Behavior Analysis for Safety and Security in Automotive Systems

Roland Rieke*, Marc Seidemann†, Elise Kengni Talla†, Daniel Zelle*, Bernhard Seeger†
*Fraunhofer SIT, Darmstadt, Germany
Email:{roland.rieke,daniel.zelle}@sit.fraunhofer.de
†Philipps-Universität Marburg, Marburg, Germany
Email:{seidemann,elise,seeger}@mathematik.uni-marburg.de

***Abstract*—The connection of automotive systems with other systems such as road-side units, other vehicles, and various servers in the Internet opens up new ways for attackers to remotely access safety relevant subsystems within connected cars. The security of connected cars and the whole vehicular ecosystem is thus of utmost importance for consumer trust and acceptance of this emerging technology. This paper describes an approach for on-board detection of unanticipated sequences of events in order to identify suspicious activities. The results show that this approach is fast enough for in-vehicle application at runtime. Several behavior models and synchronization strategies are analyzed in order to narrow down suspicious sequences of events to be sent in a privacy respecting way to a global security operations center for further in-depth analysis.**

***Keywords*-automotive security; connected car; predictive security analysis; security modeling and simulation; security monitoring; complex event processing; process discovery.**

I. Introduction

Connected cars, road-side units, and external services establish a new ecosystem with important advantages - such as situational awareness - that enable vehicles to act autonomously and intelligently. However, when formerly closed automotive systems now evolve into open systems, then external security threats indirectly impact the safety mechanisms that probably have been developed with a closed-world assumption for the vehicle in mind [1]. Security problems in modern cars have been revealed in [2] and, most recently, in [3], where it has been shown that a large number of modern vehicles could be attacked by remotely injecting messages to in-vehicle networks such as the Controller Area Network (CAN) bus. Safety critical Electronic Control Units (ECUs) that are connected to this bus finally influence physical actions such as steering and braking. It is thus very important to improve security of in-vehicle networks and, as long as there are no effective means to prevent specific attacks, there should be methods in place to automatically detect them and warn the driver [4].

The aim of this work is, to complement specification-based on-board attack detection [5] by measures to detect *unanticipated sequences of events* in order to identify subtle signs of suspicious activities that could indicate new *unknown* attacks. In our approach, we assume that in the future there will be a multi-level hierarchy of systems for detection and prevention of malicious activities in place. In the vehicle, signature-based measures to detect and combat *known* attacks will act autonomously within the on-board network. However, the detection of new *unknown* attacks in general requires (a) more resources than available in one vehicle and (b) more knowledge than available locally. As a consequence, we expect that - in addition to the local detectors - there will be advanced back-end systems that will collect information from fleets of vehicles and analyze these based on the global view, such as the cloud-assisted defense framework described in [6]. This centralized analysis will allow for generation of new signatures that in turn will be deployed to the in-vehicle detectors. This paper contributes to such an approach by providing means for local in-vehicle processing of event streams in order to identify *anomalous* behavior. The *normal* behavior will be filtered out and only the *unanticipated* behavior will be sent to the global operations center for further analysis. It is important to note that for privacy reasons only data strictly needed for security guarantee must be send to remote analysis, while unnecessary details must be deleted or made anonymous [7]. Our goal is to identify a very small partition ($< 1\%$) of unanticipated behavior based on models of normal behavior generated in a clean environment.

The main contributions of this work are (a) the design and implementation of a model-based method to compare the measured behavior of a vehicle with the expected behavior, (b) the experimental determination of model complexity that is necessary to find security anomalies and fast enough in practice, and (c) the analysis of the effects of several possible strategies to re-synchronize the model with the event stream when events don't fit to the expected behavior.

This article is structured as follows: Section II discusses related work, while Section III introduces the application scenario and the preparation of test data. Section IV describes the discovery of behavior models and their processing during runtime. Section V presents the results of various experimental setups, and Section VI concludes this paper.

II. Background and Related Work

For Intrusion Detection System (IDS) in automotive context, several publications propose anomaly detection based on clearly defined and specified normal on-board system behavior [8] and suggest specification-based attack detection [5], assuming that a representation of the normal behavior of communication and ECUs can be derived from the system policy and the expected usage of a component, which is then compared to the observed behavior. In [9] a set of in-vehicle detection sensors is described from an abstract point of view. Details on implementation and verification are not given in the above publications. On the commercial side, Towersec provides a product called ECUSHIELD [10] that runs on a CAN bus accessible ECU, telematics controller or infotainment unit, and continuously monitors the events in order to identify new threats. According to [11], Argus also provides an IDS that identifies attacks in the in-vehicle network and blocks them in real time. To the best knowledge of the authors of this paper, details about the approaches used in the Towersec and Argus products are not publicly available.

