The AV says: Your Hardware Definitions Were Updated!

Marcus Botacin*, Lucas Galante†, Fabricio Ceschin*, Paulo C. Santos‡
Luigi Carro‡, Paulo de Geus†, André Grégio*, Marco A. Z. Alves*
*Federal University of Paraná (UFPR-BR) – {mfbotacin, fjoceschin, gregio, mazalves}@inf.ufpr.br
†University of Campinas (UNICAMP-BR) – {galante, paulo}@lasca.ic.unicamp.br
‡Federal University of Rio Grande do Sul (UFRGS-BR) – {pcssjunior, carro}@inf.ufrgs.br

***Abstract*—Although malware is a threat for most systems, the main line of defense against them (AntiViruses, or AVs) are performance-intensive applications that cause slow down due to the need of constant target-system monitoring. An effective alternative for accelerating AVs operation is to move them from software to hardware, thus eliminating their imposed performance overhead. Hardware-AVs, in turn, present another drawback: the update of malicious definitions is essential for AVs working in constant changing scenarios, but challenging to be deployed in hardware. In this paper, we propose REHAB (REconfigurable, Hardware-Assisted Blocker for malware), a reconfigurable, hardware-based AV that eliminates the performance overhead imposed by standard AVs and streamlines malicious definitions updates. REHAB is based on low-level features (e.g., branch prediction and cache accesses rates) which are classified using machine-learning (ML) algorithms (SVM, Random Forest and MLP) implemented in FPGA to facilitate components integration. We show that REHAB is a practical, effective solution for malware detection that also addresses concept-drift caused by new malware trends, since we are able to adjust the weights of our hardware-modelled neural network through software updates, and we also support fine-grained settings, such as changing the entire ML classifier in runtime (e.g., from Random Forest to MLP).**

I. Introduction

Malware is a major concern for end-users and system administrators, as it may cause either financial and image losses. Besides their significant drawbacks, such as their performance-intensive cost [3], [17], [30], antivirus solutions (AVs) are the main line of defense against malware for most users. To protect their users, AVs need to constantly monitor running processes in the search for any suspicious signs, which causes the machine to leverage significant part of its processing power to run the AV instead of actual users' processing tasks. Thus, enhancing AVs not only helps to create more secure environments, but also more efficient systems.

An enhancement strategy to speed up AVs is to move them from software to hardware [2], [31], which eliminates all overhead of running additional AV code among all other user's applications. This paradigm shift, however, introduces two new challenges: (i) identifying new features for malware classification, as the previously leveraged software features will not be available in hardware (semantic gap); (ii) allowing AVs' malware databases updates, since hardware storage is much more limited and less flexible in comparison to software. Recent research on security has addressed the first challenge by showing that hardware events can be leveraged as features for malware detection [11]. While the second challenge remains an open problem in the security field, recent research on computer architecture enabled circuit logic updates through the use of reconfigurable hardware for many tasks [13], [22]. However, none of them addressed the development of domain-specific security solutions. Therefore, we plan to bridge this gap by proposing a hardware AV using low-level events, implemented in a reconfigurable way that facilitates easy updates by software. On the one hand, we will still benefit from years of AV industry knowledge on deploying updates for the most recent threats. On the other hand, our novel solution does not suffer from the overhead imposed by standard, software-based AVs.

We introduce the REconfigurable, Hardware-Assisted Blocker for malware (REHAB), an AV modeled as a low-level mechanism that captures data from Hardware Performance Counters (HPCs) [20]—the total branch rate, mispredicted branch rate, total cache access rate, cache miss rate, and average total instructions—and deploys a classifier that can be updated by software. In REHAB's model, the AV company establishes the classifier setup, so AV updates can both modify the weights used for classification (allowing for more strict or lax policies) as well as replace the entire classifier by a new implementation, thus addressing malware classification problems, such as concept drift [7], [21].

REHAB was implemented in FPGA and evaluated with 4,077 real Linux malware samples. We show that REHAB can have its hardware reconfigured, which represents the support for different in-hardware classifiers able to be updated, and in therefore be able to identify different classes of malware broadly. REHAB design achieves detection rates of up to 97% while demanding heterogeneous hardware resources, requiring from 1.3K up to 11K Logical Unit Tables (LUTs) and making use of up to 39 DSPs instances (worst-case).

Our contributions are threefold: **1.** we introduce the design and implementation of REHAB, a reconfigurable, hardware AV whose goal is to eliminate the overhead of standard software AVs whereas still allowing updates to be performed from software; **2.** we evaluate REHAB to show its ability to operate in practical scenarios, including the complete replacement of the implemented classifier when ML classifier's concept drift effects are identified. **3.** we present an exploratory design evaluation to show the impact of reconfiguring logic circuits

aimed to implement distinct ML classifiers, thus providing support data for future developments.

This paper is organized as follows: Section II presents background information to support our development; Section III presents the design and implementation of Malware REHAB; Section IV evaluates the Malware REHAB in actual scenarios; Section V discuss the impacts and limitations of our proposed solution; Section VI presents related work to better position REHAB. Finally, we draw our conclusions in Section VII.

II. Background

A. AV Paradigms

Behavior-based Detection is the paradigm most used by the current AVs. The function calls of each runningapplication are intercepted by the AV, and their parameters are scanned by suspicious signs, such as an application exfiltrating user's sensitive data via network or process writing code into third processes (injection). Due to this intercepting characteristic, AV's code is always running among application's code, thus imposing a permanent monitoring overhead.

