Identify Potential Attacks from Simulated Log Analysis

Cinzia Bernardeschi*, Andrea Domenici*, Francesco Mercaldo† ‡, Antonella Santone§
*Department of Information Engineering, University of Pisa, Pisa, Italy
{cinzia.bernardeschi,andrea.domenici}@unipi.it
†Department of Medicine and Health Sciences "Vincenzo Tiberio", University of Molise, Campobasso, Italy
francesco.mercaldo@unimol.it
‡Institute for Informatics and Telematics, National Research Council of Italy (CNR), Pisa, Italy
francesco.mercaldo@iit.cnr.it
§Department of Biosciences and Territory, University of Molise, Pesche (IS), Italy
antonella.santone@unimol.it

***Abstract*—Modern vehicles contain an ecosystem of several electronic units able to exchange data using the serial communication provided by the CAN bus. This protocol can be afflicted by a plethora of attacks that can expose the driver and the passengers to risks for their safety. In this paper we propose a method to detect potential attacks in automotive networks. We start from the analysis of a log obtained from a simulation and we consider a formal verification environment to verify whether the formal model we built from the log is safe. As a proof of concept, we evaluate the proposed method in a case study related to adaptive cruise control, to preliminarily demonstrate its effectiveness.**

I. Introduction

Modern vehicles exhibit a large number of networked components that as a whole are responsible for monitoring and controlling the state of the car [1]. Each component is responsible for communicating with neighboring components: this traffic generates a large amount of data [2]. As a matter of fact, automobiles contain upwards of fifty electronic control units (ECUs) networked together [3]. For this reason, the overall safety of the car relies on near real-time communication between the ECUs [4]. While ECUs are communicating, they must detect skids, predict crashes, perform anti-lock braking and a lot of other safety-critical tasks. [5], [6].

Usually, ECUs are networked using one or more buses complying with the Controller Area Network (CAN) standard. CAN[1] is a high-integrity serial data communication technology developed in the early 1980s by Robert Bosch GmbH with the aim to provide efficient communication between several automotive applications [7]. The ECUs communicate with each other by sending CAN packets. These packets are broadcast to all components on the bus and each component checks if a packet is intended for it, although segmented CAN networks exist [8], [9]. In CAN packets there is no built-in source identifier or authentication. These are the reasons why it is easy for malicious components [10] to both sniff the CAN network as well as masquerade as other ECUs and send CAN packets. The lack of a source identifier also makes it difficult to monitor traffic because it is impossible, a priori, to know which ECU is sending or receiving a particular packet.

The safety of drivers and passengers relies critically on the code running in their automobiles and the threat to their physical well-being is real [11], [12], [3].

In this paper, a method is exposed to identify conditions suggesting the possibility of an impending attack, based on the analysis of logs from simulated attacks. Once these conditions have been identified and characterized, it would be straightforward to implement algorithms that recognize them at run-time, enhancing system security and safety. The method relies on a tool that, given a simulation log, creates an abstraction of that log in the form of a labeled transition system, and a model-checking prover for temporal logic is used to check if a given condition on the system state (e.g., distance and relative speed of vehicles) is compatible with an attack.

Any kind of simulation environment can be used with this method. In automotive applications, systems are usually modeled and simulated with a block-based graphical tool, such as Simulink [13]. Such tools model a system with a set of interconnected functional blocks realizing mathematical or logical operations on system quantities, such as sampling, amplification or attenuation, or integration. Discrete components are modeled as automata. In this work, systems are assumed to be modeled in Simulink.

As a proof of concept, the method has been applied to a case study related to adaptive cruise control (ACC), a widely used system that automatically adjusts vehicle speed to maintain a safe distance from vehicles ahead. Such systems may use a radar or laser sensor or a camera setup to measure the distance and relative speed of the vehicle ahead, allowing the controller to decelerate or accelerate accordingly. ACC is already available in many cars and will be a key component in the future driverless cars.

II. Related work

In last years, several researches investigated about the safety and security in automotive networks, for instance Checkoway et al. [14] show that remote exploitation is feasible through

[1]www.can.bosch.com

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

a broad range of attack vectors (i.e., mechanical tools, CD players, Bluetooth and cellular radio).

Researchers in [15] demonstrate that a long-range wireless attack is possible using a real-world car and dangerous smartphone applications in a connected car environment.

Authors in [16] design an anomaly detector considering a Long ShortTerm Memory neural network in order to detect CAN bus attacks. Their proposal works by learning to predict the next data word originating from each bus sender. Their detector is able to recognize the synthesized anomalies with low false-positive rates.

Alheeti et al. [17] propose an intelligent IDS-based system to secure external communication for self-driving and semi-self-driving vehicles. They design a VANET environment exploiting the NS2 simulator, in order to generate two behaviors: normal and malicious.