In general, IDS [12] systems usually do not have knowledge of the vehicle applications, thus lacking the connection between the reported security problems and the affected critical behavior. IDS systems can be based on several different technologies, such as Complex Event Processing (CEP) [13], [14] or Deep Learning (DL) [15], [16]. For example, in [17] an IDS applying DL is described. However, the system is only verified on synthetic data. Such systems often deliver a very low false positive rate, but they usually identify outliers based on inspection of single events. Thus, they might miss attacks based on valid events sent in the wrong context. Predictive security analysis for event-driven processes has been introduced in [18], [19], [20]. Here, we use a similar approach but a different realization adapted for the requirements of the automotive domain. We combined Process Mining (PM) [21], [22] with CEP technologies in order to allow for analysis of anomalous behavior within on-board systems in connected cars that is characterized by anomalies in sequences of events and not detectable in single events on a CAN bus of a modern vehicle.

III. Application Scenario and Requirements

Today's upper-class cars contain more than 70 ECUs. These are connected in different bus systems which are interconnected via a central gateway and have different tasks. The central gateway is also connected to an On-Board Diagnose (OBD) port. This port is used by repair shops to find problems and to perform exhaust measurements.

To detect attacks inside an automotive network we connected a CAN bus adapter to the OBD port of a Renault Zoe (2016) and stored every transmitted message. Every CAN message is stored with a time stamp in seconds, the bus the message came from, the message ID that indicates the purpose of the message, a indicator Rx or Tx for Receive exchange or Transmission exchange, the length of the message and the payload. This recording procedure was done twice. Once during a drive for about 10 minutes in an urban environment and once in a standing car with ignition turned on.

To simulate an attacker, we synthesized known attacks from [23] and introduced messages with wrong steering angles, brake positions and speed information and injected these into the message stream on the bus. Attacks like these lead to physical impacts and may harm peoples lives. Table I shows an extract of the recorded messages (marked by Rx) and a synthesized attack (marked by Tx). In a similar way, attacks like the one described in [9], namely, a sequence of events which is shifting the vehicle speed from a very low value (e.g. 20 km/h) to a very high value (e.g. 200 km/h) and backward without expected intermediate values, could be synthesized and injected. Never the less the attack sends only valid messages to the bus stream. The messages can be the equivalent of an emergency brake or a fast acceleration.

TABLE I: Excerpt from extracted CAN bus trace

time	proc	ECU	Rx/Tx	len	payload
…	…	…	…	…	…
264.452194	2	32C	Rx	2	3E 80
264.452353	1	1F6	Rx	8	DE 20 08 2F 00 FF FF FF
264.452599	1	1F8	Rx	8	4F 04 80 1F FE 00 02 0D
264.452841	1	432	Rx	8	50 00 02 0B AC 80 00 58
264.453075	1	130	Rx	8	00 E8 6F FE 62 AB DC C1
264.453792	1	212	Rx	6	FE 1D C0 6C FF FF
264.455620	1	C6	Rx	8	93 67 7F FF 80 08 B6 49
264.457624	1	17A	Rx	8	FF FF FF AA 00 F0 31 A3
264.457784	1	439	Rx	3	3E 80 F0
264.458022	1	C6	**Tx**	8	6C E3 7F FF 80 08 82 28
264.458030	1	17E	Rx	8	FF FF FF 00 FF 40 00 FF
264.458254	1	186	Rx	7	19 50 32 03 20 00 20
264.458762	1	12E	Rx	8	C9 80 00 7F E0 FF FF 00
…	…	…	…	…	…

IV. Implementation of Anomaly Detection System

The detection process of anomalous behavior is basically divided into two stages. In the first stage, denoted as *discovery*, a model representing the normal behavior of the car is derived. In the second stage, denoted as *conformance checking*, the model is utilized to determine deviation from the model, i.e., anomalies. The basis of the normal behavior model are traces recorded from the CAN bus of the car. Because we assume that the behavior of the car does not change often, the discovery stage is an offline operation. As representation model we chose Petri nets, which were mentioned first in [24] and are very common in process mining ([22], [25]). We utilize the *Alpha* algorithm [26] to derive a Petri net from a given trace, presented in Algorithm 1. For our implementation we used the open source libraries available at [27].