Profiling-based Detectors establish profiles about which is considered normal during system's operation and detect deviations from these profiles using, for instance, ML classifiers. Hardware AVs often do not have information about software function calls (semantic gap), thus profile are usually established from processor *metadata*, such as branch mispredictions and cache misses rates. This strategy detects malware because whereas benign software (*goodware*) are often well-behaved applications, exploiting temporal and spatial locality, malware often follow unusual paths to exploit vulnerabilities, thus leading to high rates of misses as a side effect.

Profile-based AVs operate in two modes: whole-system and per-process basis. The first considers all applications which executed within an interval in a single manner for its decision, thus imposing low overhead at the cost of a smaller accuracy. The latter considers individual process data, achieving high accuracy, but imposing greater overhead, as a decision algorithm runs for each active process. Figure 1 exemplifies a per-process classifier distinguishing two cases of malware from goodware executionss.

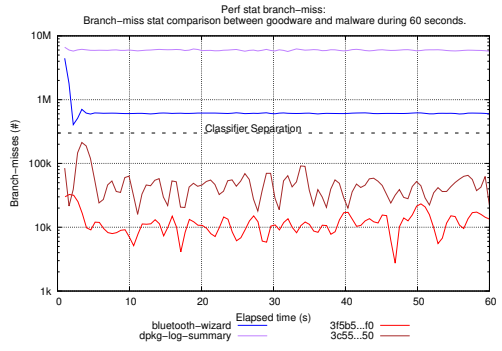

Fig. 1. **Malware Classification using low level features.** The branch misses rate can be used to separate goodware from malware sample's execution.

B. ML Classifiers

Support Vector Machine (SVM) considers a geometrical hyperplane as a decision surface, such that the separation between the samples is maximal [5]. The algorithm uses support vectors (samples from both classes) that are closest to the hyperplane, aiming to maximize the margin from the training data, resulting in two parameters w and b [9]. Thus, one can use both parameters to predict the class of a given sample x_i using the equation $f(x) = \sum_{i=1}^{n} w_i x_i + b$. If $f(x) < 0$, it belongs to class 0, otherwise, class 1. Figure 2 (left) shows SVM's working, with the support vectors creating a *hyperplane* (vertical line) with maximum separation between the two classes. Its prediction circuit implementation (right) is composed by multipliers (that multiply X_i by w_i) and adders (that sum the output of the multipliers with b) responsible for generating the output that defines the class of the sample..

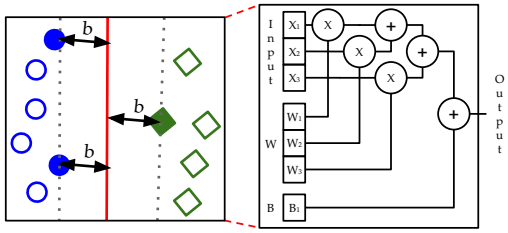

Fig. 2. **SVM**. A hyperplane with maximum separation between two classes is created and used to predict samples (left). The circuit implemented (right) multiplies the input (x_i) by the learned parameters (w_i) and adds b.

Random Forest (RF) consists of a collection (ensemble) of decision trees (classifiers that create a set of if-then-else rules to classify new samples, each of them trained on bagged data using a random selection of features and cast a vote for the most popular class for a given input [6]. For each decision tree, a subset of the train set is used, randomly selecting the features used for that given tree. Then, for each trained tree, a class is predicted, and the majority one is determined as the final decision (voting). A single decision tree is shown in Figure 3 (left), where each node corresponds to a decision for a given feature x_i. The corresponding circuit for this classifier (right) is composed by comparators (for each feature), whose output is the input of a MUX that selects the decision path to follow (true or false).

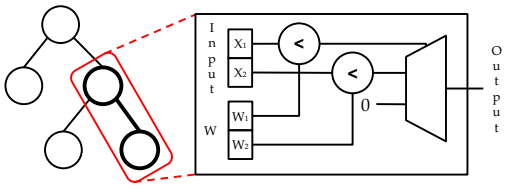

Fig. 3. **Random Forest**. A single decision tree (from the ensemble) is shown (left) with the corresponding circuit of two nodes (right), with comparators and a MUX deciding the decision path.

Multilayer Perceptron (MLP) is a feed-forward neural network that is implemented by multiple neurons, called perceptron (the smallest component of a neural network), which are aligned in n hidden layers capable of extracting useful information of

the input and generating an output (class prediction). A single perceptron is composed by a set of weights w and a bias b, both computed in the training process, and by an activation function φ responsible for determining the shape and intensity of the output value [18]. Every MLP neuron has its own sets of weights and bias, which, given an input x, computes its output using the Equation $f(x) = \varphi(\sum_{i=1}^{n} w_i x_i + b)$ [18]. The input of a hidden layer is the combined output of the previous layer, i.e., a vector containing all the values calculated before (since MLP is fully-connected). Figure 4 (left) shows an example of a MLP network with just one hidden layer and the circuit implementation (right) of a single neuron, which performs the same function of SVM, but relies on an activation function before obtaining an output. The main difference between a single perceptron and SVM is that SVM finds the best *hyperplane* possible (with maximum margin), while perceptron adjusts the *hyperplane* according to its training process: when it misclassify a sample, its weights are updated to correct it.

