An Intrusion Detection System Against Rogue Master Attacks on gPTP

Alessio Buscemi*, Manasvi Ponaka ‡, Mahdi Fotouhi*, Florian Jomrich†, Christian Koebel†, Thomas Engel*
*‡ Faculty of Science, Technology and Medicine, University of Luxembourg, †Honda R&D Europe (Germany) GmbH
*{$name.surname$}@uni.lu, †{$Name\_Surname$}@de.hrdeu.com, ‡{$name.surname.001$}@student.uni.lu

***Abstract*— Due to the promise of deterministic Ethernet networking, Time Sensitive Network (TSN) standards are gaining popularity in the vehicle on-board networks sector. Among these, Generalized Precision Time Protocol (gPTP) allows network devices to be synchronized with a greater degree of precision than other synchronization protocols, such as Network Time Protocol (NTP). However, gPTP was developed without security measures, making it susceptible to a variety of attacks. Adding security controls is the initial step in securing the protocol. However, due to current gPTP design limitations, this countermeasure is insufficient to protect against all types of threats. In this paper, we present a novel supervised Machine Learning (ML)-based pipeline for the detection of high-risk rogue master attacks.**

***Index Terms*—Time Sensitive Networking, Cybersecurity, Connected Vehicles, Automotive Ethernet**

I. Introduction

Cyber attacks against in-vehicle networks have been disregarded for many years due to the necessity for the adversary to have physical access to the vehicle, thus making the attack scenario unrealistic. However, the recent advancements in the field of Connected Automated Mobility (CAM) raise serious concerns regarding the security of in-vehicle networks. A number of wireless attacks against vehicles have been carried out successfully by researchers [1]–[3]. In 2015, Miller and Valasek [3] drove a Jeep Cheerokee off the road by exploiting the remote injection of Controller Area Network (CAN) messages through the infotainment system, thus showing the devastating potential of wireless attacks.

As Original Equipment Manufacturers (OEMs) transition from traditional networks towards automotive Ethernet to provide a wide range of new automotive services [4], cybersecurity concerns become increasingly important. This further emphasises the need to secure gPTP. gPTP – IEEE 802.1AS [5] – is the automotive profile of Precision Time Protocol (PTP) – IEEE 1588 [6], the time synchronization protocol of TSN.

For clock dissemination, gPTP employs a master-slave architecture. The slave clocks are synchronized to the Grand Master (GM), which is the root timing reference clock. The GM communicates synchronization information to the clocks on its network segment. The GM is not fixed but is elected dynamically in accordance with the Best Master Clock Algorithm (BMCA). This method ranks the clocks based on their characteristics, such as priority number, class etc. The clock with the highest ranking becomes the GM iteratively. After the GM is elected, a synchronization *master* clock is chosen for each network segment in the system.

It is to be noted that, in the case of a static configuration, the GM is preliminary set by the user and does not change. Hence, the BMCA cannot be used. Nonetheless, not allowing the GM to be chosen by the network in an automated fashion poses concerns regarding the network robustness (i.e. if the GM node is disrupted, the whole network experiences a Denial of Service (DoS)). For this reason, in this work, we assume a scenario in which the GM is chosen dynamically.

A Boundary Clock (BC) transmits precise time to the other segments to which it is linked. In gPTP, management and synchronization are accomplished through the following packets i) Event messages – time-related information that is utilized to synchronize clocks across the network ($Sync$, $Follow\_Up$, $Delay\_Req$, and $Delay\_Resp$), ii) General messages – used to compensate communication medium delays ($Pdelay\_Req$, $Pdelay\_Resp$, and $Pdelay\_RespFollow\_Up$), iii) Announce messages —- used to build a clock hierarchy and elect the GM based on the BMCA ($Announce$).

This work is a first attempt to investigate Intrusion Detection System (IDS) for gPTP based on real-world attacks conducted on a physical testbed. We introduce a novel IDS capable of detecting a class of rogue master attacks [7]. A rogue master attack is an attack in which an adversary exploits the BMCA to acquire the GM position and send malicious synchronization messages. An IDS monitors network traffic for suspicious activity and issues alerts when such activity is discovered. IDS provide additional security by detecting attacks which bypass hard coded security controls.

The contributions of this work can be summarized as follows:

- We analyze the impact that rogue master attacks have on the clocks of the target slaves. Based on this evaluation, we design a set of features aiming at distinguishing the behavior of the clock in normal and attack scenarios;
- We present a novel IDS capable of detecting rogue master attacks gradually which causes false time in the End Station (ES);
- We analyze strengths and weaknesses of the presented approach and discuss possible improvements towards the commercial usage of IDS for TSN.