Algorithm 1: Petri net construction with [27]

```
Input  : String inFile
Output: Pair<Petri net,Marking>
XLog l;
XEventClassifier c;
AlphaMinerParameters p;
XAttributeMap m ← new XAttributeMapImpl();
l ← CreateXLogFromCSV.readInputFromCSV(inFile,
  attributeMap);
c ← new XEventAttributeClassifier(”ZoeEventClassifier”,
  ”MessageID”); p ← new
  AlphaMinerParameters(AlphaVersion.CLASSIC);
return AlphaClassicMinerImpl.run(null, log, classifier,
  parameter);
```

The anomaly detection is implemented in our CEP middleware framework [28]. In this framework, several Event Processing Agents (EPAs) are connected to an Event Processing Network (EPN). We implemented a specialized Petri net EPA (PEPA) which maintains the current state of a Petri net representing the behavior model. This PEPA consumes events from the trace as input and outputs only those events that are classified as anomalies. As the discovery stage is an offline operation, the PEPA is initialized with the Petri net representing the model.

Algorithm 2 describes the anomaly detection routine of a PEPA. An incoming event is classified as conform with the behavior, iff there is a valid transition from an active place of the Petri net. Otherwise, the event is classified as anomaly. The PEPA checks the routine in Algorithm 2 for each incoming event and outputs only those events which are classified as anomaly, i.e., do not match the expected behavior.

Algorithm 2: Processing of an event

```
Input  : Petri net p, XEvent e
Output: False in case of an anomaly, true otherwise
Transition t ⟵ null;
String id ⟵ e.getAttributes().get("id ").toString();
foreach Transition tmp in p.getTransitions() do
    if tmp.getLabel() = id then
        t ← tmp;
        break;
    end
end
if t = NULL then
    return false ;        // No transition at all
end
try
    Marking o ← new Marking();
    o.addAll((semantic.getCurrentState());
    PetrinetExecutionInformation result ←
      (PetrinetExecutionInformation)
      semantic.executeExecutableTransition(t);
    marking ← result.getTokensProduced();
    return true;
catch IllegalArgumentException
    return false ;        // No active transition
end
```

We differentiate two different types of anomalous events, *unknown* and *unanticipated* events. While the former type of event has never occurred before, the latter only does not fit in the current context of the behavior model. In case of an unanticipated event, we implemented three different strategies to calibrate the Petri net for further processing: (a) simply ignores the unanticipated event and remains the current state of the Petri net; (b) resets the Petri net to its initial state; (c) creates a copy of the PEPA and applies (a) and (b) in parallel.

V. Analysis of the Effectiveness of the Approach

As a basis for the experiments reported in this section, we used a logfile containing 1.014.070 events that was recorded while driving the car (cf. Sect. III). The logfile covers 537 seconds of realtime traffic on the CAN bus, and thus, the average event rate is 1.888 events/second. From this fact it follows that the processing of the events needs to be faster than this rate because otherwise it would be necessary to drop events. The majority of the measurements in this section have been produced on a personal computer with Intel Core i5 CPU and 8GB memory. Whenever computations have been run on another machine, the times have been adjusted according to the speed difference. We plan to make this logfile publicly available, in order to enable reproduction of the results and comparison with other approaches in the field.

A. Construction of the behavior model

The first task was the construction of a suitable behavior model. The aim was the experimental determination of model complexity that is sufficiently complete to identify behavior anomalies and small enough to allow realtime processing in practice (at the speed of the CAN bus). For our experiments we used the *Alpha* algorithm to compute several models based on different excerpts of the available logfile (see Table II). A stream of events characterizes one specific execution trace of the observed automotive system. In order to avoid state space explosion problems, it is important to define a mapping of the runtime events to the abstract actions considered by the model discovery algorithm. The coarsest abstraction that still contains all security relevant information should be used [19], [20]. For the following experiments, we projected each CAN message to its message ID that indicates the purpose of the message. It is important to note that this mapping has to be optimized with respect to specific attacks. For example, when attacks are characterized by wrong steering angles or sudden changes in speed, then these parameters need to be part of the model and thus should be included in the event mapping. The number of events taken into account varied between 500 and 4.000. Larger models would slow down the realtime processing below the limit of 1.888 events/second.

From Table II it can be concluded that the time needed to compute such a model is growing faster than linear from 94 events/second in M1000, 68 events/second in M2000" and 53 events/second in M4000 to only 2,4 events/second in M4000'. An aside: We found that the *Alpha* algorithm is very slow when the number of events exceeds 14.000. Fortunately, the time to compute such a model is not so critical because we assume that the behavior of a car will not change very often (basically in case of software updates). Therefore, we assume that the model can be computed offline.

