Hardware-assisted Detection of Malware in Automotive-Based Systems

Yugpratap Singh, *Student Member, IEEE*, Abraham Peedikayil Kuruvila, *Student Member, IEEE*
Kanad Basu, *Senior Member, IEEE*
The University of Texas at Dallas
{yps170000, apk190000, kxb190012}@utdallas.edu

***Abstract*—In the age of Internet-of-Things (IoT), automobiles have become heavily integrated and reliant on computerized components for system functionality. Modern vehicles have many Electronic Control Units (ECUs) that control ignition timing, suspension control, and transmission shifting. The Engine Control Module (ECM) is generally recognized as one of the most essential components owing to its functionality of regulating air and fuel input to the engine. Consequently, automotive security is an emerging problem that will only escalate as vehicles integrate more computerized components in conjunction with wireless system connectivity. Attackers that successfully gain access to important vehicular components and compromise existing functionality can induce a plethora of malevolent activities. With the evolution and exponential proliferation of Malware, identifying malicious entities is critical for maintaining proper system performance. Traditional anti-virus software is inadequate against complex Malware, which has engendered a push towards Hardware-assisted Malware Detection (HMDs) using Hardware Performance Counters (HPCs). HPCs are special purpose registers that track low-level micro-architectural events. In this paper, we propose using Machine Learning models trained on HPC data to identify malicious entities in the ECM. Our experimental results determine that the proposed ML-based models can successfully identify malicious actions in an automotive system with a classification accuracy of up to 96.7%.**

***Index Terms*—Automotive Security, Hardware Performance Counters, Machine Learning, Cyber-physical systems**

I. Introduction

The widespread proliferation of technology across a plethora of fields has prompted an advancement in the computerization of automobiles. Modern cars incorporate many human-convenient features such as Global Positioning Systems (GPS), built-in Wi-Fi, and backup cameras. However, integration of contemporary technology extends beyond the passenger-level. Since its inception, vehicles have been produced with significant improvements to the underlying mechanisms that operate the fundamental control modules. Modern automobiles contain many Electronic Control Units (ECUs) that control principal aspects of the car from the aforementioned user conveniences to specific engine parameters such as valve timing and gear control. The Engine Control Module (ECM) is imperative for vehicular performance and is widely acknowledged as one of the most critical components of an automobile. The ECM ensures that the engine is operating under ideal conditions by utilizing actuators to tweak sensor values of airflow and fuel injection.

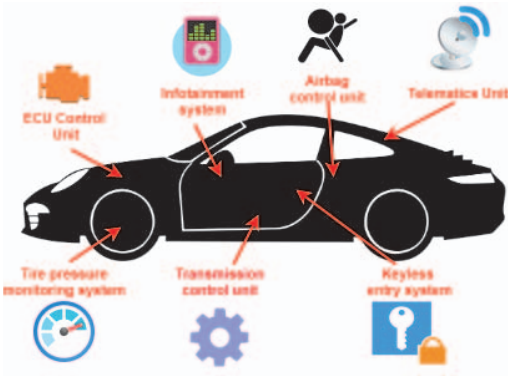

Figure 1: Potential Vehicular Attack Vectors.

While the aforementioned vehicular advancements provide improved performance, stability, and safety, the integration of ECUs opens a new door for malicious attackers to sabotage system behavior. Figure 1 shows the possible attack vectors in a vehicle. Since ECUs accommodate Bluetooth and Wi-Fi capabilities, attackers can exploit these services to gain remote access [1]. Once the intrusion has been established, attackers can utilize the Controller Area Network (CAN) bus, normally utilized for device communication inside the automobile, to maliciously execute their pernicious activities. In one such attack, adversaries were successfully able to reprogram a JEEP's firmware to enable attacker access to the car's functionalities [2]. The attackers were able execute a plethora of mischievous endeavors such as honking the horn, disabling the brakes, and bringing the car to a halt.