II. Background and Related work

In this section, we introduce key concepts to the comprehension of the work and results in the manuscript. First, we present a detailed explanation of the synchronization mechanism

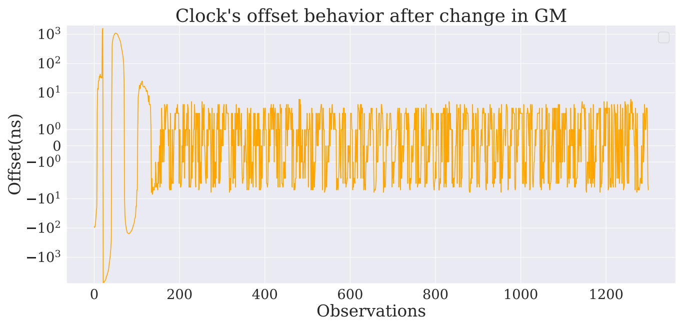

Figure 1. Clock time offset between slave and GM after a new GM is elected.

between slaves and GM. Then, we address relevant research on in-vehicle IDS.

A. *Clock synchronization*

A gPTP network is composed of two types of nodes, ES and bridges. ES have only one port connected to the network, while bridges typically have multiple ports. Each port in the network can be *slave*, i.e. its role is to receive packets, or *master*, i.e. its role is to send packets. The ES can assume one of two roles, GM or slave. All the ES, except for the GM, have one port in slave status. In bridges, the ports serve to connect to the ES and to other bridges.

To spread its reference clock, the GM periodically sends $Sync$ and $Follow\_up$ messages through its master ports to all the nodes that are directly connected to it. These packets then transit in the rest of the network passing from master to slave ports.

The bridges receiving these packets need to correct the received synchronized time by adding a $CorrectionField$. The $CorrectionField$ is computed by taking into consideration i) the forwarding delay, i.e. the residence time incurred at the bridge, and ii) the transit delay, i.e. the time needed for the information to propagate through the communication path from the GM to the ES or the bridge. PTP clock synchronization assumes that the network delay between slaves and their master is symmetric.

The clock *time offset* is the difference between the time of the slave and the GM. When an ES takes the role of the GM master, the slaves needs to firstly synchronize to it. In the synchronization phase, there are high fluctuations in the offset between the two nodes, which is reduced with the additional sending of synchronization packets. According to the PTP standard, two clocks are synchronized if the offset is consistently between -100 and 100 ns. Figure 2 shows an example of synchronization process between a slave and a GM.

To avoid confusion between a) the offset in the timestamp contained in malicious $Sync$ packets, and b) the offset between the slave's clock and the GM's clock (see Section II-A), hereafter we will refer to the former as *timestamp offset*, and the latter as *clock offset*.

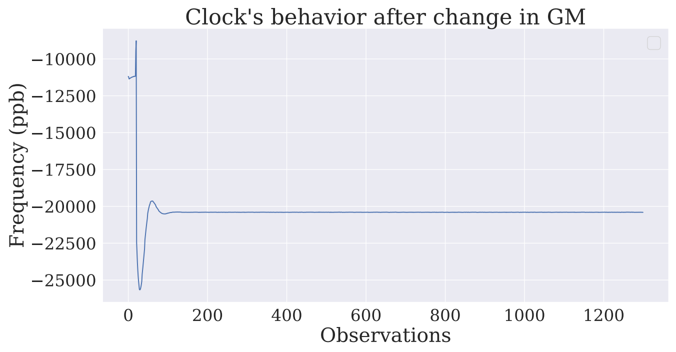

Figure 2. Example of a slave clock relative frequency when a new GM is elected.

Despite aiming at providing high time precision, the quartz clock embedded in each commercial hardware has its own unique default frequency. It follows that, based on the information regarding propagation delay and offset, the slave clocks continuously adjust their frequency to be synchronized with the GM. Therefore, synchronizing with a GM implies that the frequency of the slave clock drifts from its nominal output frequency. The difference between the actual frequency and the nominal frequency can be defined as the *relative frequency*. The relative frequency can be measured in parts per million (ppm) or parts per billion (ppb). Figure 2 shows an example of the clock relative frequency when a new GM is elected. Similarly to the offset, after an initial synchronization phase characterized by a highly volatile behavior, the relative frequency stabilizes around a specific value.

In this work, we investigate the impact that rogue master attacks have on the offset and relative frequency of slave clocks.