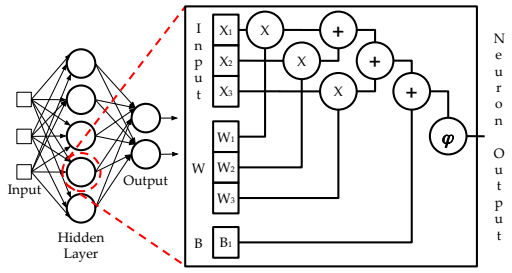

Fig. 4. **MLP**. A feed-forward neural network composed by multiple neurons (left), which circuit implementation (right) is similar to SVM, but using an activation function to calculate the output.

III. Design & Implementation

A. Model & Assumptions

System Model. REHABs aims to eliminate the AV monitoring load imposed to the CPU and GPGPUs and allow them to be fully used for processing user's tasks. We assume that REHAB is supported by an hybrid CPU-FPGA platform, such as Intel's Xeon [29]. We consider that, in the current scenario, unlike the CPU and even GPGPUs, the FPGA co-processor will be only eventually used by most user's applications; hence we can outsource AV's processing loads to it without significantly impacting system's performance. Moreover, recent research work has been suggesting that FPGAs can outperform GPGPUs when operating as co-processors [16].

Threat Model. Malware REHAB is a profiling-based AV solution which considers low-level events as features for profiling-based malware detection, thus not requiring any static signature to operate. Because of this characteristic, REHAB detects anomalies in kernel and userland, however ensuring kernels integrity and preventing privilege escalation is out of REHAB's scope, since it is not an AV responsibility.

AV Operation Model. REHAB is not supposed to replace existing AV solutions, but to enhance them by providing a mechanism for eliminating real-time data collection and analysis overheads. When a given threat is detected at the hardware level, REHAB raises an interrupt to allow the software-based AV to proceed with its inspection and block threat's execution. Therefore, REHAB benefits from years of AV industry's knowledge on detecting malware samples. REHAB's operation model expects the AV company to provide classifier definitions to be applied to the collected data. AV updates can be deployed by software, thus replacing the previously modeled classifier by the new definitions leveraging the reconfigurable hardware capabilities. The circuit reconfiguration might be limited to changing classifier's weights, thus enforcing more strict security policy, or also consider the replacement of the so-far deployed in-hardware classifier by a new one which outperforms it. We expect that classifiers definitions and weights will be defined and delivered by the AV companies according to the threats they are identifying in the wild for the period covered by the updates. REHAB design benefits from AV companies knowledge to reduce its footprint by implementing only classifier's prediction steps, leaving model's training at AV companies' charge.

B. Design

REHAB comprises three components: (*i*) an userland AV; (*i*) a kernel driver; and (*i*) a reconfigurable hardware classifier integrated into the main CPU. The userland component is a standard AV which will receive malware detection notifications from the hardware layer and perform infection blocking. It will also download definition updates from the AV company servers and demand the hardware layer to deploy them. The second component is a kernel driver which enables userland-hardware communication. It works as an interface to the hardware since CPU control registers should not be directly accessible from userland to avoid malware tampering. These components are implemented as standard software pieces and since this work is focused in hardware aspects, we focus our description on describing the innovative REHAB hardware component.

The third component, depicted in Figure 5, is the innovative hardware layer responsible for periodically collecting CPU events data according to an established sampling rate, attributing it to a feature vector and classifying it as malicious or not. The events data can be collected directly from Hardware Performance Counters (HPCs) available in modern CPUs [20], Therefore, supported by the *driver* mentioned above, HPCs can be accessed via shared mapped memory, thus not requiring any CPU modification. Once the feature vectors are constructed, they are classified by the currently deployed ML algorithm, which raises an interrupt when an execution is deemed suspicious. This circuit is under continuous operation, thus monitoring the system execution according to the previously set sampling rate.

When a definition update is received by the userland AV, the data is forwarded via the kernel driver that writes it in REHAB's memory via memory mapped registers and forces it to reload its configurations from the externally-written memory (Figure 6). Upon set with new values, REHAB starts detecting malware using the new definitions.

C. Implementation

To prototype REHAB, we leveraged a set of independent tools so that we could easily adjust each parameter. Our

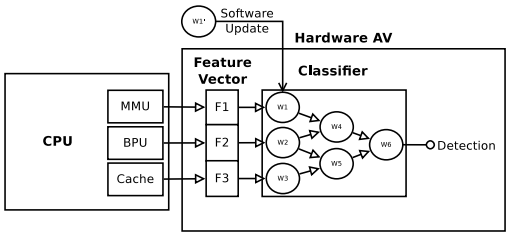

Fig. 5. **REHAB Architecture**. CPU's HPC data is used as feature for a FPGA-based, reconfigurable ML classifier updatable via software.

development consisted of three steps: (*i*) Data collection; (*ii*) Classifier Selection; and (*iii*) Classifier Prototyping.

To prototype the data collection procedure, we leveraged *Linux Perf* to collect HPCs data periodically, considering whole-system (10s) and per-process (1s) AV profiling modes, thus simulating REHAB's continuous monitoring. REHAB considers the following HPCs: (*i*) total branch rate; (*ii*) mispredicted branch rate; (*iii*) total cache accesses rate; (*iv*) cache misses rate; and (*v*) average total executed instructions.

Upon collecting HPCs data, we prototyped multiple REHAB classifiers by leveraging Python `sklearn` library. This step simulates an AV company investigating the best classifier for the malware active in a given period. We selected the three classifiers most used in the malware detection literature: (*i*) Linear Support Vector Machines (SVM); (*ii*) Random Forest (RF); and (*iii*) Multi-Layer Perceptron (MLP) neural network.