B. Event stream processing

The next interesting question was to analyze execution times of behavior checking when comparing the expected behavior given by the model with the measured behavior given by an event stream. At runtime the event stream has to be mapped

TABLE II: Construction time and size of models

Petri net model discovery						
Model	*Start*	*Events*	*Time*	*Tran.*	*Places*	*Edges*
M500	0	500	1.176	83	112	455
M1000	200.000	1.000	10.611	97	232	1.423
M1000'	490.000	1.000	1.717	95	170	879
M1000"	700.000	1.000	2.647	95	170	849
M2000	200.000	2.000	11.333	104	317	2.056
M2000'	490.000	2.000	3.233	102	271	1.337
M2000"	700.000	2.000	29.213	101	313	1.819
M3000	200.000	3.000	65.473	104	566	4.603
M3000'	490.000	3.000	235.250	104	623	4.899
M4000	200.000	4.000	75.018	105	537	3.816
M4000'	490.000	4.000	1.671.779	105	900	7.994

Model is the identifier used for further reference; *Start* denotes the position in the logfile where the first event for the model is taken; *Events* denotes the number of consecutive events used for the model discovery; *Time* is the maximum time in milliseconds for the generation of the model, where the maximum is taken based on two runs with very small deviations; *Trans.*, *Places*, *Edges* denote the number of transitions, places and edges of the generated Petri net.

to an abstract stream by the same projection used in the model discovery phase. In order to improve performance, this filtering and aggregation step could be implemented by specialized pre-processing components [29], [20]. The experiments revealed that out of the three implemented strategies to calibrate the Petri net, strategy (b) did not detect anomalies successfully in the given event streams, and for strategy (c) the throughput was very slow except for the smallest M500 model. This is because each fork had to compute the followup states in the Petri nets independently. In the following, we therefore report only the results of strategy (a).

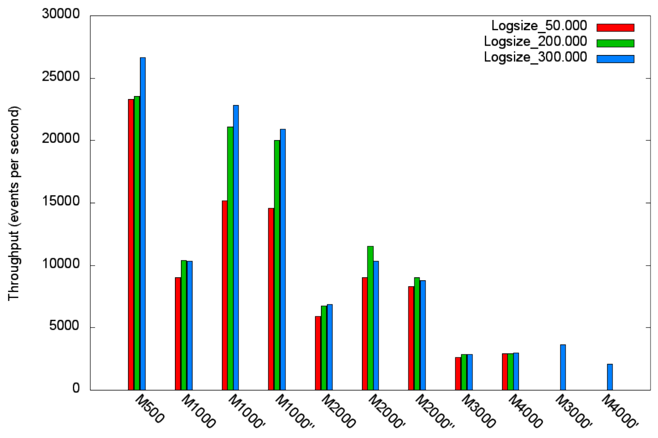

Fig. 1: Throughput when unanticipated events are ignored

As shown in Fig. 1, the throughput is dependent on the complexity of the model as well as on the anomaly management strategy. However, for the complex models M3000, M3000' and M4000 analyzed here, the throughput is above 2.650 events/second and even for the most complex model M4000' the throughput is slightly above 2.000 events/second. This means that the data which are generated by the given car at a rate of 1.888 events/second could be processed in real time. We also checked the maximum storage used when processing the different models and found that the allocated storage never exceeded 800 MB. Thus, this resource is not a problem for the processing of the given event streams.

Summarizing these experiments, we conclude that our proposed approach and implementation are able to execute the behavior check at execution time fast enough with the given computing platform. We have also done the same experiments with models computed by the *Alpha*+ algorithm [26] but we did not find significant differences.

C. Model quality

For the experimental determination of model complexity that is most suitable to find security anomalies, the first goal is to reduce the number of *unknown* events that are not in the model because they didn't appear in the part of the logfile that was used to generate the Petri net. Column *Unknown* in Table III shows the results for the different models that we used. We conclude that for our setup the models M4000 and M4000' which have been generated on the basis of 4.000 events from different ranges within the logfile provide the best coverage although there are still 280 events that appear in the test data consisting of 1.000.000 events recorded while driving the car but not in the models.

In order to identify *normal* behavior based on the Petri net models, we had to decide what to do when an event is identified that does not fit to the expected behavior. As stated above, in strategy (a) we ignored these *unanticipated* events that means we didn't change the state of the model and proceeded with checking the next event. Table III shows the results computed for the different models.