B. *In-vehicle IDS*

IDS for in-vehicle networks have been widely studied in the recent years. They are popular because they do not require substantial changes to the communication protocol, and add low or no overhead to the communication stream. IDS can be primarily divided into *signature-based* and *anomaly-based* [8].

Signature-based systems establish a taxonomy of known attacks and scan the CAN traffic to find a match with any of them [9]. Anomaly-based systems, instead, observe the Electronic Control Units (ECUs) and the traffic in search of behaviors that deviate from the usual nature of the network [10]–[12]. Anomaly-based IDSs generally achieve lower detection performance than the signature-based ones but are capable of detecting unknown attacks.

Features typically exploited by in-vehicle IDS are based on i) physical characteristics of the ECUs' clocks, ii) timing of the transiting packets, iii) features-based, i.e. specific patterns are extracted from the traffic and the ECUs.

In this work, we propose a prototype of signature IDS which exploits features extracted from the ES clocks.

III. Attack

In this section, we describe the threat model and provide detailed information regarding the rogue master attacks performed

Table I
EXAMPLE OF ROGUE MASTER ATTACK WHERE AN OFFSET OF 100 NS IS ADDED TO EACH $Sync$ PACKET.

Sync packet n.	Original Ts	Tampered ts
1	T_1	F_1 = T_1 + 100 ns
2	T_2 = T_1 + $\sigma_{1,2}$	F_2 = T_2 + 2*100 ns
3	T_3 = T_2 + $\sigma_{2,3}$	F_3 = T_3 + 3*100 ns
	...	
N	T_N = T_{N-1} + $\sigma_{n,n-1}$	F_N = T_{N-1} + n*100 ns

in this work.

A. Threat Model

The threat model is the one of an internal attacker, i.e. an adversary capable of manipulating legitimate traffic in the network and/or generating its own traffic while making it appear legitimate to the attacked nodes. To simulate an attacker with such capabilities, in this work we employ ptpatk, a tool built by us, which is based on LinuxPTP (version 3.1-00116-g24220e8) (see [7]). LinuxPTP is an open-source repository for the development of PTP on Linux [13].

B. Rogue Master Attack

In the rogue master attack, the malicious slave appears to be time-aware and enters the gPTP domain by replying to the $Pdelay\_req$ packets that have been correctly received from the switch.

The malicious node listens to the network for $Pdelay\_req$ packets and creates tampered $Pdelay\_resp$ and $Pdelay\_resp\_follow\_up$. Additionally, the malicious node generates $Pdelay\_resp$ and $Pdelay\_resp\_follow\_up$ packets. In order for the domain to be effectively joined, the adversary must have the switches calculate a minimal propagation delay.

After joining the domain, the adversary works toward the goal of obtaining the position of GM. The malicious node transmits high-priority $Announce$ packets in an attempt to convince other nodes in the domain that its clock is the most accurate available clock; if successful, it is selected as the GM. After the malicious node has reached the state of GM, it can send tampered gPTP packets aiming at a diverse set of hijackings.

In this work, we examine a special instance of a rogue master attack in which the adversary transmits arbitrary timestamps to the slaves in the form of $Sync/Follow\_up$ packets. We consider the case in which the adversary starts the attack immediately after gaining the GM position. The consecutive $Sync$ packets sent by GM contain a timestamp with a predefined offset in order to progressively induce *false time* in the slave. For a better comprehension, let us assume an adversary who keeps adding 100 ns to the timestamp of each new $Sync$. Being $\sigma_{i,j}$ the time interval between the $Sync$ packet i and $Sync$ packet j, the tampered timestamp will drift from the original timestamp, as shown in Table I.

For the sake of readability, hereafter we will refer to an attack which continuously adds an offset of x ns to the $Sync$ packets as a *seq +x* attack. For instance, if the adversary aims at adding extra 1000 ns to the timestamp of each $Sync$ packet, we refer to it as a *seq +1000 ns* attack.

IV. IDS

In this section, we describe the characteristics of our novel IDS. In our case study, each ES has its own instance of the IDS. The local IDS continuously observes the clock of the ES thanks to the information provided by its daemon, with the aim of identifying unusual behavior. The detection pipeline is divided into two parts:

- **Phase 1** – it starts when there is a GM change and the slave ES is trying to synchronize, as described in Section II-A.
- **Phase 2** – it occurs when the slave is already synchronized with the GM.

A. Phase 1

The pseudocode of Algorithm 1 describes the steps of Phase 1. The algorithm requires i) a daemon D mounted on the ES, which handles the incoming gPTP traffic at software level and monitors its clock, ii) an integer S defined by the user, which corresponds to the size of the window considered for the change of phase, iii) a ML-model $M1$, which serves as the intrusion detector, previously trained on clock data collected during synchronization.

