Building Defensive Self-Knowledge Using Embedded Machine Learning in Avionics

Julien DEPAILLAT[1], Philippe BAUMARD[2] and Pierre PARADINAS[3]

[1]*Conservatoire National des Arts et Métiers (CNAM), CEDRIC-ESDR3C & AKHEROS, Ph Student*
[2]*Conservatoire National des Arts et Métiers (CNAM), ESDR3C*
[3]*Conservatoire National des Arts et Métiers (CNAM), CEDRIC*

Abstract
With the increasing demand for smart and connected devices (IoT) and embedded systems have become an integral part of our daily lives. However, this also brings new challenges in terms of security, as these systems often deal with sensitive data and / or perform critical operations. Machine learning techniques have emerged as a promising solution for enhancing the security of embedded systems. By leveraging large amounts of data, machine learning algorithms can identify patterns and anomalies that may indicate a security breach and trigger appropriate responses in real-time. In this article we will provide an overview of the use of machine learning in securing embedded systems highlighting its benefits, potential challenges while discussing some of the recent research in this area.

Keywords
Embedded Systems, Machine Learning, HIDS

1. Introduction

The security of Internet of Things (IoTs) and embedded systems is a major concern in today's world. With the proliferation of connected devices, these systems are prime targets for cyber-criminals looking to exploit their vulnerabilities to gain access to sensitive data or to conduct broader attacks against critical infrastructure.

In recent years, there have been numerous cyber-attacks on industrial and energy systems, further demonstrating the security challenges facing IoTs and embedded systems. Among the most famous are *Stuxnet*, which targeted Iranian industrial systems and slowed down their nuclear program [1], *BlackEnergy* and *Industroyer* targeting the Ukrainian electrical network [2, 3], *TRITON* discovered in a petrochemical complex in Saudi Arabia targeting the *Triconex Schneider Electric* responsible for the security of the installations [4] or *Mirai* infecting thousands of IoTs in order to provoke Distributed Denial of Service (DDoS) attacks [5].

Vulnerabilities can be found at different layers, including the hardware, network and software layers. Hardware layer vulnerabilities are difficult to patch and can be exploited by attackers who have physical access to the device [6].

C&ESAR'23: Computer & Electronics Security Application Rendezvous, Nov. 21-22, 2023, Rennes, France
julien.depaillat@protonmail.com (J. DEPAILLAT); philippe.baumard@lecnam.net (P. BAUMARD); pierre.paradinas@cnam.fr (P. PARADINAS)

Attacks on the network layer include attacks on the communication radio, which can lead to man-in-the-middle attacks or eavesdropping [7]. The software layer contains the operating system, drivers, and applications that use the previous layers to create value, and is also not free of attacks. For example, the infamous Mirai botnet attack targeted devices with known default passwords and resulted in massive DDoS attacks [5].

Various countermeasures have been proposed to provide attack prevention [8], including password protection, data encryption, and restricted access control. However, the effectiveness of these countermeasures has been limited, as many IoT devices still contain vulnerabilities that can potentially be exploited via software updates [6].

Extensive research has been conducted on ways to protect against these attacks. Protection methods can be grouped into two categories: prevention and tolerance methods [6]. Prevention methods aim to eliminate program vulnerabilities and prevent certain types of attacks, while tolerance methods detect attacks and offer system recovery to stop the attack in progress. Previous research has examined various protection methods such as StackGuard [9] and software/hardware-based Control Flow Integrity (CFI) [10], which have limitations that make them controllable [6]. Some text proposes a methodology to implement a performance-based classification system that adopts a remote attestation mechanism to protect IoT devices [11]. Other proposed protection methods include code obfuscation [12], memory encryption [13], monitoring timers [14] or secure boot [15]. The use of Physically Unclonable Functions (PUFs) for authentication is not consistent with the testing approach because it requires hardware modifications [16].

In this article we will first present the context and then focus on Intrusion Detection System (IDS) based on machine learning techniques and their performance on embedded applications. Finally, we will present the experiment we are conducting to develop a Host-based Intrusion Detection System (HIDS) based on behavioral incongruity to detect attacks targeting a Flight Management System (FMS) application.

2. Context

The use of embedded systems in various domains, such as transportation, health, and industry, has been increasing, which leads to the need for secure and resilient systems. In avionics, safety is a top priority. On-board systems must be reliable and robust to ensure proper operation of the aircraft and to guarantee the safety of passengers and crew. Due to their physical isolation, strict protocols and proprietary software, these systems have long been considered unassailable. As mentioned in the introduction, the multiplication of attacks and propagation vectors have called into question this consideration. Indeed, the director of the European Union Aviation Safety Agency (EASA), Patrick Ky announced in 2015 that « Believing that air transport is safe from this kind of threat is tantamount to veiling the face » [17]. Indeed, taking into account the strategic stakes represented by aviation, military and civilian, we understand that this sector can be a prime target for attackers. Their motivations can be many and varied: intention to harm, theft of information, profit, promotion of political objectives ("hacktivist" motivations), etc. Therefore, it is important to guard against external attacks, carried out on “classic” information systems and networks, but also internal attacks which may come from malicious intentional acts, affecting in particular the production, update and deployment chains or unintended uses of embedded systems. One approach to achieving this goal is intrusion detection, which involves detecting and preventing malicious and malevolent behavior. More and more studies are emerging on the possibilities of integrating IDS in order to protect them [18, 19, 20].