TABLE III: Anomaly rate for strategy *Ignore*

	Model	*Unknown*	*Unanticipated*	*Ignored (sum)*
Test data: 1.000.000 events	M500	96.938	5.210	102.148 (10,21%)
	M1000	3.102	9.066	12.168 (1,22%)
	M1000'	4.158	880	5.038 (0,50%)
	M1000"	4.170	645	4.815 (0,48%)
	M2000	457	9.045	9.502 (0,95%)
	M2000'	809	490	1.299 (0,13%)
	M2000"	985	1.130	2.115 (0,21%)
	M3000	457	8.844	9.301 (0,93%)
	M3000'	457	451	908 (0,09%)
	M4000	280	8.623	8.903 (0,89%)
	M4000'	280	455	735 (0,07%)

The results of the experiments with this synchronization strategy demonstrate that our approach delivers very low anomaly rates when the model is carefully adjusted. It is an important result that the identification of the parts of the log that contain a good coverage of the system behavior is essential for the quality of the model, e.g. the model M1000" with only 849 edges is of better quality than the much more complex model M4000 with 3.816 edges.

VI. Conclusion and Future Work

The aim of this paper was to provide an approach and an implementation for in-vehicle processing of event streams in order to identify anomalous behavior with respect to sequences of events and not only single events. The normal behavior should be filtered out thus leaving only a small percentage of abnormal behavior that might be caused by malicious agents and might lead to safety critical actions in the actuators of the car. Thus, these unanticipated events need to be further processed for further security analysis, either by other in-vehicle components, such as those proposed in [9], or by a global operations center.

In order to satisfy this objective, this work contributes the design and implementation of a model-based method to compare the measured behavior of a vehicle with the expected behavior. The experiments have shown that the proposed approach is appropriate to compute the models we considered and to execute the behavior check on the given computing platform fast enough to follow the event stream at execution time without the need to drop events. Our overall goal has been to identify a very high partition of normal behavior based on models generated in a clean environment. We have shown that the anomaly rate for strategy *Ignore* has been below 1% for models with sufficient coverage and 0.07% for the best model.

Future work on model discovery should provide algorithms that adapt the model during execution time on-the-fly whenever false positives are found. Future work on the improvement of the detection of attacks needs to explore an analysis of the payload of the CAN bus events because some attacks can only be detected when payload is taken into account in the model. Future work on performance issues should aim to transfer the software to an embedded platform that is likely to be used in modern cars. Finally, there are many interesting issues when designing a global system that gets event streams from many cars and that identifies attacks which are only detectable with a global view.

Acknowledgment

Roland Rieke and Daniel Zelle acknowledge the financial support by the German Federal Ministry of Education and Research in the context of the projects secUnity (ID 16KIS0398) and SeDaFa (ID 16KIS0432K).

References

[1] K. Schmidt, P. Tröger, H.-M. Kroll, T. Bünger, F. Krueger, and C. Neuhaus, "Adapted development process for security in networked automotive systems," *SAE Int. J. Passeng. Cars – Electron. Electr. Syst.*, vol. 7, pp. 516–526, 04 2014.

[2] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham *et al.*, "Experimental security analysis of a modern automobile," in *2010 IEEE Symposium on Security and Privacy*. IEEE, 2010, pp. 447–462.

[3] C. Miller and C. Valasek, "Remote exploitation of an unaltered passenger vehicle," IOActive Labs, August 2015, [Online; accessed 09-Sep-2016].

[4] T. Hoppe, S. Kiltz, and J. Dittmann, "Security threats to automotive can networks – practical examples and selected short-term countermeasures," *Reliability Engineering & System Safety*, vol. 96, no. 1, pp. 11–25, 2011.

[5] U. E. Larson, D. K. Nilsson, and E. Jonsson, "An approach to specification-based attack detection for in-vehicle networks," in *Intelligent Vehicles Symposium, 2008 IEEE*, June 2008, pp. 220–225.

[6] T. Zhang, H. Antunes, and S. Aggarwal, "Defending Connected Vehicles Against Malware: Challenges and a Solution Framework," *Internet of Things Journal, IEEE*, vol. 1, no. 1, pp. 10–21, Feb. 2014.

[7] R. Rieke, L. Coppolino, A. Hutchison, E. Prieto, and C. Gaber, "Security and reliability requirements for advanced security event management," in *Computer Network Security*, ser. LNCS, I. Kotenko and V. Skormin, Eds. Springer, 2012, vol. 7531, pp. 171–180.