First, a list L is initialized (line 1). This list (or any other mutable data structure) will contain the measurements related to the slave clock. Then, the synchronization loop starts (line 2). Every time a new clock measurement is collected from the slave's daemon, it is transmitted to the IDS along with the GM address (line 3). Subsequently, a check on the GM is performed (line 4). If the GM has changed, Phase 1 is reinitialized (line 5). Otherwise, the clock measurement is added to L (line 7), and a window W containing the last S clock's offsets is extracted (line 8). The window W is then used to assess whether the synchronization is still taking place (line 1). If not, the loop is exited.

After the synchronization phase is complete, a sample is extracted in accordance with the process described in Section IV-C. This sample is fed to the intrusion detection classifier (line 10), which predicts whether an intrusion occurred during the synchronization or not (line 11). If an attack is detected, a warning is issued. Otherwise, Phase 2 is started (lines 12-16).

B. Phase 2

Phase 2 requires in input, i) the daemon D, ii) the window size S, iii) the last window W extracted in Phase 1, iv) a previously trained ML model $M2$, which serves as intrusion detector.

Similarly to Phase 1, while the GM does not change, clock measurements are extracted from the daemon and appended to L (lines 1-3). Then, iteratively, a sample is extracted from the last window W and passed to $M2$ (lines 4-6).

Differently from Phase 1, if an attack is detected, an alert is not immediately issued. By contrast, the sample is passed to a database $anomaly\_DB$ (lines 7-9). At every new entry, an evaluation is made on the database to determine whether an intrusion is taking place and, if it is the case, issue an alert (line 10-12). In other words, instead of considering the prediction related to a single window, the IDS conducts its final evaluation on a number of samples. This approach aims at reducing false alarms.

IDSs for safety-critical applications typically prioritize *recall* (the % of real attack instances detected by the model over the total number of real attack instances) above *precision* (the % of real attack instances over the total number of attacks instances detected by the model) [14]. However, given that one of the primary use-cases of gPTP is in-vehicle networking, false alarms are non-negligible.

Let us consider an IDS with an accuracy lower than 100%, e.g. 99.8%. In light of the fact that a fresh clock measurement is around 125 ms, the IDS would evaluate eight samples per second. Consequently, we would anticipate an alert to be triggered approximately every minute. Whether the alerts would be addressed to the drivers themselves, external operators or Security Operation Center (SOC) (a plausible scenario for Connected Automated Vehicles (CAVs)), a reaction could be initiated, thereby potentially disrupting the vehicle's normal functioning. In Section V-E we discuss in detail pros and cons of this approach.

Finally, if no attack has been detected and the GM changes, Phase 1 is started again.

C. Features Engineering

In intrusion detection, ML classifiers should be trained using data which represents the inherent differences between the behavior of the system's components in normal time and under attack. In this work, we initially considered two different sources for the extraction of such features, the gPTP traffic and the behavior of the slaves' clocks. Given that rogue master attacks aim at false time and desynchronization, which are two aspects directly affecting a slave's clock, we decided to focus our investigation towards this source. In this regard, we considered the clock offset and relative frequency, which are described in Section II-A.

Regarding Phase 1, we observed that the behavior of the clock can vary greatly even between different instances of normal gPTP traffic (as described in Section II-A). Nonetheless, it typically takes longer for a slave to synchronize with the GM if it is under attack. Additionally, it seems that, during the attack, the fluctuations in the offset and frequency are generally wider than they are at normal time.

For what concerns Phase 2, we observed that:

- Even when they are under attack, the slaves still exhibit fluctuations in the offset clock that are comparable to the behavior that is expected of them. This shows that the clocks manage to maintain the synchronization despite the offset added by the adversary in the $Sync$ packets' timestamp;
- During an attack, the normal behavior of the clock is interrupted by abrupt changes in the offset, which sometimes lead to temporary desynchronization. Based on the analysis of a number of attacks which add different magnitudes of offset to the $Sync$ packets' timestamp, we observed that the higher the offset added by the adversary, the wider are these oscillations.
- The slave's frequency greatly depends on the GM. Considering that each node has its own default frequency, different GMs cause different relative frequency in the slave. Given that the purpose of a rogue master attack is to grant the GM position to a node which normally would not achieve it, the frequency can be a clear indicator of an attack;

