. In our case, we were able to highlight security characteristics for the WASI calls, and through the grouping of operations/functionality, we were able to introduce new features that present more complete information about the behavior of each application.

The data categorization also presented good results for both online/offline solutions.

For the WebAssembly sandbox, the WASI calls are representative and, with our experiments, we can say that our proposal is a viable solution for intrusion detection solutions for the Web. Considering that different sandbox solutions exist, this proposal can be adapted to other environments. The categorical classification presented is flexible and easily applicable to other types of communication as system calls and IPC mechanisms.

We applied a range of machine learning classifiers intending to evaluate the efficacy obtained by exclusively using categorical data for intrusion detection.

As previously discussed, the models do not appear to be harmed by this design choice and we can go further by saying that the use of categorical data to emphasize specific features of the data helps the efficacy of the models.

6 CONCLUSION

This paper presents an intrusion detection solution for WebAssembly. We use data from the interactions between the WebAssembly application and the sandbox runtime to identify malicious applications. Such interactions (WASI calls) were categorized with the goal of highlighting key features for the machine learning classifiers.

We proposed a categorical classification to be used for intrusion detection, to be applied on the WASI calls data. The operations and functionalities were the main aspects used for the classification. The result was a generic model that is not limited to WebAssembly applications, enabling the categorization of system calls or IPC messages.

Our results show that for both online/offline solutions, the use of WASI calls is adequate for intrusion detection. Most of our models achieve a high detection rate with low false negative and false positive values. The models in both cases were able to learn from the categorical features offered in the training phase. The results showed adequacy in the use of data categorization for intrusion detection, allowing the discussion of the effectiveness of using categorical data for machine learning classification strategies in intrusion detection.

For future work, we will improve the detection strategy for WebAssembly in a more complex environment, not being limited to Web services. We will also explore further the benefits of using data categorization in machine learning models. We also aim to explore different data types that can be used for intrusion detection and may provide performance improvements when using data categorization.

ACKNOWLEDGEMENTS

This study was financed in part by the *Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001* and *Fundação de Amparo à Pesquisa e Inovação do Estado de Santa Catarina (FAPESC)*. The authors also thank the UDESC, UFPR and UTFPR Computer Science departments.

REFERENCES

- Axelsson, S. (2000). Intrusion detection systems: A survey and taxonomy.
- Battagline, R. (2021). *The Art of WebAssembly: Build Secure, Portable, High-Performance Applications*. No Starch Press, San Francisco, CA, USA.
- Bay, S. D. and Pazzani, M. J. (1999). Detecting change in categorical data: Mining contrast sets. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 302–306.
- Bernaschi, M., Gabrielli, E., and Mancini, L. V. (2002). Remus: A security-enhanced operating system. *ACM Transactions on Information and System Security (TIS-SEC)*, 5(1):36–61.
- Beyer, C. (2023). Amalgamated webassembly system interface test suite. <https://github.com/caspervonb/wasi-test-suite>.
- Brito, T., Lopes, P., Santos, N., and Santos, J. F. (2022). Wasmati: An efficient static vulnerability scanner for WebAssembly. *Computers & Security*, 118:102745.
- BytecodeAlliance (2021a). Wasmtime. <https://github.com/bytecodealliance/wasmtime/blob/main/docs/WASI-overview.md>.
- BytecodeAlliance (2021b). Wasmtime. <https://docs.wasmtime.dev/introduction.html>.
- Castanhel, G. R., Heinrich, T., Ceschin, F., and Maziero, C. (2021). Taking a peek: An evaluation of anomaly detection using system calls for containers. In *Proceedings of the IEEE Symposium on Computers and Communications*, pages 1–6. IEEE.
- Castanhel, G. R., Heinrich, T., Ceschin, F., and Maziero, C. A. (2020). Sliding window: The impact of trace size in anomaly detection system for containers through machine learning. In *Anais da XVIII Escola Regional de Redes de Computadores*, pages 141–146. SBC.
- Ceschin, F., Gomes, H. M., Botacin, M., Bifet, A., Pfahringer, B., Oliveira, L. S., and Grégio, A. (2020). Machine learning (in) security: A stream of problems. *arXiv preprint arXiv:2010.16045*.
- CHARU, C. A. (2017). *OUTLIER ANALYSIS*. Springer.
- Connor (2023). Atmo. <https://www.civo.com/marketplace/atmo>.
- Das, K., Schneider, J., and Neill, D. B. (2008). Anomaly pattern detection in categorical datasets. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 169–176.
- Debar, H., Dacier, M., and Wespi, A. (1999). Towards a taxonomy of intrusion-detection systems. *Computer Networks*, 31(8):805–822.
- Denis, F. (2023). webassembly-benchmarks. <https://github.com/jedisct1/webassembly-benchmarks>.
- Forrest, S., Hofmeyr, S. A., Somayaji, A., and Longstaff, T. A. (1996). A sense of self for unix processes. In *Proceedings 1996 IEEE Symposium on Security and Privacy*, pages 120–128. IEEE.
- Galvin, P. B., Gagne, G., Silberschatz, A., et al. (2003). *Operating system concepts*, volume 10. John Wiley & Sons.