Once classifiers and their parameters were defined, we prototyped the classifier's predictors in the Xilinx Artix-7 FPGA to identify how many resources would be required for actual implementation, hence evaluating REHAB's viability.

IV. Evaluation

To evaluate REHAB's detection and update capabilities, we first show why the current behavior-based software AVs are performance-intensive, as well as the benefits of a paradigm shift to profile-based AVs. Further, we show the benefits of a hardware accelerator to mitigate the impact of frequent AV checks of per-process, profile-based AVs. Finally, we show that such solution must be reconfigurable to support the frequent AV updates required for accomplishing high detection rates.

A. Experimental Setup

TestBed. All profiling tests were performed with a fresh installation of Ubuntu 16.04 LTS desktop x64 running on a 16GB, i7-7700 @ 3.60GHz computer.

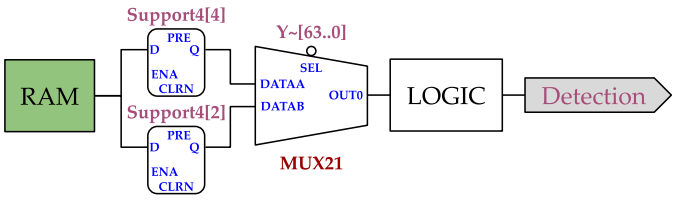

Fig. 6. **Excerpt of a ML classifier implemented in FPGA**. ML parameters are loaded from an external memory at startup and can be updated by software writes to the external RAM memory.

Dataset. For all experiments, we used the binaries of `/bin` and `/usr/bin` directories of a fresh Ubuntu installation as *goodware*-labeled samples, and two datasets of real Linux *malware* samples: (i) the VirusShare (virusshare.com) dataset is composed by 2,642 unique samples collected between 2007 and 2010; and (ii) the VirusTotal (virustotal.com) dataset is composed by 1,435 samples unique samples collected in 2017. These datasets present a family distribution as shown in Table I.

TABLE I
Malware Family Distribution in the evaluated datasets.
Exploits are prevalent.

Family	Exploit	Virus	Backdoor
Prevalence	25%	23%	16%
Family	Rootkit	Worm	Others
Prevalence	11%	15%	10%

B. AV Paradigm Shift

Current AVs operate in a behavioral way by continuously intercepting application's API calls, thus imposing them a significant performance overhead. REHAB operates in a profiling-way, thus only inspecting system state at a given sampling rate. To evaluate the impact of the paradigm shift, we compared the resources usage by a whole-system, sampling-based AV and a behavior-based AV when monitoring the execution of the previously described datasets. We considered ClamSentinel [8] AV for Linux as the behavior-based AV and implemented ML classifiers predictors to operate as a sampling-based AV (compiled using GCC -O3).

During ClamSentinel's monitoring, the system presented an average CPU usage rate of 37.5% (1.4G inst/s), 20% of all these instructions (270M inst/s) were spent running AV's code. For each global 10s sampling interval, a total of 2.7G instructions were spent by ClamSentinel. Each ML classifier consumed, on average, 650M instructions per scan. When considering the same 10s interval, the sampling AV consumed 650M instructions instead of 2.7G instructions from the behavior-based AV, a 4× execution speed up.

C. The Need for Hardware Acceleration

While running a ML classifier costs more than intercepting function calls and checking parameters, there is a performance gain resulting from the fact that function calls are constantly intercepted. However, ML classifiers run only periodically, since the HPCs perform the profiling data collection. This trade-off is affected by the security model defined by the AV, since too long checking intervals might lead to vulnerable system by attacks occurring between two checks. If an AV operates in a model in which checks are performed every 1s or in a per-process-basis (many running processes per second), no performance gain will be observed. Thus, the paradigm shift to low-level features is a **requirement** for overhead elimination, allowing AVs to not check individual API calls but the general hardware state, but it is **not enough** for complete overhead elimination, thus **requiring a hardware accelerator**.

Table II shows the difference in executing the software and the hardware ML prediction algorithm in terms of processing

TABLE II
EXECUTION SPEEDUP PER AV CHECK. HARDWARE ACCELERATOR IS ESSENTIAL FOR OVERHEAD ELIMINATION.

ML algorithm →	**SVM**	**RF**	**MLP**
CPU	220μs	270μs	240μs
FPGA+Comm	124.5ns	111.2ns	158.9ns
Speedup	1.7k×	2.4k×	1.5k×

time. All algorithms in their FPGA implementation run faster than their software counterparts, since the FPGA provides dedicated circuits, thus not being affected by general-purpose hardware issues, such as pipeline stalls and memory latency. Therefore, each AV scan (for each process or sampling interval) performed in the FPGA represents a significant speed up. Moreover, the FPGA execution time is independent of AV sampling rate, thus resulting in overall performance gains for a sampling AV. Finally, the FPGA implementation completely mitigates the overhead of running AV code among other task's code, since all AV tasks are outsourced to the accelerator.

D. The Need for Reconfigurable Hardware

We here show that reconfiguring hardware is essential for AVs keeping up with high detection rates since each ML classifier presents characteristics more appropriated to detect malware observed in distinct periods of times and regions. We considered a per-process profiling AV for all experiments.

AV Detection Baseline We first evaluated whether REHAB's proposed AV logic layer was effectively able to distinguish between malware and *goodware* execution based on the collected performance counters data. In practice, distinct classification algorithms present distinct accuracy rates due to their intrinsic factors, as following demonstrated for multiple classifiers trained using the VirusTotal dataset.