Algorithm 1

```
Input: Daemon $D$, Window Size $S$, Model $M1$
Output: Prediction $P$
 1: $L \leftarrow []$
 2: while is_syncing($W$) do
 3:     clock_measurement, GM ← $D$.get_clock_data()
 4:     if $GM$.has_changed() then
 5:         Phase1()
 6:     end if
 7:     $L$.append(clock_measurement)
 8:     $W \leftarrow$ last_clocks($L$, $S$)
 9: end while
10: sample ← extract_features(clock_list)
11: $P \leftarrow$ predict($M1$, sample)
12: if $P$ == 'Attack' then
13:     send_warning()
14: else
15:     Phase2($W$, $L$, $S$)
16: end if
```

Algorithm 2

```
Input: Daemon $D$, Window Size $S$, last_window $W$, Model $M2$
Output: Prediction $P$
 1: while not $GM$.has_changed() do
 2:     clock_measurement, GM ← $D$.get_clock_data()
 3:     $L$.append(clock_measurement)
 4:     $W \leftarrow$ last_clocks($L$, $S$)
 5:     sample ← extract_features($L$)
 6:     $P \leftarrow$ predict($M2$, sample)
 7:     if $P$ == 'Attack' then
 8:         $anomaly\_DB$.append($P$)
 9:     end if
10:     if attack_detected(anomaly_DB) then
11:         send_warning()
12:     end if
13: end while
14: Phase1($D$, $W$, $S$)
```

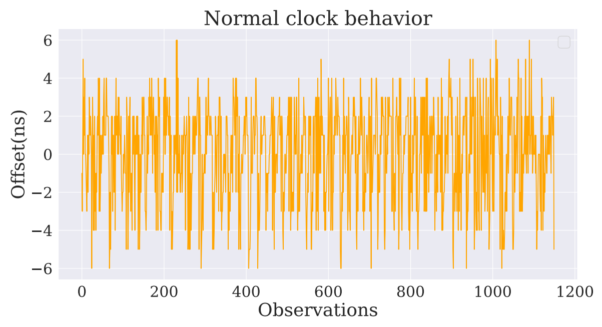

Figure 3. Time series of a slave clock offset, as observed during a data collection session in which no attack is performed.

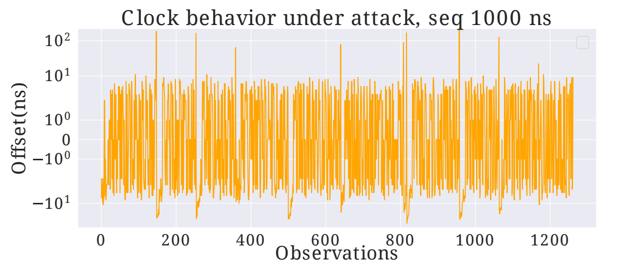

Figure 4. Time series of a slave clock offset, as observed during a data collection session in which a rogue master continuously adds 1000 ns to the $Sync$ packets timestamp.

- Similarly to the offset, there are sudden oscillations in the clock relative frequency when the slave is under attack.

Figure 3 and Figure 4 illustrate the differences between the clock's offset at normal time and during a seq +1000 ns attack. Figure 4 highlights that, during the attack, the oscillations are comparable to the normal behavior, but sudden increase occur occasionally.

Figure 5 and Figure 6 show the differences between the clock's relative frequency at normal time and under a seq+ 1000 ns attack. The first aspect to be noticed is that the relative frequency fluctuates around a different mean. In addition, similarly to the offset, occasionally the frequency displays wide sudden oscillations.

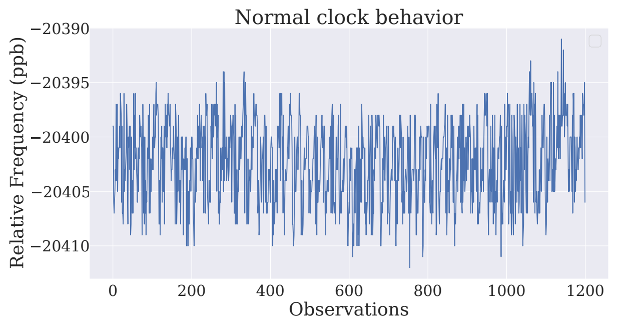

Figure 5. Time series of a slave clock relative frequency, as observed during a data collection session in which no attack is performed.