The vulnerabilities in ECUs compel examining security measures for securing automotive-based systems. Traditionally, Anti-Virus Software (AVS) has been utilized to deal with malicious software. However, attackers have become accustomed to developing applications to circumvent AVS. This inevitably leads to a cat-and-mouse game between attackers and AVS. Consequently, AVS incurs large computational overhead from frequent updates intended to counter the attacker's evasion techniques. Therefore, researchers have started exploring alternatives to AVS.

One promising solution is the utilization of Hardware Performance Counters (HPCs) which are special purpose registers

978-3-9819263-5-4/DATE21/©2021 EDAA

found in modern processors that track low-level microarchitectural events. Prior research has shown that HPC data, collected from applications, can be used to train Machine Learning (ML) models to classify between benign and malicious programs [3]. Furthermore, since HPCs are integrated into the processor, this solution uses minimal overhead when compared to software equivalents. In this paper, we propose utilizing HPCs in conjunction with ML models for hardware-assisted detection of malicious attacks in ECMs. The key contributions of this paper are as follows:

- We design a malicious parameter modification attack for ECMs considering their characteristics and operation in an automobile.
- We assess the impact of this attack targeting the ECM on a simulated automobile architecture.
- We develop hardware-assisted ECM anomaly detection via ML trained on ECM-based HPC data.
- We evaluate different supervised ML classifiers to detect the parameter modification attack.

To the best of our knowledge, this is the first utilization of hardware-assisted detection via HPCs for securing ECMs. The rest of the paper is organized as follows. Section II introduces prior research for automobile security. Section III provides supplemental information about HPCs and system components. Section IV explains our proposed methodology. Section V presents the experimental results. Lastly, Section VI concludes the paper.

II. Related Work

A. Hardware Performance Counter-based Malware Detection

Malicious applications, more commonly known as Malware, induce a plethora of pernicious operations that jeopardize system security. Repercussions of Malware include unwanted system modifications, unauthorized access to privileged information, and total system shutdown. With AVS incurring large computational overheads, Hardware-based Malware Detection (HMD) has been proposed for Malware detection. This technique employs low level hardware-related events, such as HPCs, to distinguish between benign and malicious software.

One of the first methods of HMDs based on HPCs for Malware detection was proposed in [4]. A Neural Network and K-Nearest Neighbor model were trained on HPC data to classify between benign and malicious applications in [5]. Multiple HPC-based frameworks have been developed employing HMDs for detecting pernicious program. NumChecker measures the system call events through HPCs for distinguishing malicious kernel control-flow modifications [6]. ConFirm is a minimal overhead Malware detection technique that uses a HPC-based comparison approach for detection of firmware modifications with simple control flows [7]. Behavior-based Adaptive Intrusion detection in Networks (BRAIN) used HPCs to aggregate modeled application behavior and network statistics for detection of Denial of Service (Dos) attacks [8]. Additionally, a double stage ML-based framework 2SMaRT was proposed wherein feature selection is employed for HPC utilization and fed into a bi-stage classifier that is efficiently able to detect runtime Malware [9]. Real-time quantification of HPCs for anomaly detection in systems was proposed by [10]. Additional HPC-based Malware detection techniques have been proposed that defends systems against malevolent applications and secures it by ensuring tampering of the hardware architecture is a burdensome endeavor [11]–[19]. Furthermore, an evaluation of the classification abilities of varying ML classifiers and the influence of varying vital model parameters was presented in [20].

B. Securing Electronic Control Units

While HMDs have been utilized for securing IoT systems, they have not been explored for securing vehicles from hostile adversaries. Prior security systems for ECUs have focused on monitoring specific parameters for detecting faults in the system. One such method analyzed the air-per-cylinder flowing into the engine and raised alerts when a specified threshold was surpassed [21]. Additionally, other safeguard methods have been proposed including an undervoltage protection technique for ensuring excessive amounts of energy are not depleted when cranking the engine [22]. Furthermore, engine protection systems have been integrated into the ECM itself in order to monitor vital fluid parameters such as coolant and oil pressure [23], [24]. This enabled continuously tracking for hazardous situations such as the engine overheating. Consequently, the engine would have its maximum speed limited in order to permit the driver to safely drive the vehicle to a servicing center for maintenance.