Tables III, IV and V show accuracy results for multiple SVM, MLP and RF parameters when classifying the VirusTotal dataset. In all cases, classifiers were trained using a 50%-50% malware/*goodware* distribution, and accuracy results are reported considering 10-fold cross-validation procedures.

TABLE III
SVM CLASSIFIER. 1000 ITERATIONS IN A LINEAR KERNEL RESULTS IN THE BEST ACCURACY FOR THE VIRUSTOTAL DATASET.

Kernel/Iter (#)	1000	10000	100000
Poly	0.2960	0.2960	0.2960
Linear	**0.8256**	0.7952	0.8088
rbf	0.4793	0.4793	0.4793

TABLE IV
MLP CLASSIFIER. ALPHA AS 100 WITH ADAM SOLVER RESULTS IN THE BEST ACCURACY FOR THE VIRUSTOTAL DATASET.

Solver/Alpha (#)	0.01	1	100	1000
sgd	0.4997	0.4997	0.4997	0.5003
adam	0.7098	0.7218	**0.7433**	0.7213
lbfgs	0.4997	0.4997	0.4997	0.4997

We notice that the Random Forest (64,16) classifier presented the best classification accuracy among all valuated models; thus it should be selected by an AV company to be distributed to their customers via the Internet.

TABLE V
RF CLASSIFIER. 16 ESTIMATORS AND A MAX DEPTH OF 64 RESULTS IN THE BEST ACCURACY FOR THE VIRUSTOTAL DATASET.

Depth/Est (#)	8	16	32	64	128
4	0.9240	0.9172	0.9178	0.9214	0.9199
8	0.9366	0.9366	0.9398	0.9434	0.9403
16	0.9377	0.9455	0.9408	0.9445	0.9429
32	0.9350	0.9439	0.9403	0.9460	0.9445
64	0.9392	**0.9466**	0.9445	0.9434	0.9445

Weighted Classifier Detection. Classifiers present distinct detection rates not only due to their inherent characteristics but also due to parameters configuration. A significant advantage of deploying a reconfigurable classifier is to adjust classifiers weights according to the detection needs identified by the AV company. A typical scenario that requires adjusting classifier's weights is when the characteristics observed in samples collected in-the-wild are significantly distinct from the ones used to train the classifier. To simulate this scenario, we also trained our classifiers using the VirusShare dataset and compared accuracy results to the classifiers trained using the VirusTotal dataset. Table VI, VII and VIII shows accuracy results for multiple SVM, MLP and RF parameters, respectively, when classifying the VirusShare dataset.

TABLE VI
SVM CLASSIFIER. 1000 ITERATIONS IN A LINEAR KERNEL RESULTS IN THE BEST ACCURACY FOR THE VIRUSSHARE DATASET.

Kernel/Iter (#)	1000	10000	100000
Poly	0.3644	0.4234	0.4234
Linear	**0.7705**	0.7353	0.7266
rbf	0.5001	0.4759	0.4759

TABLE VII
MLP CLASSIFIER. ALPHA AS 1 WITH ADAM RESULTS IN THE BEST ACCURACY FOR THE VIRUSSHARE DATASET.

Solver/Alpha (#)	0.01	1	100	1000
sgd	0.4999	0.4999	0.4929	0.4999
adam	0.7288	**0.7614**	0.6951	0.7067
lbfgs	0.4999	0.4999	0.4999	0.4997

TABLE VIII
RF CLASSIFIER. 16 ESTIMATORS AND A MAX DEPTH OF 16 RESULTS IN THE BEST ACCURACY FOR THE VIRUSSHARE DATASET.

Depth/Est (#)	8	16	32	64	128
4	0.9564	0.9569	0.9577	0.9601	0.958
8	0.9644	0.9642	0.9653	0.9644	0.9661
16	0.9626	**0.9671**	0.9655	0.9639	0.9671
32	0.9644	0.9642	0.965	0.9644	0.9661
64	0.962	0.965	0.9442	0.9653	0.9647

We notice that for MLP and RF, the best results for the VirusShare dataset are obtained when using distinct parameters than used for the VirusTotal database, thus showing that providing AVs with the ability to reconfigure their classifiers dynamically is essential to increase security coverage. For the particular case of the RF classifier being deployed by the AV company, the results indicate that to support such

parameter change, the hardware would be required to reduce its implemented classifier's depth from 64 to 16 to achieve the best accuracy rate in the new scenario, which means skipping the last classifier stages. This procedure can be implemented in hardware via either changing multiplexers outputs to consider the signal from an early circuit phase or by reconfiguring the whole hardware to implement an smaller classifier version.

Overcoming Classifier Drift. Classifiers handling very diversified data, such as malware samples, after some operation time may present a natural accuracy reduction due to an effect known as concept drift [7], [21], when the learned model do not correspond anymore to the operational scenario because it changed significantly. In the malware case, it may occur, for instance, due to the high number of malware variants daily created. A significant advantage of having a fully reconfigurable logic layer is that AV solutions can change not only classifier weights but also the entire classifier when it starts drifting. To evaluate this possibility, we leveraged classifiers trained with one dataset to predict the malware samples from another dataset, which allows simulating the concept drift effect. More specifically, we considered the six classifiers presenting the best accuracy results, as previously presented, and leveraged the three classifiers trained with the VirusTotal dataset to predict the VirusShare dataset samples and vice-versa. Our evaluation results are presented in Table IX.