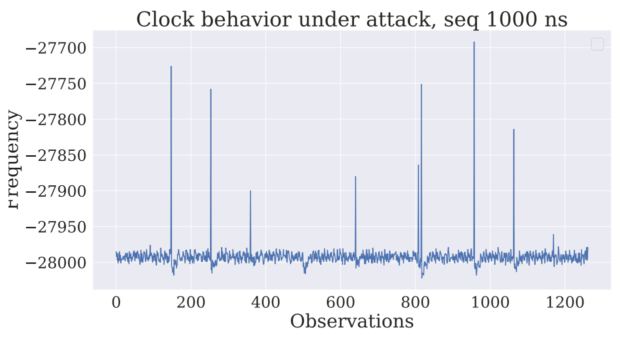

Figure 6. Time series of a slave clock relative frequency, as observed during a data collection session in which a rogue master continuously adds 1000 ns to the $Sync$ packets timestamp.

Following the findings presented in this subsection, we designed the following set of features for both phases, i) Frequency related features – mean, standard deviation (std), maximum, minimum, ii) Offset related features – std, mean and median of the absolute values, maximum, minimum. Furthermore, for Phase 1, we also consider the duration. In fact, we have empirically discovered that, typically, during a rogue master attack the synchronization phase lasts slightly longer than in a normal scenario.

V. Performance Evaluation

In this section, we describe the environment in which the evaluation of our IDS was carried out, and we evaluate the results that were obtained.

A. Scenarios

In this work, we consider an automotive scenario in which the IDS requires a complete mapping of the relative frequency of each slave clock according to each possible GM.

In this scenario, the IDS can be trained to identify relative frequencies which are unexpected, thus raising a suspicion of anomaly. As demonstrated by the results presented in Section V-E, features built on the clock frequency highly contributes to the detection performance. However, this scenario presents two limitations:

- **Continuous Update –** clocks are subject to environmental conditions, e.g. temperature, as well as decay [15]. For this reason, the frequency references have to be continuously updated;
- **Scalability –** Let us assume that each ES in the network is a legitimate candidate for the GM position. Then, in a in-vehicle gPTP network with N ECU attached, the IDS would need to include $N(N-1)$ slave/GM frequency combinations.

For these reasons, in this work we also evaluate the IDS under the assumption that a mapping of the slave/GM frequencies is not possible.

B. Testbed

We built a physical testbed based with real gPTP-capable switches. For the ES Accelerated Processing Units (APUs), we have used LinuxPTP (version 3.1-00116-g24220e8), an open-source repository for the development of PTP on Linux [13]. Finally, to ensure that our evaluation of the attacks relies only on the internal clocks of the testbed, we have blocked the access of the devices to external time resources, such as NTP. The hardware components of the physical testbed are listed as follows:

- Three gPTP capable Netgear switches (GS716Tv3 ProSafe 16-port Gigabit Ethernet Smart Switch, 6.3.1.19-39, B1.0.0.4) [16].
- Four APUs apu2e4 [17] each with three Intel Ethernet Controller i210 and running Ubuntu 16.04.

Further information regarding the employed testbed is provided in [7].

C. Dataset

To test the IDS, we collected clock measurements for a total time of 2000 min ($\sim$ 33 hours) for each APU in the testbed. Specifically, the data was collected in 400 sessions of 5 min each, 100 for each of the three considered attacks (seq +100, seq +1000, seq +1000), and 100 representative of the normal network behavior. At the beginning of each session, the LinuxPTP daemons are started on all APUs. Then, the data is collected and stored into logs, or *traces*.

The traces corresponding to a normal scenario were gathered by allowing the APUs to elect the GM via the BMCA at any time without changing any of the APUs' specifications. For what concerns the attack scenarios, the default configuration of one APU was altered to force the other APUs to elect it as GM. Specifically, $priority1$ and $priority2$ were lowered in order to guarantee higher priority to the malicious APU. This APU was then used to carry out the attack in all tests.

Following a preliminary analysis, we found that training the classifiers on the samples related to seq +1 ns and seq +10 ns attacks significantly reduces the recall of the models. This is specially true for the model trained on samples with only offset features. In fact, the behavior of the attacked clock appears to change little from its usual activity. For this reason, we decided to discard these traces and re-train the models with samples related to attacks adding 100 or more ns. We defer the analysis of attacks that introduce a timestamp offset of less than 100 ns to future research.

D. Model

To test our tool, we employ a Random Forest (RF) model. The choice of RF classifier was motivated by its robustness to outliers and good prediction performance achievable with less tuning and lower computational resources compared to other classifiers. These characteristics makes RF a good candidate for integration in ECUs. As described in Section IV-C, we opted to test our pipeline using two sets of models for each node:

- 2 models trained on all features, one for Phase 1 and one for Phase 2;
- 2 models trained excluding frequency-related features, one for Phase 1 and one for Phase 2;

For each of these models, we performed a Grid Search tuning of the hyperparameters.