Although these systems provide a level of safety and transparency in monitoring the vehicle for conventional operational behavior, they fail to provide any meaningful sense of security of attackers aiming to compromise system functionalities. In this paper, we propose HPC-based detection of malicious entities for securing the ECM from parameter modification attacks that aim to induce pernicious unanticipated behaviors.

III. Background

A. Electronic Control Units

Electronic Control Units (ECUs) are embedded systems that control one or more of the electrical systems or subsystems in an automobile. For example, the Engine Control Module (ECM) is tasked with ensuring that the engine receives the appropriate resources it needs, such as air and fuel, to support efficient performance. Other ECUs include the Powertrain Control Module (PCM), the Transmission Control Module (TCM), the Brake Control Module (BCM), the Central Control Module (CCM), the Central Timing Module (CTM), the General Electronic Module (GEM), the Body Control Module (BCM), and the Suspension Control Module (SCM). With a plethora of ECUs monitoring and maintaining specific aspects of the car, communication between components is imperative. A Controller Area Network (CAN) bus is employed for interfacing between different systems [25]. Consequently, this enables smooth vehicular performance and safety. However,

with the advancements in connectivity, new attack doors become available for attackers to exploit.

B. Prior Vehicular Attacks

For hackers, the most significant benefit of the addition of ECUs is the removal of the need to have physical access to the vehicle. They can now target the embedded connectivity modules and utilize it to send the malicious commands to the desired ECUs. This was first highlighted back in 2015 with the infamous Jeep hack [2]. Attackers were able to rewrite the vehicle's firmware to gain full control of the vehicle. Consequently, the driver was powerless to stop the attackers from cutting the brakes, turning on the wipers, blasting the radio, and stopping the engine.

However, Jeep vehicles are not the only automobiles susceptible to these attacks. A wide range of modern vehicles from popular brands, such as Honda, Toyota, and Audi, are vulnerable due to the integration of communication ECUs such as Bluetooth, Wi-Fi, and keyless entry [1]. Furthermore, the integration of the CAN bus for component communication exacerbates this situation as once the attacker has obtained access to the vehicle, they can exploit it to carry out their attacks. In addition to the JEEP attack, white-hat hackers were able to exploit a vulnerability in the Bluetooth ECU in order to obtain remote code execution in a telematic unit of the automobile enabling eavesdropping [1]. For each vehicle, there exists a different attack surface that is dependent on the available ECUs. As automobiles become more sophisticated and integrate additional IoT capabilities, vulnerable exploits will increase exacerbating the situation. As a result, it is imperative that proper security methods are in place to protect the vehicle from malicious attackers in order to secure system security and passenger safety.

C. Hardware Performance Counters

Hardware Performance Counters (HPCs) have gained notable recognition as a promising candidate for detecting malicious attacks. HPCs are special purpose registers that track the count of hardware-related events. Some of these low-level microarchitectural events include Branch-misses, Cache-misses, Bus-Cycles, and LLC-load-misses. HPCs are present in most modern processors, and since they provide useful information, they were originally developed for performance and software tuning. The number of HPCs that are available vary from processor to processor. For example, an Arm Cortex A53 processor, found in a *Raspberry Pi 3 Model B+*, supports up to 20 HPCs. However, all 20 of these HPCs cannot be monitored concurrently. Only a finite number of HPCs can be simultaneously collected, typically a maximum of four to six HPCs based on the processor architecture [5], [6], [26]. While HPCs are utilized for system optimizations, prior research has demonstrated that HPCs in conjunction with ML models can accurately detect Malware [5], [6], [11]. HPCs have a plethora of convenient Malware detection utilization benefits. When considering their integration into many processors, HPCs provide a low overhead solution. Furthermore, unlike their software equivalents, hardware events are much more difficult to compromise. Additionally, HPCs provide impeccable profiling capabilities as the obtained HPC trace of each program is unique. As a result, HPCs are one of the premier solutions for distinguishing Malware and ensuring system security.