2.1. What are Intrusion Detection System

IDS are increasingly popular computer security tools [18, 19, 20] to protect embedded systems against potential threats. An IDS is a software or hardware device that monitors the activities of a computer system or network to detect attempts at intrusion, exploitation or security breach. They mainly consist of a data collection module and a detection module. There are two types based on their placement and the data they collect: Network-based Intrusion Detection System (NIDS) and HIDS.

The NIDS collection modules are placed at strategic locations on the network, for example on a router in a subnet, a Demilitarized Zone (DMZ) or on a firewall ensuring the connection between the internal network and the Internet network. This type of IDS collects data such as source and destination IP addresses, ports used, communication protocols and packet information (content, size, frequency).

The HIDS collection modules are placed directly on the terminals to be monitored, such as servers, workstations or in our case embedded systems. They analyze application data, files, log data, system data, etc.

Each of them is also differentiated according to their method of detection. There are again two categories: signature-based IDS and anomaly-based IDS.

The former uses a detection method based on the comparison of collected data with a pre-established signature base that represents known malicious behavior and is used to detect attacks in real time.

The latter, based on anomalies, are increasingly used in embedded systems. They seek to define a set of models that will characterize the observed system. Once this characterization has been established, anomaly detection is done via a comparison of the current behavior with respect to the latter. If too much deviation is calculated, this means that an intrusion attempt or a security violation is in progress. This method has a major advantage over the signature-based method. Indeed, a new attack or a variant will have an unknown, different signature, and will therefore not be detected by the IDS based on signatures. On the other hand, these attacks will cause a change in the functioning of the system and its application and create a deviation from the normal behavior, therefore generating an alert for those based on anomalies.

Embedded defensive learning therefore consists of training artificial intelligence algorithms to detect and react to anomalies and potential attacks on embedded systems. This type of learning uses machine learning techniques to create patterns of normal system behavior and then uses them to detect potential anomalies.

2.2. Using Machine Learning to Build Defense Knowledge of Embedded Systems

To enhance intrusion detection capabilities, the theory of incongruity can be used in machine-to-machine interactions, including in embedded systems. The theory involves constructing a learning module that calculates self-congruity and self-incongruity values for each machine, node, or component to detect their own behavior inconsistencies, incongruities, or dissonances. This can be used for forensic analysis and to search for dormant threats such as Advanced Persistent Threats (APTs). The patent application proposed by Baumard [21] advocates for an autonomous system for incongruous behavior detection in machine-to-machine interactions without using a previously built normative rule of behavior. However, Bourdon et al. [16] suggests that using statistical analysis or machine learning in combination with other detection methods can enhance detection capabilities, while addressing scalability and standardization issues. Bourdon et al. [16] also discusses the implementation and improvement of a platform for intrusion detection in connected objects based on hardware counters analysis, thus highlighting the need for compatibility with the characteristics of the objects studied. Overall, the theory of incongruity can be valuable for enhancing intrusion detection capabilities in embedded systems, but further research on its effectiveness in comparison to other detection methods is needed [16, 21, 22, 23].

A performance-and machine-learning-based classification system has also been proposed as a runtime attack detection method based on program behavior, hardware performance counters, and machine learning [6]. An experimental approach to detecting behavioral anomalies using hardware performance counters was also explored, and was conducted on platforms with representative IoT devices in a real-world industrial environment [16].

3. Developing an embedded Host-based Intrusion Detection System

Before building an embedded HIDS it is important to highlight the constraints related to this domain in order to be sure not to interfere with its proper functioning. We define here the five main criteria for choosing an anomaly detection technique suitable for avionics:

- Real-Time: the HIDS must not disturb the normal execution cycle of the application. The technique used must be able to process all the data recorded during one execution cycle of the monitored application. The machine learning algorithm must be optimized to ensure that the processing of data and the production of results are carried out in less than 5% of the total cycle time [18].
- Memory footprint: memory resources are generally chosen according to the application and the desired goal. The margin in terms of available memory is therefore often small. It may be necessary to use compression methods in order to store the models, process the data and generate the results. It is also advisable to choose a machine learning algorithm that is not memory intensive.
- Detection capacity: the HIDS must detect anomalies with a low or zero rate of false alerts (false positives) for the highest possible rate of true alerts (true positives). Furthermore,

the results generated must be understandable and easily interpretable in order to be exploitable by an operator in flight.
- Offline: the HIDS must be able to operate without a connection to an external network in order to ensure the best possible protection during operations in areas without internet access. This increases security since no changes can be made remotely.
- Intellectual property: the HIDS must be able to create behavioral models of the system without functional knowledge of the application. No changes to existing code should also be necessary.

An embedded HIDS must therefore respect constraints of time, storage space, limited communication capacity in addition to a minimal modification of the existing one. Indeed, it will in all cases be necessary to add a component or to integrate the HIDS directly into the system environment in order to be able to learn behaviors and detect anomalies.