TABLE IX
CLASSIFIER'S CONCEPT DRIFT. WHEREAS THE MLP CLASSIFIER BEST SCORED IN THE VIRUSTOTAL DATASET, THE RANDOMFOREST CLASSIFIER WAS THE BEST CHOICE FOR THE VIRUSSHARE DATASET, THUS SHOWING THE NEED OF HAVING RECONFIGURABLE AV MECHANISMS.

Classifier/Dataset	VirusShare	VirusTotal
Random Forest	**0.9144**	0.6953
MLP	0.881	**0.9738**
SVM	0.9079	0.5728

Although previous experiments demonstrated that Random Forest is the best classifier to train and predict samples having the same characteristics and same datasets (Table V and VIII), it does not hold true when concept drifting is considered. Whereas Random Forest still presents the best performance for classifying the VirusShare (oldest) dataset even when trained with the VirusTotal (newest) dataset, the MLP classifier outperforms RF in the opposite scenario, when classifying the VirusTotal (newest) dataset after being trained with the VirusShare (oldest) dataset. Therefore, an AV company operating in such dynamic scenario would be required to update its deployed classifier to achieve the highest possible malware detection rate, thus reinforcing the need for reconfigurable hardware AV platforms. From a hardware perspective, the classifier change implies in completely reconfiguring the FPGA to reflect a distinct classifier than previously deployed.

E. Implementation Evaluation

After identifying the best parameters for all classifiers, we prototyped their implementation in FPGA to identify how large their circuits would become, since the larger the circuit, the larger the FPGA requirements (e.g., chip area). In our experiments, all classifiers implementation fit in the FPGA, thus being feasible to be implemented in actual systems.

Table X shows REHAB's multiple classifiers implementation data in contrast to other work implementing the same classifiers in FPGA. Whereas results cannot be directly compared since classifiers parameters (e.g., depth, number of trees and perceptrons) depend on the target application (e.g., malware detection, image classification, etc.), we notice that REHAB requires less LUTs than these other implementations, thus ensuring its implementation viability.

There are two main reasons for REHAB's compactness: (*i*) REHAB uses fewer features as input in comparison to the related work shown in Table X, since it relies on already very qualified information as input. REHAB considers an average of HPCs values (temporally rich information) whereas the related work consider raw image files, which requires additional hardware circuits to be described in terms of features. moreover (*ii*) REHAB only implements the prediction step of ML algorithms, thus skipping the logical units responsible for implementing the training step present in similar approaches, which allows REHAB to use $\approx 50\%$ gates than these.

We also notice that each classifier presents its own characteristics regarding implementation, with simpler algorithms requiring fewer logic units to be implemented. For instance, the SVM classifier always requires the same number of LUTs, as it always performs the same dot product computation to the input vector, being the implementation which requires fewer logic units. The MLP also performs the same dot product computation to the input vector, but its requirements change according to the number of perceptrons performing this same operation in parallel. As neural networks are usually composed of a significantly large number of neurons (and sometimes layers, depending on the complexity of the problem), the MLP implementation is often the most costly one. Finally, the Random Forest classifier implementation varies both according to the number of the used tree but also according to the tree characteristics themselves, presenting intermediary requirements in comparison to SVM and MLP.

Classifiers' Implementation Tradeoffs. Given the differences as mentioned earlier, for each classifier implementation and our goal to reconfigure the matching mechanism from one classifier to another, we here investigate how much a circuit change when a reconfiguration is triggered.

As SVM implementation is constant, we first explore the multiple possible implementation decision for the Random Forest Classifier. Table XI shows how the amount of hardware resources changes, in average, when we increased the depth of each single decision tree from the best trained RF model. We notice that the decision tree starts growing slow, as the base case already encompasses multiple logic unities to implement the necessary "computations", such as "reading" the input wires and and "writing" to output wires, which are reused for the other layers. After the 7^{th} layer, the decision tree starts growing significantly, as no reuse is possible and almost the same number of existing decision nodes are added.

TABLE X
IMPLEMENTATION OF CLASSIFIERS (BEST TO WORST CASES). EACH CLASSIFIER PRESENTS DISTINCT CHARACTERISTICS. REHAB DEMANDS SMALLER RESOURCES IN AN OVERALL MANNER.

Classifier	Work	LUTs/REGs/MULs/DSPs	Classifier	Work	LUTs/REGs/MULs/DSPs	Classifier	Work	LUTs/REGs/MULs/DSPs
	This	520/196/5/20		**This**	707/40/0-7.5K/240/0/0		**This**	170/89/5-11K/690/502/38
SVM	[25]	832/–/–/–	RF	[14]	4k-24K/–/–/–	MLP	[14]	6.7K/5K/–/–
	[23]	748/–/–/–		[24]	600-118K/–/–/–		[12]	26.8K/4K/–/–

TABLE XI
DECISION TREE GROWTH. INITIAL GROWTH IS SLOW DUE TO COMPONENT REUSE AND THE "OVERHEAD" CIRCUIT RESPONSIBLE FOR "READING" AND "WRITING" WIRES.

Depth	1	2	4	7	8	16	32	64
LUTs	63	114	370	570	707	1313	1982	2534

The Random Forest algorithm grows not only by increasing the tree's depth but also by adding more trees to the forest. Table XII shows the average result of adding more decision trees to a forest. It is possible to notice that whereas adding more trees to a forest increases the total circuit size, as the adder tree is increased, the total number of LUTs do not double because many gates responsible for signals comparison are reused and only multiplexed to distinct outputs.

TABLE XII
ADDING RANDOM FOREST TREES. ADDING TREES DO NOT DOUBLE THE TREE SIZE BECAUSE COMPONENTS REUSE WHICH ARE MULTIPLEXED TO DISTINCT OUTPUTS.