- Hastie, T., Tibshirani, R., Friedman, J. H., and Friedman, J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer.
- Hoffman, K. (2019). Programming webassembly with rust: unified development for web, mobile, and embedded applications. *Programming WebAssembly with Rust*, pages 1–220.
- Kim, M., Jang, H., and Shin, Y. (2022). Avengers, Assemble! survey of WebAssembly security solutions. In *Proceedings of the 15th International Conference on Cloud Computing*, pages 543–553, Barcelona, Spain. IEEE.
- Lam, A. (2005). New ips to boost security, reliability and performance of the campus network. *Newsletter of Computing Services Center*.
- Lehmann, D. and Pradel, M. (2019). Wasabi: A framework for dynamically analyzing WebAssembly. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1045–1058, Providence, RI, USA. ACM.
- Lemos, R., Heinrich, T., Maziero, C. A., and Will, N. C. (2022). Is it safe? identifying malicious apps through the use of metadata and inter-process communication. In *Proceedings of the 16th IEEE International Systems Conference*, pages 1–8. IEEE.
- Lemos, R., Heinrich, T., Will, N. C., Obelheiro, R. R., and Maziero, C. A. (2023). Inspecting binder transactions to detect anomalies in android. In *Proceedings of the 17th Annual IEEE International Systems Conference*, Vancouver, BC, Canada. IEEE.
- Li, J., Zhang, J., Pang, N., and Qin, X. (2018). Weighted outlier detection of high-dimensional categorical data using feature grouping. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 50(11):4295–4308.
- Liang, H., Pei, X., Jia, X., Shen, W., and Zhang, J. (2018). Fuzzing: State of the art. *IEEE Transactions on Reliability*, 67(3):1199–1218.
- Liao, Y. and Vemuri, V. R. (2002). Using text categorization techniques for intrusion detection. In *USENIX Security Symposium*, volume 12, pages 51–59.
- Liu, M., Xue, Z., Xu, X., Zhong, C., and Chen, J. (2018). Host-based intrusion detection system with system calls: Review and future trends. *ACM Computing Surveys*, 51(5):98.
- Markman, A. and Ross, B. (2003). Category use and category learning. *Psychological bulletin*, 129:592–613.
- Ménétreay, J., Pasin, M., Felber, P., and Schiavoni, V. (2021). Twine: An embedded trusted runtime for WebAssembly. In *Proceedings of the 37th International Conference on Data Engineering*, pages 205–216, Chania, Greece. IEEE.
- Mishra, P., Varadharajan, V., Tupakula, U., and Pilli, E. S. (2018). A detailed investigation and analysis of using machine learning techniques for intrusion detection. *IEEE communications surveys & tutorials*, 21(1):686–728.
- Molnar, C. (2020). *Interpretable machine learning*. Lulu.com.
- Musch, M., Wressnegger, C., Johns, M., and Rieck, K. (2019). New kid on the Web: A study on the prevalence of WebAssembly in the wild. In *Proceedings of the 16th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 23–42, Gothenburg, Sweden. Springer.
- Powers, D. and Xie, Y. (2008). *Statistical methods for categorical data analysis*. Emerald Group Publishing.
- Qiang, W., Dong, Z., and Jin, H. (2018). Se-Lambda: Securing privacy-sensitive serverless applications using SGX enclave. In *Proceedings of the 14th International Conference on Security and Privacy in Communication Systems*, pages 451–470, Singapore. Springer.
- Romano, A., Lehmann, D., Pradel, M., and Wang, W. (2022). Wobfuscator: Obfuscating JavaScript malware via opportunistic translation to WebAssembly. In *Proceedings of the 43rd Symposium on Security and Privacy*, pages 1574–1589, San Francisco, CA, USA. IEEE.
- Romano, A. and Wang, W. (2020). Wasim: Understanding WebAssembly applications through classification. In *Proceedings of the 35th International Conference on Automated Software Engineering*, pages 1321–1325, Melbourne, Australia. IEEE.
- Stallings, W., Brown, L., Bauer, M. D., and Bhattacharjee, A. K. (2012). *Computer security: principles and practice*. Pearson Education Upper Saddle River, NJ, USA.
- Stiévenart, Q. and De Roover, C. (2020). Compositional information flow analysis for WebAssembly programs. In *Proceedings of the 20th International Working Conference on Source Code Analysis and Manipulation*, pages 13–24, Adelaide, Australia. IEEE.
- Stiévenart, Q. (2023). Sac 2022 dataset. https://figshare.com/articles/dataset/SAC_2022_Dataset/17297477.
- Taha, A. and Hadi, A. S. (2019). Anomaly detection methods for categorical data: A review. *ACM Computing Surveys (CSUR)*, 52(2):1–35.
- WasmEdge (2023). WasmEdgeRuntime. <https://github.com/WasmEdge/WasmEdge>.
- WebAssembly (2023). Wasi tests. <https://github.com/WebAssembly/wasi-testsuite>.
- Wu, S. and Wang, S. (2011). Parameter-free anomaly detection for categorical data. In *International Workshop on Machine Learning and Data Mining in Pattern Recognition*, pages 112–126. Springer.
- Yassin, W., Udzir, N. I., Muda, Z., Sulaiman, M. N., et al. (2013). Anomaly-based intrusion detection through k-means clustering and naives bayes classification. In *Proc. 4th Int. Conf. Comput. Informatics, ICOCI*.