3.1. The collection module

In order to be able to create its model set and start monitoring the system, the HIDS must be fed with data. As stated previously, it is necessary to respect performance criteria and not modify the existing one. The collection module must therefore be an extremely fast part in terms of execution time. In order not to clutter up the memory space, the collected data must be processed as quickly as possible in order to be destroyed and leave space for the following data. In any case, it will be necessary to allocate a fixed memory space dedicated to this storage. If the allocated space is full and new data arrives, it is necessary to define the behavior to adopt. As a general rule, we preferred to ignore the new data and wait for a new execution cycle of the application to continue the analysis in the most consistent way possible. Embedded applications, especially in the avionics field, are generally deterministic, which means that it always produces the same results for the same input in a known period of time, facilitating this process. To agree to the point of non-modification, it is necessary to collect events not provided by the application and generalizable to all on-board systems. There are several categories:

- Hardware Performance Counters (HPCs)
- OS errors
- System / API calls
- Communications / IO
- Memory

Each of these categories provides different information about system activity. Some may be easier to implement than others, notably *HPCs* and *OS errors* which can be used for other activities such as *Health Monitoring* and do not require so no change to the system. The studies by Boyer [6] and Bourdon [16] are based on the use of HPCs to detect anomalies. They put forward certain precautions: pay attention to changes of context which can disturb the reading of the counters, the difference between processors which do not all have the same counters or even the operating system used, two executions in a different environment will not give rise to the same evolutions. In addition, the number of simultaneous readings of HPCs is limited depending on the processor, so you must first choose the most relevant counters for each application. *OS errors* include ARINC 653 error codes (in avionics), missed deadlines, numeric errors, or illegal requests [24]. The information provided by this data alone is insufficient to effectively detect intrusions.

The *system calls* provide very interesting information to characterize the behavior of an application. In the case of a deterministic one, we expect to find the same sequences of calls at similar frequencies. According to Kadar [25] it is an efficient way to detect intrusions; detection accuracy is generally above 90%, despite a high false positive rate, which can easily reach 15% in recent research. These rates depend of course on the learning algorithm used as well as the settings chosen and can therefore be improved by using the right combinations. Furthermore, it is possible to recover the arguments used or the values of the memory pointers on each call in order to enrich the learning. In a similar principle, observing *API calls* provides valuable information about its execution. These calls have the advantage of offering better portability from one system to another regardless of its architecture, in particular in the avionics which are based on the ARINC 653 [18] standard.

communication data can also be useful for detecting anomalies, such as unauthorized connection attempts or malformed data packets. Embedded systems often communicate via specific protocols, such as the CAN [26] protocol used in the automotive industry, and monitoring these protocols can help detect attacks specific to them. Communication biases represent privileged access for an attacker. By monitoring the *IO* data, we can obtain a characterization of the interactions between the various components of the system, which can help detect compatibility or configuration problems [27]. It can also improve system performance by identifying bottlenecks or inefficiencies in input/output and communication processes. Generally, embedded systems are connected to several sensors that provide various measurements of elements such as temperature, pressure, acceleration, speed, etc. By correlating this type of data to system calls, the coverage and robustness of learning are considerably increased.

Finally, a last category can be considered: *memory*. It contains all the data being processed in the system, including program instructions being executed and temporary data. By monitoring memory, it can detect anomalies such as incorrect memory accesses, buffer overflows, attempts to inject malicious code, or memory leaks [28]. It can detect suspicious or malicious behavior, such as attempts to install malicious software, to take control of the system or to alter the data handled by the application degrading its integrity [29]. However, it is more complex to implement and requires very special attention so as not to disturb the proper functioning of the system.

3.2. The learning / detection module

This module represents the heart of the HIDS. It will be in charge of learning the models characterizing the application, which will then allow anomalies to be detected. As with the collection module, it should be as optimized as possible in order to respect the technical constraints associated with the system that hosts it. The choice of technique and learning algorithm is very important. There are 3 techniques: supervised, unsupervised and semi-supervised learning.

Supervised learning [30] is a machine learning technique where a statistical model is trained from a set of labeled data. In this type of learning, the model must predict an output based on known inputs and outputs. These known outputs, or labels, are provided in the training dataset. The learning process consists of adjusting the parameters of the model to minimize the deviation between the predicted outputs and the known labels. Once the model has been trained on a dataset, it can be used to make predictions on new data. Supervised learning is commonly used in applications such as image classification, email spam detection, speech recognition, or real estate value prediction. Commonly used algorithms for supervised learning include decision trees, neural networks, Support Vector Machine (SVM), and random forests.

Unsupervised learning [31] focuses on creating patterns and structures from unlabeled or previously annotated data. It can be used for data classification, image segmentation, anomaly detection, dimensionality reduction, product recommendation, clustering analysis and other data analysis tasks. Commonly used algorithms for unsupervised learning include k-means, self-encoding neural networks, convolutional neural networks, Boltzmann machines [32], Hopfield neural networks [33], and Principal component analysis (PCA).

Finally, semi-supervised learning [34] is a combination of the elements of supervised and unsupervised learning. In this type of learning, the model is trained on a dataset containing both labeled and unlabeled data. In the embedded context, it is assumed that the execution environment is healthy, which constitutes the labeled data to guide learning. It is also possible to integrate known attack datasets into the learning to improve its robustness. However, it is impossible to ensure that all cases of attacks will be tested. Semi-supervised learning is particularly useful when labeled data is expensive or difficult to obtain. This makes it a good choice for the embedded domain since learning does not necessarily need an attack to establish normal behavior. Commonly used algorithms include labeling propagation methods, mixture models, neural networks, and semi-supervised SVMs. These algorithms can be adapted to work with different types of data, such as text, image, and sequence data.