D. Measurement metrics

Since we employ ML in conjunction with HPCs for malicious ECU attacks, it is imperative that our models furnish exceptional measurement metrics. In this section, we provide a brief explanation on the measurement metrics that were utilized to evaluate our ML models. For ML classification, we consider malicious applications as the positive class and benign applications as the negative class. Consequently, we define a True Positive (TP) as a pernicious program correctly labeled as the positive class and a False Positive (FP) when incorrectly labeled as the negative class. Similarly, we define a True Negative (TN) as a benign program accurately labeled as a negative class and a False Negative (FN) when erroneously labeled as the positive class. We explicitly looked at three performance metrics furnished from our ML models: accuracy, precision, and recall.

Accuracy is the percentage of the number of correctly classified predictions to the total number of predictions, which we represent below as:

$$Accuracy = \frac{No.\ of\ (TP\ +\ TN)}{No.\ of\ (FP\ +\ FN\ +\ TP\ +\ TN)} \quad (1)$$

Precision indicates the amount of positive class classifications that were correct.

$$Precision = \frac{No.\ of\ TP}{No.\ of\ FP\ + No.\ of\ TP} \quad (2)$$

Recall is the ratio of actual positives to the total number of positive predictions.

$$Recall = \frac{No.\ of\ TP}{No.\ of\ FN\ + No.\ of\ TP} \quad (3)$$

IV. Methodology

In this section, we provide an overview of the ECU attack we developed to disrupt normal operational functionality in the ECM. In addition, we describe our proposed HPC-based detection methodology for anomaly detection.

A. Malicious Signal Attack Development

The primary attack used for this experiment is external signal-based, in which a disruptor signal was introduced into the ECU. The premise of the disruptor signal is to interrupt the constant non-malicious signal from the sensor to the ECU. The disruptor signal achieves this task by sending a value of constant zeros for one second. In normal circumstances, the sensor would be sending a tooth value of the crankshaft which ranges from 1 to 24. However, during this attack, the ECU receives 0s. This small but significant change in the input signal causes the piston's valves to stop opening and closing

for air. If this were to occur, the engine would likely halt due to the lack of air. Furthermore, the engine would have a high probability of breaking down due to an excess of oil not being combusted in the piston chamber. Consequently, these changes would likely cause permanent damage to the vehicle's engine. In addition, during the attack, the automobile would cease to run as it would die on the spot. In order to detect this malicious attack, we propose utilizing HPC-based detection to prevent any undesired behavior in the automobile.

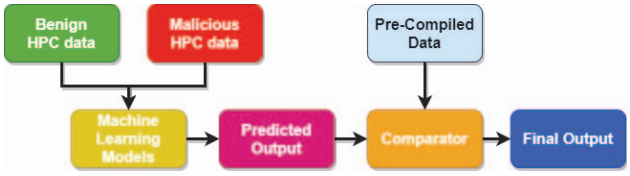

Figure 2: High Level Flow Diagram of Proposed Attack Detection Methodology.

B. HPC-based Attack Detection

Our proposed HPC-based ECU anomaly detection consists of four steps as shown in Figure 2. First, we collect HPC data for profiling conventional operational behaviors, which we label as the negative class. Subsequently, we execute the pernicious attack explained in Section IV-A, and capture HPC data for this anomaly inducing situation which gets labeled the positive class. Next, with the obtained HPC data, we train a ML classifier to classify between benign and malicious samples. Third, to test our model for robustness, we collect additional HPC data for both classes and classify them through our trained ML model. Finally, the furnished results are analyzed and compared with the expected results in order to determine our model's competency.