[8] F. Kargl, P. Papadimitratos, L. Buttyan, M. Müter, B. Wiedersheim, E. Schoch, T.-V. Tongh, G. Calandriello, A. Held, A. Kung, and J.-P. Hubaux, "Secure Vehicular Communications: Implementation, Performance, and Research Challenges," *IEEE Communications Magazine*, vol. 46, no. 11, p. 2–8, 11/2008 2008.

[9] M. Müter, A. Groll, and F. C. Freiling, "A structured approach to anomaly detection for in-vehicle networks," in *Sixth International Conference on Information Assurance and Security, IAS 2010, Atlanta, GA, USA, August 23-25, 2010*. IEEE, 2010, pp. 92–98.

[10] Towersec, "Towersec ECUSHIELD," http://tower-sec.com/ecushield/, 2016, [Online; accessed 09-Sep-2016].

[11] Argus, "Argus Intrusion Detection and Prevention System," https://argus-sec.com/argus-invehicle-network-protection/, 2016, [Online; accessed 12-Sep-2016].

[12] H.-J. Liao, C.-H. R. Lin, Y.-C. Lin, and K.-Y. Tung, "Intrusion detection system: A comprehensive review," *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 16 – 24, 2013.

[13] D. Luckham, *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley, 2002.

[14] F. Terroso-Saenz, M. Valdes-Vela, and A. F. Skarmeta-Gomez, "A complex event processing approach to detect abnormal behaviours in the marine environment," *Information Systems Frontiers*, vol. 18, pp. 765–780, 2016.

[15] R. Socher, M. Ganjoo, H. Sridhar, O. Bastani, C. D. Manning, and A. Y. Ng, "Zero-shot learning through cross-modal transfer," *CoRR*, vol. abs/1301.3666, 2013.

[16] Z. Wang, "The applications of deep learning on traffic identification," in *BlackHat USA*, 2015.

[17] M.-J. Kang and J.-W. Kang, "Intrusion detection system using deep neural network for in-vehicle network security," *PloS one*, vol. 11, no. 6, p. e0155781, 2016.

[18] R. Rieke and Z. Stoynova, "Predictive security analysis for event-driven processes," in *Computer Network Security*, ser. LNCS, I. Kotenko and V. Skormin, Eds. Springer Berlin / Heidelberg, 2010, vol. 6258, pp. 321–328.

[19] R. Rieke, J. Repp, M. Zhdanova, and J. Eichler, "Monitoring security compliance of critical processes," in *Proc. of the 22nd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP'14), Turin, Italy*. IEEE, February 2014, pp. 525–560.

[20] R. Rieke, M. Zhdanova, and J. Repp, "Security compliance tracking of processes in networked cooperating systems," *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA)*, vol. 6, no. 2, pp. 21–40, June 2015.

[21] F. M. Maggi, M. Montali, M. Westergaard, and W. M. P. van der Aalst, "Monitoring business constraints with linear temporal logic: An approach based on colored automata," in *Proc. of the 9th International Conference on Business Process Management (BPM'11)*, ser. LNCS. Springer, August–September 2011, vol. 6896, pp. 132–147.

[22] A. Rozinat and W. van der Aalst, "Conformance checking of processes based on monitoring real behavior," *Information Systems*, vol. 33, no. 1, pp. 64–95, March 2008.

[23] C. Valasek and C. Miller, "Adventures in automotive networks and control units," IOActive Labs, White Paper, 2014, [Online; accessed 09-Sep-2016].

[24] C. A. Petri, "Kommunikation mit Automaten," TH Darmstadt, Dissertation, 1962.

[25] W. M. P. van der Aalst, *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011.

[26] W. van der Aalst, T. Weijters, and L. Maruster, "Workflow mining: Discovering process models from event logs," *IEEE Trans. on Knowl. and Data Eng.*, vol. 16, no. 9, pp. 1128–1142, Sep. 2004.

[27] W. van der Aalst *et al.*, "ProM Tools," https://svn.win.tue.nl/trac/prom/.

[28] B. Hoßbach, N. Glombiewski, A. Morgen, F. Ritter, and B. Seeger, "JEPC: the java event processing connectivity," *Datenbank-Spektrum*, vol. 13, no. 3, pp. 167–178, 2013.

[29] P. Verissimo *et al.*, "MASSIF architecture document," FP7-257475 MASSIF European project, Tech. Rep., April 2012.