Once the technique has been selected, it is necessary to choose the machine learning algorithm adapted to the on-board criteria. Sayadi [35], whose study is mainly based on the use of HPCs, offers a performance comparison of different machine learning algorithms on the detection of Rootkit, Backdoor and Trojan (table 1) as well as their hardware cost (table 2).

The presented algorithms belong to 5 different families:

- Probabilistic graphs : BayesNet
- Neural networks : MLP
- Rule systems : OneR, JRip
- Decision Trees : J48, REPTree
- Machine Support Vectors : SMO

Bayesian Networks (BayesNets) belong to the class of probabilistic graphical models, which are a category of machine learning that use graphs to represent probabilistic relationships between variables. In a BayesNet, the nodes represent the variables (here the HPCs) and the arcs the probabilistic dependencies between the variables. They are based on Bayes' theorem [36] and use conditional probability methods to evaluate probabilities and predictions.

Multi-Layer Perceptron (MLP) are a type of forward propagation (or feedforward) neural network composed of multiple layers of neurons, where each neuron in one layer is connected

Table 1
ML algorithm comparison of malware accuracy detection

Algorithm	Rootkit	Backdoor	Trojan	Average
BayesNet	88.1%	91.6%	99.0%	92.9%
MLP	94.0%	92.4%	89.8%	92.1%
OneR	81.5%	92.0%	99.0%	90.9%
JRip	84.8%	92.0%	66.3%	81.0%
J48	85.4%	92.0%	65.7%	81.0%
REPTree	82.8%	92.0%	66.3%	80.4%
SMO	91.4%	89.5%	98.8%	93.2%

to all neurons in the next layer [37]. MLP are particularly suitable for nonlinear classification and prediction tasks.

OneR [38] is a type of simple and interpretable algorithm that uses a single rule to predict the class of a new example. It works by choosing the input variable that provides the best prediction for each class and then uses that variable to create a decision rule. This rule is then used to predict the class of new examples. They are widely used for their simplicity and interpretability but can be less accurate than other more complex algorithms.

In the JRip algorithm [39] the training data is divided into an increasing set and a pruning set. First, an initial set of rules is trained on the growing set, using a heuristic method. This over-sized rule-set is then repeatedly simplified by applying a set of pruning operators. At each simplification step, the pruning operator chosen is the one that produces the greatest error reduction on the pruning set. The simplification step ends when applying a pruning operator would increase the error on the pruning set.

J48 is a decision tree-based algorithm [39] that uses a divide and conquer approach to generate a decision tree from input data. The decision tree is a tree structure in which each internal node represents an input variable, each branch represents a value of this variable and each leaf represents an output class.

Like other decision tree-based algorithms, REPTrees generate a decision tree from the input data, using a divide and conquer approach [40]. The particularity of REPTrees is that they use a regression technique to decide on the optimal division of the nodes of the tree. REPTrees are often used for their simplicity and speed, but can suffer from over-fitting and have lower performance than other more complex algorithms for high-dimensional classification problems.

SVM are algorithms that learn to classify data by finding the decision boundary that maximizes the margin between the two classes. The Sequential Minimal Optimization (SMO) is a specific optimization method to solve the margin optimization problem of SVM in an efficient way [41]. SVM and SMO are appreciated for their ability to generalize previously unseen data and for their robustness against noise.

Based on table 1, we can see that the MLP performs better for the detection of Rootkits and Backdoors, whileBayesNet, OneR and SMO dominate for Trojans. The JRip, J48 and REPTree algorithms are far behind in this category. On average, we find that algorithms based on long and complex learning give the most relevant results. However, according to the table 2 they are also the most demanding in terms of execution time and memory used. Although very accurate,

Table 2
ML algorithm comparison of hardware overhead

Algorithm	Latency	Memory (block)
BayesNet	60ns	7645
MLP	1020ns	25667
OneR	10ns	292
JRip	20ns	156
J48	30ns	584
REPTree	30ns	377
SMO	220ns	2246

MLP has the highest latency and highest memory usage, so it doesn't seem to be suitable for critical embedded system. SMO has the best detection results. However, it also has the second most significant impact in terms of execution time. BayesNet seems like a good compromise. They have a very good detection score, an acceptable execution time and a moderate memory impact.

It is important to note that no particular optimization has been made to the implementation of these algorithms. It is therefore possible to make improvements on the execution time and the memory cost. Moreover, depending on the characteristics of the platform and its end goal, it is also possible to choose to optimize one to the detriment of the other.

Once the algorithm is defined, all that remains is to adapt it to the platform. Learning is the most costly part in terms of resources. If this is too important, it is possible to collect the selected data and perform the learning on a conventional computer. It will then suffice to transfer the models to the platform so that the detection module can use them as a normal basis.

4. The experimentation

4.1. Context

To develop our HIDS, we have set up a collaboration with a leading European company that develops Real-Time Operating System (RTOS) used in the avionics field. In order to get as close as possible to reality, we will monitor a FMS application, provided by another leading European company that provides systems and applications in this field. A FMS is an on-board computer used for automatic navigation and guidance, presentation of information, management of aircraft systems, efficient management of fuel and reduction of operating costs [42]. This FMS already works on our partner's RTOS and will therefore be embedded on a T2080 board. It is largely used in the avionics world for this type of application. It has a 4-core processor clocked at 1.8GHz sharing a 2MB L2 cache memory. The Operating System (OS) allows us to define the priority of each application, their order and duration of execution as well as the core(s) used. The FMS application requires 2 cores, so we will use 1 core to perform data collection, machine learning and detection and 1 core to execute our attacks. To validate our experiment we will use 3 criteria:

- The operations of the application must not be disturbed by system modifications, data collection and the HIDS.
- The monitored application must not be modified in any way to adapt to changes made to the OS or to the HIDS.
- A maximum attack detection rate for a false positive rate of 0%.