Trees (#)	1	2	3	4	8	16
LUTs	707	908	1132	1411	1708	7511

We have also evaluated how MLP circuits grow when more perceptron are added, either on different or in the same layers, as shown in Table XIII. We notice that this circuit fastly grows since independent multipliers are allocated due to the inherent parallelism capabilities of the MLP implementation.

TABLE XIII
ADDING MLP LAYERS. THE CIRCUIT SIGNIFICANTLY GROWS DUE TO THE NEED OF ADDING MULTIPLIERS FOR THE DISTINCT PERCEPTRON'S VALUES.

Perceptrons	1	2	4	8	16	64	128
LUTs	170	328	520	1446	2816	5196	11004

V. DISCUSSION

Contributions. REHAB moves AV operation from software to hardware, thus eliminating the overhead imposed by the constant software monitoring procedures. As an advantage over previous hardware AVs, REHAB is implemented in a reconfigurable manner, with its configurations defined via software requests. Thus, REHAB can be updated by software, remains compatible with current AVs and the transition to this new paradigm is transparent to users and vendors.

Limitations. REHAB intends to enhance existing AVs by adding an efficient matching layer. Thus, it does not replace existing AVs, which are still required to block and remedy infections after they are identified by REHAB, and to detect threats outside REHAB's scope, such as browser exploitation or privilege escalation. REHAB also presents a detection delay when the circuit is reconfigured. We consider this delay as acceptable since it only happens few times a day, when the software AV demands a hardware update.

Transition to Practice. REHAB's operation is supported by FPGAs, which are currently available in few systems. Despite that, we believe that REHAB is a promising practical solution, since newer systems are already natively FPGA-powered (see Intel's Xeon [29]), and these may become standard.

REHAB Beyond Traditional Malware. In addition to detecting traditional malware samples (e.g., exploits) via their introduced side-effects (e.g., processor execution metadata), REHAB could also be used to detect attacks against the hardware itself. For instance, by monitoring the performance counter related to the number of cache flush instructions (`clflush`), REHAB would be able to detect rowhammer attacks [4] due to the abnormal high rate of cache flushes imposed by this type of attack. In this sense, REHAB could be reconfigured by AV vendors according the protection offered by them to their clients, adding or suppressing hardware protection capabilities on-demand.

Future Work. REHAB was designed as a Proof-of-Concept to showcase the viability of a reconfigurable hardware AV, and still has development gaps. As future work, we will investigate additional low-level features (e.g., memory accesses patterns) for increased detection capabilities.

VI. RELATED WORK

Hardware-based AVs were proposed by many researchers to detect attacks to enforce some security policy. Arora et al. [2] proposed to detect flow violations using a static call graph model. Zhang et al. [31] also proposed to detect flow violations using eXecuting Only Memory (XOM). These first generation approaches, however, cannot be considered as general AVs, but as attack-specific ones. A second generation of solutions broadened their operation from strict policies to modelled behaviors. Das et al. [10] model software behavior as a Deterministic Finite Automaton (DFA) for malware detection. The FPGA approach of Rahmatian et al. [27] compresses signatures as `n-grams`. Despite more flexible than strict policies, these approaches are still statically modelled, thus not easily updatable. REHAB is also relates to research proposing algorithms for reconfigurable and efficient regular-expression matching [15], [19], [28], an strategies that might also be employed by AVs leveraging static signature matching as detection mechanism. REHAB, in turn, presents a dynamic detection mechanism specifically aimed to be deployed by AVs.

Profiling-based AVs were proposed by many research in the hardware-security field. They usually focus on reading HPC data [11] and classifying them using a ML algorithm. Despite eliminating data collection overhead, these approaches still consume CPU processing time for executing the ML step, still implemented in software. The malware-aware processor [26] enhances this strategy by implementing the classifier in hardware. However, as most previous solutions, its classifier is statically implemented, as their underlying system is not reconfigurable, a drawback overcome by REHAB.

Reconfigurable Hardware is a growing research field, with many proposal over the last years. The proposals range from multiprocessing support [22] to bayesian computing [13]. However, as far as we know, no work proposed leveraging reconfigurable hardware support for moving AV's operation from software to hardware in an updatable manner. We also highlight that whereas many work proposed implementing ML algorithms in hardware, as surveyed in [1], they implement entire classifiers, aiming to also train their models in FPGA and not only matching, as here proposed. Our lightweight design reduces implementation efforts, requires small FPGA area and allows higher clock rates.

VII. Conclusion

We introduced the REHAB for malware, a Reconfigurable, Hardware-Assisted AV aimed at eliminating the overhead of software AVs while streamlining malware definition updates. REHAB leverages low-level events directly captured from CPU HPCs (Hardware Performance Counters), such as cache misses and branch prediction rates, and use them as features for ML classifiers implemented in FPGA, which raises an interrupt to notify the AV when an execution is deemed suspicious. The FPGA implementation allows the software-based AV component to update hardware classifier definitions according to the trends identified by the AV company. REHAB was evaluated with more than 4K real Linux malware collected in the wild, which resulted in detection rates up to 97% and imposed negligible overhead during monitoring (2K times speed up). REHAB classifiers were updated from RF to MLP when concept-drift was detected, showing that REHAB is a practical and effective solution for malware detection in actual, constant changing threat scenarios.