Table II
PRECISION AND RECALL ACHIEVED BY ALL THE CONSIDERED MODELS.

	All features		**Without frequency**	
	Phase 1	**Phase 2**	**Phase 1**	**Phase 2**
Precision	98.1%	99.8%	99.5%	99.3%
Recall	96.4%	99.9%	94.5%	44.8%

E. Results

We define all samples belonging to attack traces as positives, and all samples belonging to normal traces as negatives. In this work, we consider two metrics, the recall and the precision (described in Section IV-B). Table II illustrates the average recall and precision obtained on all ES when classifying each sample independently, i.e. one sample per trace for Phase 1, and N samples per trace for Phase 2, where N is the number of windows extracted. The table also serves as a comparison between the performance of the models trained on all features and the models trained without considering frequency-related features.

Both models were able to attain high levels of precision and recall throughout Phase 1, which demonstrates that the samples are consistently identified correctly. According to the table, the performance of the model trained on all features is even higher for Phase 2. By contrast, when features related to the frequency are not considered, we can observe a clear degradation in the recall. This difference in performance can be explained by two factors:

- as mentioned in Section IV-C, in spite of the attack, the slave is able to retain its usual behavior most of the time with respect to the offset (except for some sudden wide oscillations);
- by contrast, the relative frequency is highly representative of the behavior of slave clocks when a new GM is elected.

Nonetheless, despite a low recall, the model achieves a high True Negative Rate (TNR) (the proportion of normal samples recognized as negatives out of the total normal samples) of 98.6%. This means that the confidence of the classifier when predicting the normal traces is high.

As anticipated in Section IV-B, the IDS does not issue an alert after a single positive sample, but it stores the sample in a database. Then, when an *alert threshold* (which is based on a predefined heuristic) is overcome, a warning is triggered. In this work, the alert threshold is determined by the percentage of positives samples out of the total classified samples in a trace. As an example, if the threshold is set to 20%, if 20% or more of the samples in the trace are labeled as positives, an alert is triggered, otherwise not.

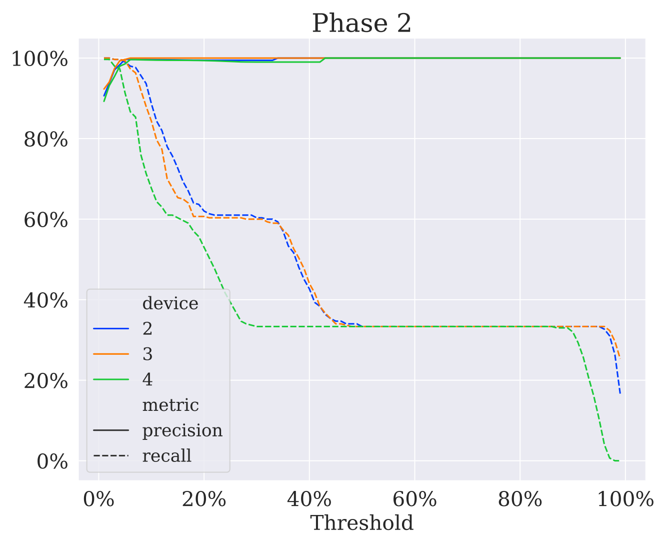

Figure 7. Detection precision and recall according to different alert thresholds.

Figure 7 illustrates how precision and recall vary according to the threshold in the target devices. As expected, the higher the threshold, the higher is the precision and the lower is the recall. In particular, precision and recall of more than 90% are achieved when the threshold is between 5% and 10%.

It is to be noted that this approach is suitable only for an offline IDS, i.e. an IDS where the trace is processed after the attack event. Therefore, it is more applicable for forensic investigation, rather than timing detection and reaction. Nevertheless, we believe that the presented results provide valuable insights for the definition of a threshold heuristic for online IDS.

VI. Conclusion

In this work, we investigated the potential of IDS on gPTP, the time synchronization protocol candidate for automotive TSN. We implemented a complete prototype on a real gPTP-enabled testbed, under the assumption that each node can host an IDS or can send data to a central IDS module within the vehicle. The threat model is the one of an adversary who can obtain the role of GM and send $Sync$ packets with a timestamp which is gradually drifting from the original timestamp.

Initially, we studied the gPTP traffic and the behavior of the clocks under attack with the aim of identifying sources for the extraction of features. Our research reveals that the frequency and offset of the clocks are useful indicators of attacks. Based on this information, we built a two-phase intrusion detection pipeline.