4.2. Objectives and implementation

In order to carry out this experiment, it is necessary to define the objectives to be achieved as well as the various actions to be undertaken. The main objective of this experiment is to determine if a HIDS based on a machine learning algorithm is viable for detecting intrusions on an embedded system. The second objective is to know the cost of it in terms of performance in order to establish the minimum resources required for its proper functioning. The third will be to determine which actions, human and automatic, could be taken in the event of detection of an attack. Finally, we will seek to assess the degree of portability of this algorithm to other embedded systems / applications. To set up this experiment it will be necessary to carry out the following tasks:

- Modify the OS in order to be able to collect the data necessary for model learning and attack detection (syscall, HPCs, timestamp, context, etc.).
- Choose the machine learning algorithm to use.
- Develop the HIDS using this algorithm so that it is compatible with the embedded OS.
- Optimize the HIDS so that it is embeddable, the least greedy and the fastest possible.
- Create normal usage scenarios of the application covering its functionalities as much as possible.
- Run these scenarios in order to create the models characterizing the proper functioning of the application.
- Create attack scenarios to disrupt the proper functioning of the application and system.
- Run these scenarios to assess the detection capabilities of the HIDS.

4.3. The attacks to detect

In agreement with the teams of our collaborators, we have drawn up a list of attacks that will be put in place to assess the detection capabilities of the HIDS:

- Pre-loaded attacks: code added in the binary application before uploading to the platform. Corruption of the on-board application set (modification of certain functionalities: trajectory calculation, GPS position, etc.).
- Spoofing attacks on sensors [43] feeding the FMS: here, we will seek to know if it is possible to determine that an attack is in progress on one or more external systems on which it the monitored one depends. Indeed, if inconsistent data between them is provided to the FMS, this can lead to unusual actions of the application symbolized by aberrant values in terms of syscall and values of the monitored HPCs.
- Injection attacks: random code, control-flow hijacking [44].

- Passive attacks: side-channel attack on the micro-architecture. Variant of Spectre [45] (memory leak of the application, particularly "cache timing").
- Active attacks: fault injection attacks on the micro-architecture. Rowhammer [46] and variant like Blacksmith [47]. Based on errors in the Dynamic Random Access Memory (DRAM), memory cells can change values influenced by the activity of neighboring cells, an attacker can illegitimately modify a memory space without having access to it.

The following attacks will not be taken into account:

- Passive physical attacks, without interaction with the system: laser attack [48], hidden channel attacks [49] on electromagnetic emissions, etc.
- Attack compromising the security of the OS: if the OS is compromised it is a safe bet that the HIDS which depends on it will be compromised too.
- Attacks compromising the system boot-chain: if the boot sequence is compromised, this means that the OS is compromised and so is the HIDS.

For all attacks we will consider that the opponent has succeeded, by some means, in making the desired changes to the platform / application without being detected. We will not seek to determine which intrusion vector was used using the HIDS even if, according to the modifications made, the details provided during the analysis and detection would allow it.

4.4. Current progress

Currently, the modifications of the OS in order to gather the syscalls, HPCs (6 different), timestamp have been made. Furthermore, we have already been able to assess the impact of this collection in terms of execution time on the application. These results are shown in figure 1. On this graph, we distinguish the time required on the ordinate to collect the number of elements indicated on the abscissa. There is an additional cost in execution time varying from 82ns to 113ns knowing that 99.8% remains below 100ns and 0.2% above.

The HIDS is based on the incongruity work carried out by Baumard [21] and mentioned in part 2.2. This HIDS uses a learning algorithm based on dynamic Bayesian networks presented by Hourbracq [50]. These are classic Bayesian networks, as presented in part 3.2, but whose nodes $\{X_i(t), i = 1 \dots n\}$, representing discrete random variables, are indexed by the discrete time t. Therefore they are Bayesian networks evolving during the execution of the application and not frozen once established. This has several advantages over the BayesNet, namely:

- A lower number of models to explain the different aspects of the application.
- The generated models are more robust.
- Infrequent behaviors with values that can be considered outliers stand out much more.

The HIDS is written in C respecting the POSIX standard allowing it a fairly easy portability. However for the needs of this experiment and in order to be compliant with the specifics of the OS and the avionics certifications, the code had to be adjusted. In addition to this adjustment, large optimizations have been made to address the resource restrictions of embedded systems. It occupies 150kB and uses 22MB of RAM. These 22MB constitute the maximum size that will be necessary for the detection phase. Note that it may vary if we decide to expand the number of