V. Experimental Results

A. Experimental Setup

To develop a proof of concept for our proposed HPC-based ECU anomaly detection, we evaluated our methodology on a developed attack within the ECM. An *Arduino Uno* was used to emulate the crankshaft position sensor which was written in *C++*. The main operation performed by the emulator was to output the position of the crankshaft for a given revolutions per minute (RPM) which was fed into a *Raspberry Pi 3 model B+* that emulated the ECM. The ECM emulator was written in *Python3*, and it properly positioned the four piston valves based on the crankshaft value. The input data indicates which pistons are being opened and closed. The emulator ran the simulation 5,000 times for 2,500 benign and 2,500 malicious runs. We employed *Perf* to collect the HPCs from the *Raspberry Pi*. These samples were collected at four different RPMs to demonstrate the robustness of our model at different engine throughputs. We analyzed which HPCs had the highest variance in performance through these differing RPMs. In Figure 3, we plot the occurrences of HPC *LLC-load-misses*. For malicious samples, the variance is much higher. This trend was continued for HPCs *CPU-cycles*, *Branch-instructions*, *Instructions*, and *L1-dcache-loads*. Subsequently, we utilize these HPCs for training our ML models written in *Python3* using the scikit-learn library [27]. An additional 5,000 runs (2500 benign and 2500 malicious) were collected for testing the model. We tested five different machine learning models to analyze their measurement metrics, as mentioned in Section III-D. For all experiments, we simulated the ECM under different RPM scenarios. First, we ran the engine at 100 RPM. Second, the simulation was recorded at 1200 RPM, which is a steady state. Third, we tested when the engine is being pushed at a standard 40 miles per hour equating to 2000 RPM. Lastly, we evaluated at 4000 RPM to produce a scenario where the engine is pushed harder than usual.

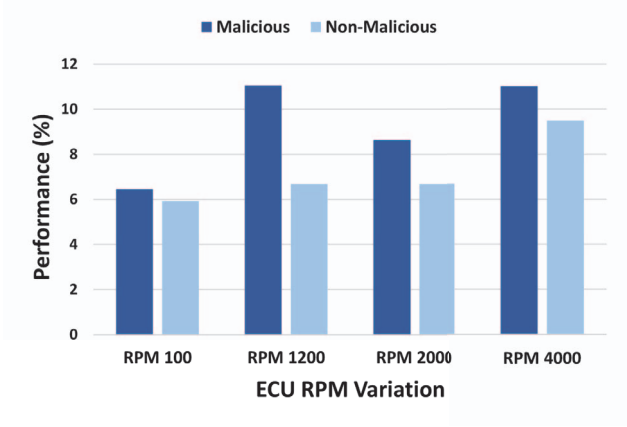

Figure 3: LLC-load-misses Throughout All RPM Test Cases.

B. Experimental Results

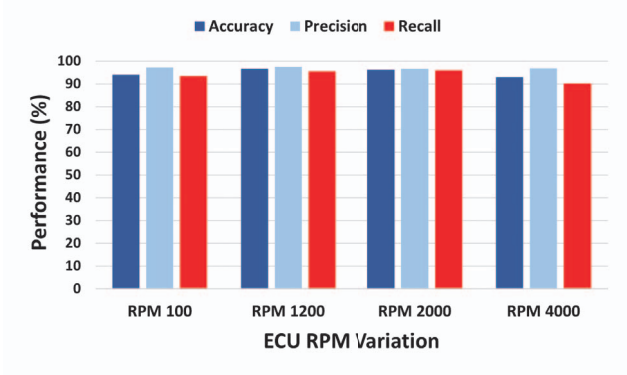

Figure 4: Results Attained with Auto Encoder.

1) Auto Encoder: When utilizing an Autoencoder classifier, the best measurement metrics of 96.7% accuracy, 97.6% precision, and 95.9% recall was achieved at 1200 RPM. As showcased in Figure 4, all performance metrics remained stable throughout all varying RPMs. When analyzing the precision of the model, the Autoencoder was able to identify

nearly all malicious samples with a range of 96.7% to 97.6%. This stability speaks on the efficiency and accuracy the Autoencoder provides for anomaly detection. Furthermore, our model demonstrates its robustness as varying RPMs did not induce a degradation in measurement metrics.