The results show that, when the offset added to each $Sync$ packet by the adversary is $\geq$ 100 ns, an IDS can detect the attacks with precision and recall up to 99%.

Future work includes evaluating the performance of IDSs against a wider pool of attacks, e.g. spoofing, DoS etc. on gPTP. In this regard, the characteristics of the gPTP traffic should be analyzed more in depth to identify new ways of fingerprinting the normal behavior of the network with high accuracy.

Finally, as described in Section I, the rogue master attack cannot be executed if the GM is static. Given the importance of having a dynamic GM to ensure system availability, we urge that the scientific community and manufacturers investigate changes to the BMCA in order to minimize the possibilities for an attacker to acquire an GM position in an automotive scenario.

Acknowledgment

This work was supported by the BRIDGES grant, funded by the Luxembourg National Research Fund (FNR), and by Honda R&D Europe (Germany) GmbH.

References

[1] C. Miller and C. Valasek, “Adventures in automotive networks and control units,” *Def Con*, vol. 21, no. 260-264, pp. 15–31, 2013.

[2] S. Jafarnejad, L. Codeca, W. Bronzi, R. Frank, and T. Engel, “A car hacking experiment: When connectivity meets vulnerability,” in *2015 IEEE Globecom Workshops (GC Wkshps)*, IEEE, 2015, pp. 1–6.

[3] C. Miller and C. Valasek, “Remote exploitation of an unaltered passenger vehicle,” *Black Hat USA*, vol. 2015, no. S 91, 2015.

[4] M. Bertoncello, G. Camplone, P. Gao, et al., “Monetizing car data—new service business opportunities to create new customer benefits,” *McKinsey & Company*, 2016.

[5] IEEE SA, “IEEE Standard for Local and Metropolitan Area Networks–Timing and Synchronization for Time-Sensitive Applications,” Standard, 2019.

[6] IEEE Std 1588™-2019, “IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems,” IEEE SA, Standard, 2019.

[7] M. Fotouhi, A. Buscemi, A. Boualouache, F. Jomrich, C. Koebel, and T. Engel, “Assessing the Impact of Attacks on an Automotive Ethernet Time Synchronization Testbed,” *2023 IEEE Vehicular Networking Conference (VNC)*, 2023.

[8] C. Young, J. Zambreno, H. Olufowobi, and G. Bloom, “Survey of automotive controller area network intrusion detection systems,” *IEEE Design & Test*, vol. 36, no. 6, pp. 48–55, 2019.

[9] M. Müter, A. Groll, and F. C. Freiling, “A structured approach to anomaly detection for in-vehicle networks,” in *2010 Sixth International Conference on Information Assurance and Security*, IEEE, 2010, pp. 92–98.

[10] A. Taylor, N. Japkowicz, and S. Leblanc, “Frequency-based anomaly detection for the automotive CAN bus,” in *2015 World Congress on Industrial Control Systems Security (WCICSS)*, IEEE, 2015, pp. 45–49.

[11] Y. Hamada, M. Inoue, H. Ueda, Y. Miyashita, and Y. Hata, “Anomaly-based intrusion detection using the density estimation of reception cycle periods for in-vehicle networks,” *SAE International Journal of Transportation Cybersecurity and Privacy*, vol. 1, no. 11-01-01-0003, pp. 39–56, 2018.

[12] H. Lee, S. H. Jeong, and H. K. Kim, “OTIDS: A novel intrusion detection system for in-vehicle network by using remote frame,” in *2017 15th Annual Conference on Privacy, Security and Trust (PST)*, IEEE, 2017, pp. 57–5709.

[13] linuxptp, *ptp4l*. [Online]. Available: http://linuxptp.sourceforge.net/.

[14] G. Kumar, “Evaluation metrics for intrusion detection systems-a study,” *Evaluation*, vol. 2, no. 11, pp. 11–7, 2014.

[15] K.-T. Cho and K. G. Shin, “Fingerprinting Electronic Control Units for Vehicle Intrusion Detection,” in *25th USENIX Security Symposium (USENIX Security 16)*, Aug. 2016, pp. 911–927.

[16] Netgear. “A New Generation of Gigabit Smart Switches.” (2019), [Online]. Available: https://www.downloads.netgear.com/files/GDC/datasheet/en/GS716Tv3-GS724Tv4-GS748Tv5.pdf?_ga=2.213794839.222651013.1652102686-21276228.1652102686.

[17] P. Engine, *apu2e4*, May 2016. [Online]. Available: https://pcengines.ch/apu2e4.htm.