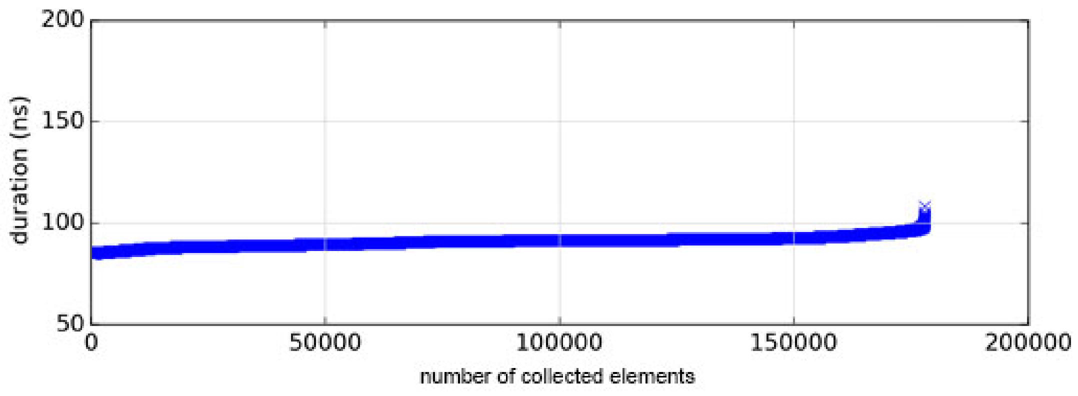

Figure 1: Time overhead to collect desired data for a 30-minute flight.

training samples. Here it is the required memory to load the behavioral models learned on the 20 flights resulting from the different learning scenarios that we have established. The flights have a duration ranging from 30 minutes to 1 hour and 30 minutes. We also plan to test on long-haul flights. The needed learning time is less than the duration of each flight, even if the lower the number of models, the longer the learning will be. Indeed, reinforcing an existing model takes longer than creating a new one.

4.5. Next steps

The next steps will be to compare the models obtained with unaltered flights not used during the learning phase in order to ensure that no alert is generated and that the models can explain the flights.

Subsequently, it will be necessary to set up the scenarios for the attacks set out in 4.3 in order to evaluate the detection capacity of the HIDS. in order to accurately determine when the attack occurs it will be necessary to add information not used by the HIDS to calculate the false positive rate and make the necessary adjustments so that it is 0%.

Finally, depending on the obtained results, we can assess the possible actions to be taken in the event of an attack. However, it should be borne in mind that in this area automatic actions will be limited and should not conflict with flight safety rules.

4.6. Current conclusions of the experimentation

With the progress currently made, we have succeeded in developing a HIDS allowing the creation of models characterizing an embedded avionics application. The additional cost necessary for the operation of the HIDS was deemed minimal by our supplier, recognized in this field, and no modification to the application was made, thus validating the first 2 criteria. Although it is a specific platform, the results obtained are encouraging as to the possibility of porting it to other embedded systems.

All that remains now is the step of constructing the attacks to evaluate the detection capacity of the HIDS and to conclude our study. If it is validated, we will be able to consider deploying the HIDS on a larger number of similar devices since the generated models can be transposed from one system to another as long as they share the same technical characteristics.

5. Conclusion

In summary, the literature highlights the security challenges of IoTs and embedded systems and the need for effective ways to protect against attacks. Various protection methods exist and implementation steps of a HIDS using machine learning has been proposed. The use of this technology can greatly help to detect various and unknown attack without the need of external database nor human intervention. It can greatly improve embedded systems cyber-security. However, more research is needed to ensure that protection for IoTs and embedded systems is effective, efficient, and adaptable to changing security threats.

References

[1] R. Langner, Stuxnet: Dissecting a cyberwarfare weapon, IEEE Security & Privacy 9 (2011) 49–51.

[2] A. Cherepanov, BlackEnergy by the SSHBearDoor: attacks against Ukrainian news media and electric industry, 2016. URL: https://www.welivesecurity.com/2016/01/03/blackenergy-sshbeardoor-details-2015-attacks-ukrainian-news-media-electric-industry/.

[3] A. Cherepanov, R. Lipovsky, Industroyer: Biggest threat to industrial control systems since Stuxnet, ESET 12 (2017).

[4] A. D. Pinto, Y. Dragoni, A. Carcano, TRITON: The first ICS cyber attack on safety instrument systems, Black Hat USA (2018) 1–26.

[5] M. Antonakakis, Understanding the Mirai Botnet, USENIX security symposium (2017) 1093–1110. URL: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis.

[6] Y. Boyer, Étude et conception de méthodes de protection face aux attaques par corruption de mémoire pour systèmes embarqués dans le contexte de l'Internet des Objets, Université Montpellier (2020).

[7] B. Bhushan, G. Sahoo, A. K. Rai, Man-in-the-middle attack in wireless and compu- ter networking — a review, 3rd International Conference on Advances in Computing, Communication Automation (ICACCA) 5 (2017) 1–6.

[8] A. Makkar, N. Kumar, A. Ghoneim, S. Hossain, S. Garg, M. Alrashoud, An Efficient Spam Detection Technique for IoT Devices using Machine Learning, IEEE Transactions on Industrial Informatics 17 (2020).

[9] C. Cowan, C. Pu, D. Maier, H. Hintony, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, Q. Zhang, Stackguard: Automatic adaptive detection and prevention of buffer-overflow attacks, USENIX Security Symposium (1998).

[10] M. Abadi, M. Budiu, Úlfar Erlingsson, J. Ligatti, Control-flow integrity, ACM Conference on Computer and Communications Security (2005).

[11] R. V. Steiner, E. Lupu, Attestation in wireless sensor networks : A survey, ACM Computing Surveys (CSUR) 49 (2016) 1–31.