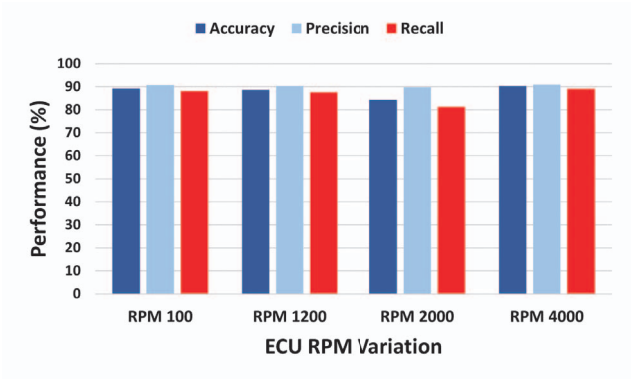

Figure 5: Results Attained with K-Nearest Neighbor.

2) K-Nearest Neighbor: Similar to Section V-B1, we repeated the experiment using a K-Nearest Neighbor (KNN) classifier shown in Figure 5. The maximum accuracy was obtained at 4000 RPM. At this RPM, the accuracy was 90.2%, precision was 90.9%, and recall was 89.1%. Throughout all the tested RPMs, the model maintained a minimum accuracy of 84.2%. While there is a slight dip in accuracy at 2000 RPM, the precision remained stable, which is more imperative as our purpose is to identify malicious samples.

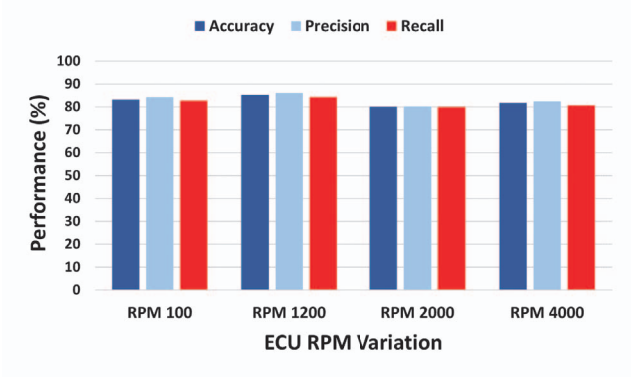

Figure 6: Results Attained with Support Vector Machine.

3) Support Vector Machine: The results for utilizing a Support Vector Machine (SVM) classifier are shown in Figure 6. The model furnished a maximum accuracy of 85.3%, precision of 86.1%, and recall of 84.2% at 1200 RPM. As the RPM ramps up to 2000 and 4000, the measurement metrics decrease. Since we desire to identify the malicious samples for ensuring system security, we particularly analyze the drop in precision. At 2000 RPM, the precision was at its lowest with a value of 80.1% for a drop of 4.98% from 1200 RPM. This drop in performance is attributed to the increase in HPC counts, especially for HPC *Instructions* resulting in the model's inability to separate overlapping samples.

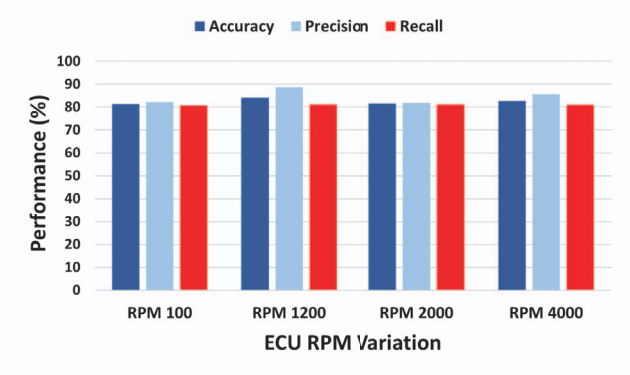

Figure 7: Results Attained with Decision Tree.

4) Decision Tree: For the Decision Tree (DT) classifier, Figure 7 presents the obtained results. The best measurement metrics were obtained at 1200 RPM. The model furnished 84.2% accuracy, 88.7% precision, and 81.2% recall. While the classifier demonstrated decent anomaly detection in ECMs at 1200 RPM, in general, the model is not as robust as required for securing ECUs. When solely analyzing the precision of the model, the largest drop occurred at 2000 RPM. An 8.4% decrease was observed as it dropped from 88.7% to 81.2% which we attribute to increased HPC counts inducing analogues samples.