Reproducibility REHAB code is available at: https://github.com/marcusbotacin/Reconfigurable-AV

Acknowledgements. This project was partially financed by the Serrapilheira Institute (grant number Serra-1709-16621), the Brazilian National Counsel of Technological and Scientific Development (CNPq, PhD Scholarship, process 164745/2017-3) and the Coordination for the Improvement of Higher Education Personnel (CAPES, Project FORTE, Forensics Sciences Program 24/2014, process 23038.007604/2014-69).

References

[1] S. M. Afifi, H. GholamHosseini, and R. Sinha, "Hardware implementations of svm on fpga: A state-of-the-art review of current practice," https://tinyurl.com/yyjzl2vd, 2015.

[2] Arora, Ravi, Raghunathan, and Jha, "Secure embedded processing through hardware-assisted run-time monitoring," in *DATE*, 2005.

[3] AV-Test, "Endurance test," https://tinyurl.com/y35egpaw, 2018.

[4] Z. B. Aweke, S. F. Yitbarek, R. Qiao, R. Das, M. Hicks, Y. Oren, and T. Austin, "Anvil: Software-based protection against next-generation rowhammer attacks," in *ASPLOS*. ACM, 2016.

[5] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *COLT*. ACM, 1992.

[6] L. Breiman, "Random forests," *Mach. Learn.*, 2001.

[7] F. Ceschin, F. Pinage, M. Castilho, D. Menotti, L. S. Oliveira, and A. Gregio, "The need for speed: An analysis of brazilian malware classifers," *IEEE S&P*, 2018.

[8] ClamSentinel, "Clamsentinel," https://tinyurl.com/y6mvrl5p, 2018.

[9] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, 1995.

[10] S. Das, H. Xiao, Y. Liu, and W. Zhang, "Online malware defense using attack behavior model," in *ISCAS*, 2016.

[11] J. Demme, M. Maycock, J. Schmitz, A. Tang, A. Waksman, S. Sethumadhavan, and S. Stolfo, "On the feasibility of online malware detection with performance counters," in *ISCA*. ACM, 2013.

[12] S. el Moukhlis, A. Elrharras, and A. Hamdoun, "Fpga implementation of artificial neural networks," 2014.

[13] R. C. G. N. Ewo, A. Pinna, B. Granado, M. Mbouenda, and H. B. Fotsin, "A hardware mpi spawn for distributed multiprocessing reconfigurable system on chip (mp-rsoc)," in *IEEE Int. Symp. on FPGA*, 2014.

[14] A. Ferreira, E. Barros, and T. Ludermir, "Fpga design of a mlp artificial neural network architecture," https://tinyurl.com/y6sfaf7o, 2009.

[15] N. B. Guinde and R. B. Lohani, "Fpga based approach for signature based antivirus applications," in *ICWET*. ACM, 2011.

[16] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "[dl] a survey of fpga-based neural network inference accelerators," *ACM TRETS*, 2019.

[17] T. hardware, "Do antivirus suites impact your pc's performance?" https://tinyurl.com/y2epwo8w, 2011.

[18] S. Haykin, *Neural Networks and Learning Machines*, ser. Neural networks and learning machines. Prentice Hall, 2009.

[19] J. T. L. Ho and G. G. F. Lemieux, "Perg: A scalable fpga-based pattern-matching engine with consolidated bloomier filters," in *Int. Conf. on FPGA*, 2008.

[20] Intel, *Intel® 64 and IA-32 Architectures Software Developer's Manual*, https://tinyurl.com/nxq9dgy, Intel, 2013.

[21] R. Jordaney, K. Sharad, S. K. Dash, Z. Wang, D. Papini, I. Nouretdinov, and L. Cavallaro, "Transcend: Detecting concept drift in malware classification models," in *USENIX*, 2017.

[22] M. Lin, I. Lebedev, and J. Wawrzynek, "High-throughput bayesian computing machine with reconfigurable hardware," in *Int. Symp. on FPGA*. ACM, 2010.

[23] D. Mahmoodi, A. Soleimani, H. Khosravi, and M. Taghizadeh, "Fpga simulation of linear and nonlinear support vector machine."

[24] H. Nakahara, A. Jinguji, S. Sato, and T. Sasao, "A random forest using a multi-valued decision diagram on an fpga," in *ISMVL*, 2017.

[25] D. H. Noronha and M. Fernandes, "Implementação em fpga de máquina de vetores de suporte (svm) para classificação e regressão," https://tinyurl.com/y6lsqkr9, 2016.

[26] M. Ozsoy, C. Donovick, I. Gorelik, N. Abu-Ghazaleh, and D. Ponomarev, "Malware-aware processors: A framework for efficient online malware detection," in *HPCA*, 2015.

[27] M. Rahmatian, H. Kooti, I. G. Harris, and E. Bozorgzadeh, "Hardware-assisted detection of malicious software in embedded systems," *IEEE Embedded Systems Letters*, 2012.

[28] P. Russek and K. Wiatr, "Fpga-accelerated algorithm for the regular expression matching system," *Int. Journal of Electronics*, 2015.

[29] D. Sidler, Z. István, M. Owaida, and G. Alonso, "Accelerating pattern matching queries in hybrid cpu-fpga architectures," in *SIGMOD*. ACM, 2017.

[30] D. Uluski, M. Moffie, and D. Kaeli, "Characterizing antivirus workload execution," *SIGARCH Comput. Archit. News*, 2005.

[31] T. Zhang, X. Zhuang, S. Pande, and W. Lee, "Hardware supported anomaly detection: down to the control flow level," https://tinyurl.com/yxj34won, 2004.