[12] A. J. Suresh, S. Sankaran, Power Profiling and Analysis of Code Obfuscation for Embedded Devices, in: 2020 IEEE 17th India Council International Conference (INDICON), 2020, pp. 1–6. doi:10.1109/INDICON49873.2020.9342447.

[13] M. Henson, S. Taylor, Memory Encryption: A Survey of Existing Techniques, ACM Comput. Surv. 46 (2014). URL: https://doi.org/10.1145/2566673. doi:10.1145/2566673.

[14] S. Lu, M. Seo, R. Lysecky, Timing-based anomaly detection in embedded systems, in: The 20th Asia and South Pacific Design Automation Conference, 2015, pp. 809–814. doi:10.1109/ASPDAC.2015.7059110.

[15] R. Rashmi, A. Karthikeyan, Secure boot of Embedded Applications - A Review, in: 2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA), 2018, pp. 291–298. doi:10.1109/ICECA.2018.8474730.

[16] M. Bourdon, Détection d'intrusion basée sur l'analyse de compteurs matériels pour des objets connectés, INSA de Toulouse (2021).

[17] B. Trévidic, L'Agence européenne de sécurité aérienne alerte contre le risque de cyber-attaque, Les Echos (2015). URL: https://www.lesechos.fr/2015/10/lagence-europeenne-de-securite-aerienne-alerte-contre-le-risque-de-cyber-attaque-277334.

[18] A. Damien, Sécurité par analyse comportementale de fonctions embarquées sur plateformes avioniques modulaires intégrées, Theses, INSA de Toulouse, 2020. URL: https://hal.laas.fr/tel-02953842.

[19] M.-K. Yoon, S. Mohan, J. Choi, J.-E. Kim, L. Sha, SecureCore: A multicore-based intrusion detection architecture for real-time embedded systems, in: 2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013, pp. 21–32. doi:10.1109/RTAS.2013.6531076.

[20] I. Studnia, Y. Laarouchi, M. Kaaniche, V. Nicomette, E. Alata, A language-based intrusion detection approach for automotive embedded networks, International Journal of Embedded Systems 10 (2018) 1. doi:10.1504/IJES.2018.10010488.

[21] P. Baumard, Autonomous detection of incongruous behaviors, European Patent Application 2 922 268 A1 (2015).

[22] S. E. Smaha, Haystack : An intrusion detection system, Fourth Aerospace Computer Security Applications Conference 44 (1988).

[23] Z. Chiba, N. Abghour, K. Moussaid, M. Rida, Intelligent approach to build a Deep Neural Network based IDS for cloud environment using combination of machine learning algorithms, Computers & Security 86 (2021) 291–317.

[24] P. Parkinson, L. Kinnan, Safety-critical software development for integrated modular avionics, Embedded System Engineering 11 (2003) 40–41.

[25] M. Kadar, S. Tverdyshev, G. Fohler, System Calls Instrumentation for Intrusion Detection in Embedded Mixed-Criticality Systems, in: M. Asplund, M. Paulitsch (Eds.), 4th International Workshop on Security and Dependability of Critical Embedded Real-Time Systems (CERTS 2019), volume 73 of *OpenAccess Series in Informatics (OASIcs)*, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2019, pp. 2:1–2:13. URL: http://drops.dagstuhl.de/opus/volltexte/2019/10893. doi:10.4230/OASIcs.CERTS.2019.2.

[26] K. Ismail, A. Muharam, M. Pratama, Design of CAN Bus for Research Applications Purpose

Hybrid Electric Vehicle Using ARM Microcontroller, Energy Procedia 68 (2015) 288–296. URL: https://www.sciencedirect.com/science/article/pii/S1876610215005640. doi:https://doi.org/10.1016/j.egypro.2015.03.258, 2nd International Conference on Sustainable Energy Engineering and Application (ICSEEA) 2014 Sustainable Energy for Green Mobility.

[27] S. Jena, A. Gupta, Embedded Sensors for Health Monitoring of an Aircraft, Springer Singapore, Singapore, 2019, pp. 77–91. URL: https://doi.org/10.1007/978-981-13-3290-6_5. doi:10.1007/978-981-13-3290-6_5.

[28] P. Weisberg, Y. Wiseman, Efficient memory control for avionics and embedded systems, International Journal of Embedded Systems 5 (2013) 225–238.

[29] M.-K. Yoon, S. Mohan, J. Choi, L. Sha, Memory Heat Map: Anomaly Detection in Real-Time Embedded Systems Using Memory Behavior, in: Proceedings of the 52nd Annual Design Automation Conference, DAC '15, Association for Computing Machinery, New York, NY, USA, 2015, pp. 1–6. URL: https://doi.org/10.1145/2744769.2744869. doi:10.1145/2744769.2744869.

[30] T. Jiang, J. L. Gradus, A. J. Rosellini, Supervised Machine Learning: A Brief Primer, Behavior Therapy 51 (2020) 675–687. URL: https://www.sciencedirect.com/science/article/pii/S0005789420300678. doi:https://doi.org/10.1016/j.beth.2020.05.002.

[31] R. Gentleman, V. J. Carey, Unsupervised Machine Learning, Springer New York, New York, NY, 2008, pp. 137–157. URL: https://doi.org/10.1007/978-0-387-77240-0_10. doi:10.1007/978-0-387-77240-0_10.

[32] N. Zhang, S. Ding, J. Zhang, Y. Xue, An overview on Restricted Boltzmann Machines, Neurocomputing 275 (2018) 1186–1199. URL: https://www.sciencedirect.com/science/article/pii/S0925231217315849. doi:https://doi.org/10.1016/j.neucom.2017.09.065.