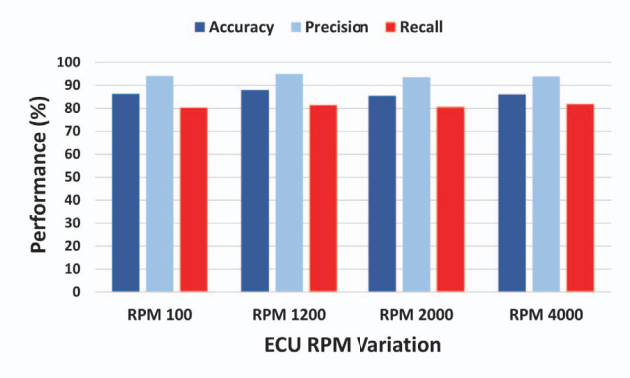

Figure 8: Results Attained with Random Forest.

5) Random Forest: The evaluation of the Random Forest (RF) classifier for anomaly detection is shown in Figure 8. For this model, we obtained the best results at 1200 RPM with an 87.9% accuracy, 94.8% precision, and 81.3% recall. Unlike the DT from Section V-B4, the RF classifier furnished consistent results regardless of the utilized RPM. With high precision values, this model was successfully able to identify a majority of the induced anomaly samples.

C. Analysis of All Models

Figure 9 showcases the accuracy for all utilized models at each RPM. The Autoencoder furnished the best results

as it maintained at least 93.1% accuracy across all RPMs. However, the KNN and RF models were both close, with average accuracies of 87.75% and 86.12% for varying RPMs. When comparing the best and worst classifier accuracy, the Autoencoder, with a value of 96.7%, had 11.4% more accuracy than the SVM which had 85.3%. When analyzing these results, it is not surprising that the Autoencoder preforms the best as it exhibits exceptional anomaly detection capabilities.

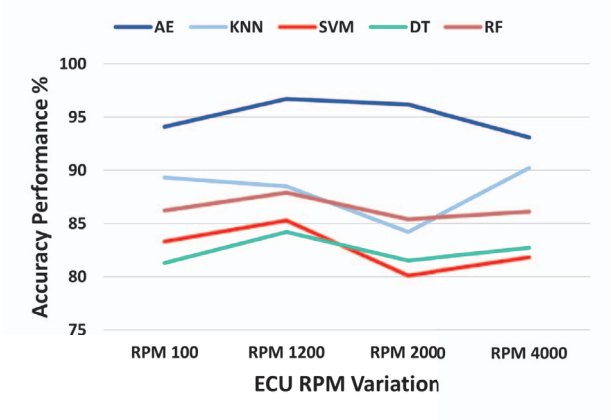

Figure 9: Accuracy Across All Machine Learning Models.

VI. Conclusion

While the modernization of automobiles to utilize computerized ECUs has produced user-convenient features, attackers have gained new entries for obtaining remote access. In this paper, we proposed a HPC-based ECU anomaly detection method for identifying parameter modification attacks within the ECM. Our Autoencoder model furnished a high classification accuracy of 96.70% reaffirming the robustness of our technique for detecting pernicious endeavors against the ECM. For future work, we plan to extend our proposed technique to distinguish other attacks in additional ECUs such as the braking module. Furthermore, we additionally intend to explore the employment of complex learning algorithms, such as Recurrent Neural Networks (RNN), and evaluate their classification performance.

VII. Acknowledgment

This research is partly supported by Ford Motors University Research Program Award# 2018-8058R.

References

[1] "Remoteattacksurfaces.pdf," http://ftpcontent.worldnow.com/wbbh/documents/Remoteattacksurfaces.pdf, (Accessed on 12/02/2020).

[2] "Hackers remotely kill a jeep on the highway—with me in it — wired," https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/, (Accessed on 12/02/2020).

[3] M. Ozsoy *et al.*, "Hardware-based malware detection using low-level architectural features," *IEEE Transactions on Computers*, vol. 65, no. 11, pp. 3332–3344, 2016.

[4] C. Malone *et al.*, "Are hardware performance counters a cost effective way for integrity checking of programs," in *Proceedings of the sixth ACM workshop on Scalable trusted computing*, 2011, pp. 71–76.

[5] J. Demme *et al.*, "On the feasibility of online malware detection with performance counters," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3, pp. 559–570, 2013.

[6] X. Wang *et al.*, "Reusing hardware performance counters to detect and identify kernel control-flow modifying rootkits," *IEEE TCAD*, vol. 35, no. 3, pp. 485–498, 2015.

[7] X. Wang, *et al.*, "Malicious firmware detection with hardware performance counters," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 2, no. 3, pp. 160–173, 2016.

[8] V. Jyothi *et al.*, "Brain: Behavior based adaptive intrusion detection in networks: Using hardware performance counters to detect ddos attacks," in *29th International Conference on VLSI Design and 15th International Conference on Embedded Systems*. IEEE, 2016, pp. 587–588.

[9] H. Sayadi *et al.*, "2smart: A two-stage machine learning-based approach for run-time specialized hardware-assisted malware detection," in *Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2019, pp. 728–733.

[10] P. Krishnamurthy *et al.*, "Anomaly detection in real-time multi-threaded processes using hardware performance counters," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 666–680, 2019.

[11] M. B. Bahador *et al.*, "Hpcmalhunter: Behavioral malware detection using hardware performance counters and singular value decomposition," in *IEEE ICCKE*, 2014, pp. 703–708.

[12] X. Wang *et al.*, "Hardware performance counter-based malware identification and detection with adaptive compressive sensing," *ACM Transactions on Architecture and Code Optimization*, vol. 13, no. 1, p. 3, 2016.

[13] A. Tang *et al.*, "Unsupervised anomaly-based malware detection using hardware features," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2014, pp. 109–129.

[14] N. Patel *et al.*, "Analyzing hardware based malware detectors," in *ACM/EDAC/IEEE Design Automation Conference*. IEEE, 2017, pp. 1–6.

[15] H. Sayadi *et al.*, "Ensemble learning for effective run-time hardware-based malware detection: A comprehensive analysis and classification," in *ACM/ESDA/IEEE Design Automation Conference*. IEEE, 2018, pp. 1–6.

[16] A. P. Kuruvila *et al.*, "Defending hardware-based malware detectors against adversarial attacks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2021.

[17] S. Islam *et al.*, "Nd-hmds: Non-differentiable hardware malware detectors against evasive transient execution attacks," in *2020 IEEE International Conference on Computer Design(ICCD)*. IEEE, 2020.

[18] L. Zhou *et al.*, "Hardware-assisted rootkit detection via on-line statistical fingerprinting of process execution," in *Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2018, pp. 1580–1585.

[19] S. M. P. Dinakarrao *et al.*, "Lightweight node-level malware detection and network-level malware confinement in iot networks," in *Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2019, pp. 776–781.

[20] A. P. Kuruvila *et al.*, "Analyzing the efficiency of machine learning classifiers in hardware-based malware detectors," in *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2020, pp. 452–457.

[21] "14983802549489155 08-08050841," https://patentimages.storage.googleapis.com/81/1f/6b/f14ec410efde40/US8050841.pdf, (Accessed on 12/03/2020).

[22] "Us7277781.pdf," https://patentimages.storage.googleapis.com/b3/20/f5/70af80b3e3788a/US7277781B2.pdf, (Accessed on 12/03/2020).

[23] "14983900211552819 23-05070832," https://patentimages.storage.googleapis.com/ca/b4/ca/cde094838764be/US5070832.pdf, (Accessed on 12/03/2020).

[24] "14990727708053001 69-07349794," https://patentimages.storage.googleapis.com/85/68/81/15c6f524b1ddb0/US7349794.pdf, (Accessed on 12/03/2020).

[25] "Automating ecu identification for vehicle security," https://nsl.cse.unt.edu/sites/default/files/biblio/documents/AutomatingECUIdentificationVehicleSecurity.pdf, (Accessed on 12/03/2020).

[26] K. Basu *et al.*, "A theoretical study of hardware performance counters-based malware detection," *IEEE TIFS*, vol. 15, pp. 512–525, 2019.

[27] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.