[33] G. Joya, M. Atencia, F. Sandoval, Hopfield neural networks for optimization: study of the different dynamics, Neurocomputing 43 (2002) 219–237. URL: https://www.sciencedirect.com/science/article/pii/S092523120100337X. doi:https://doi.org/10.1016/S0925-2312(01)00337-X, selected engineering applications of neural networks.

[34] Z.-H. Zhou, Semi-Supervised Learning, Springer Singapore, Singapore, 2021, pp. 315–341. URL: https://doi.org/10.1007/978-981-15-1967-3_13. doi:10.1007/978-981-15-1967-3_13.

[35] H. Sayadi, H. M. Makrani, O. Randive, S. Manoj, S. Rafatirad, H. Homayoun, Customized Machine Learning-Based Hardware-Assisted Malware Detection in Embedded Devices, in: 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), 2018, pp. 1685–1688. doi:10.1109/TrustCom/BigDataSE.2018.00251.

[36] J. Joyce, Bayes' Theorem, in: E. N. Zalta (Ed.), The Stanford Encyclopedia of Philosophy, Fall 2021 ed., Metaphysics Research Lab, Stanford University, 2021, pp. 1–5.

[37] M. Gardner, S. Dorling, Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences, Atmospheric Environment 32 (1998) 2627–2636. URL: https://www.sciencedirect.com/science/article/pii/S1352231097004470. doi:https://doi.org/10.1016/S1352-2310(97)00447-0.

[38] Z. Muda, W. Yassin, M. N. Sulaiman, N. I. Udzir, Intrusion detection based on k-means clustering and OneR classification, in: 2011 7th International Conference on Information Assurance and Security (IAS), 2011, pp. 192–197. doi:10.1109/ISIAS.2011.6122818.

[39] A. Rajput, R. P. Aharwal, M. Dubey, S. Saxena, M. Raghuvanshi, J48 and JRIP rules for e-governance data, International Journal of Computer Science and Security (IJCSS) 5 (2011) 201.

[40] S. Kalmegh, Analysis of weka data mining algorithm reptree, simple cart and randomtree for classification of indian news, International Journal of Innovative Science, Engineering & Technology 2 (2015) 438–446.

[41] S. Shevade, S. Keerthi, C. Bhattacharyya, K. Murthy, Improvements to the SMO algorithm for SVM regression, IEEE Transactions on Neural Networks 11 (2000) 1188–1193. doi:10.1109/72.870050.

[42] R. P. G. Collinson, Autopilots and flight management systems, Springer Netherlands, Dordrecht, 1996, pp. 366–405. URL: https://doi.org/10.1007/978-94-011-0007-6_8. doi:10.1007/978-94-011-0007-6_8.

[43] D. Davidson, H. Wu, R. Jellinek, V. Singh, T. Ristenpart, Controlling UAVs with Sensor Input Spoofing Attacks, in: 10th USENIX Workshop on Offensive Technologies (WOOT 16), USENIX Association, Austin, TX, 2016, pp. 1–10. URL: https://www.usenix.org/conference/woot16/workshop-program/presentation/davidson.

[44] N. Carlini, A. Barresi, M. Payer, D. Wagner, T. R. Gross, Control-Flow Bending: On the Effectiveness of Control-Flow Integrity, in: 24th USENIX Security Symposium (USENIX Security 15), USENIX Association, Washington, D.C., 2015, pp. 161–176. URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/carlini.

[45] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, Y. Yarom, Spectre Attacks: Exploiting Speculative Execution, in: 2019 IEEE Symposium on Security and Privacy (SP), 2019, pp. 1–19. doi:10.1109/SP.2019.00002.

[46] O. Mutlu, J. S. Kim, RowHammer: A Retrospective, 2019. arXiv:1904.09724.

[47] P. Jattke, V. Van Der Veen, P. Frigo, S. Gunter, K. Razavi, BLACKSMITH: Scalable Rowhammering in the Frequency Domain, in: 2022 IEEE Symposium on Security and Privacy (SP), 2022, pp. 716–734. doi:10.1109/SP46214.2022.9833772.

[48] J.-M. Dutertre, S. De Castro, A. Sarafianos, N. Boher, B. Rouzeyre, M. Lisart, J. Damiens, P. Candelier, M.-L. Flottes, G. Di Natale, Laser attacks on integrated circuits: From CMOS to FD-SOI, in: 2014 9th IEEE International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS), 2014, pp. 1–6. doi:10.1109/DTIS.2014.6850664.

[49] T. Neubert, C. Vielhauer, Kill Chain Attack Modelling for Hidden Channel Attack Scenarios in Industrial Control Systems, IFAC-PapersOnLine 53 (2020) 11074–11080. URL: https://www.sciencedirect.com/science/article/pii/S2405896320305231. doi:https://doi.org/10.1016/j.ifacol.2020.12.246, 21st IFAC World Congress.

[50] M. Hourbracq, P.-H. Wuillemin, C. Gonzales, P. Baumard, Apprentissage et sélection de réseaux bayésiens dynamiques pour les processus online non stationnaires, Revue des Sciences et Technologies de l'Information - Série RIA : Revue d'Intelligence Artificielle 32 (2018) pp. 75–109. URL: https://cnam.hal.science/hal-03228681.