Liu et al. [18] propose a system to secure the communications system for vehicles depending on roadside units. The proposed infrastructure contains a Certification Authority (CA) based cluster distributed in different regions. The aim is to show that IDS using a CA database provide more protection against malicious vehicles with legal certificates.

Lyamin et al. [19] propose an algorithm to detect denial of service attacks in real-time, based on performance metrics such as the percentage of false alarms for any jamming channel and the average beacon time for vehicular networks.

Golle et al. [20] discuss a sensor-based approach that makes nodes able to detect incorrect information and identify the nodes that are the source of the incorrect information

Yan et al. [21] propose a method to gather data from three resources: radar, traffic and neighboring cars with the aim to safeguard cars from attacks. Their system computes the similarity between these data.

In [22], vehicles and attacks are modeled in a logic language, and the effects of attacks are studied with co-simulation and theorem proving [23], [24].

III. Model Checking background

In the following, preliminary notions on formal methods are recalled. To apply formal methods we need a *formal model* representing the system under study, and a *formal system* to express and prove properties of the system. In this work, we adopt Milner's Calculus of Communicating Systems (CCS) [25] to build a formal model, and modal mu-calculus [26] as the formal system.

The Calculus of Communicating Systems is one of the best known process algebras. The basic idea of process algebras is representing the structure of concurrent processes as expressions that can be transformed algebraically according to given transformation (or inference) rules. The expressions are composed of atomic symbols representing system's actions, and operators to build and combine sequences of actions in various ways. The semantics of an expression of a process algebra are given by a labeled transition system (LTS), i.e, an automaton whose states correspond to process algebra expressions and whose transitions correspond to transformations of expressions. In this way, a process in its initial state is represented as an expression describing all potential evolutions of that state, depending on the occurrence of different actions, which in turn determine which transformation rules are applicable. A transition from a state s to a state s', due to an action α, is represented as $s \xrightarrow{\alpha} s'$.

Temporal logics are logical formalisms designed to express properties of the possible evolutions of LTSs. A model-checking prover is used to verify if a temporal formula holds for a given LTS.

The syntax of the extended mu-calculus is the following, where K ranges over sets of actions (i.e., $K \subseteq \mathcal{A}$) and Z ranges over system variables:

$$\phi ::= \texttt{tt} \mid \texttt{ff} \mid Z \mid \phi \wedge \phi \mid \phi \vee \phi \mid [K]\phi \mid \langle K \rangle \phi \mid \nu Z.\phi \mid \mu Z.\phi$$

A fixpoint formula may be either $\mu Z.\phi$ or $\nu Z.\phi$, where μZ and νZ *binds* free occurrences of Z in ϕ. An occurrence of Z is free if it is not within the scope of a binder μZ (resp. νZ). A formula is *closed* if it contains no free variables. $\mu Z.\phi$ is the least fixpoint of the recursive equation $Z = \phi$, while $\nu Z.\phi$ is the greatest one. From now on we consider only closed formulae.

Scopes of fixpoint variables, free and bound variables, can be defined in the mu-calculus in analogy with variables of first order logic.

The satisfaction of a formula ϕ by a state s of a transition system is defined as follows: each state satisfies $\texttt{tt}$ and no state satisfies $\texttt{ff}$; a state satisfies $\phi_1 \vee \phi_2$ ($\phi_1 \wedge \phi_2$) if it satisfies ϕ_1 or (and) ϕ_2. $[K]\,\phi$ is satisfied by a state which, for every performance of an action in K, evolves to a state obeying ϕ. $\langle K \rangle\,\phi$ is satisfied by a state which can evolve to a state obeying ϕ by performing an action in K.

For example, $\langle a \rangle\,\phi$ denotes that there is an a-successor in which ϕ holds, while $[a]\,\phi$ denotes that for all a-successors ϕ holds.

The precise definition of a closed formula satisfaction φ by a state p (written $p \models \varphi$) is given in Table I.

A transition system $\mathcal{T}$ satisfies a formula ϕ, written $\mathcal{T} \models \phi$, if and only if $q \models \phi$, where q is the initial state of $\mathcal{T}$.

In the sequel we will use the following abbreviations:

$$\begin{aligned}
\langle \alpha_1, \ldots, \alpha_n \rangle\,\phi &= \langle \{\alpha_1, \ldots, \alpha_n\} \rangle\,\phi \\
\langle - \rangle\,\phi &= \langle \mathcal{A} \rangle\,\phi \\
\langle -K \rangle\,\phi &= \langle \mathcal{A} - K \rangle\,\phi \\
[\alpha_1, \ldots, \alpha_n]\,\phi &= [\{\alpha_1, \ldots, \alpha_n\}]\,\phi \\
[-]\,\phi &= [\mathcal{A}]\,\phi \\
[-K]\,\phi &= [\mathcal{A} - K]\,\phi
\end{aligned}$$

In this work we resort to the Concurrency Workbench of New Century (CWB-NC) [27] formal verification environment, which supports several different specification languages, among which CCS. In the CWB-NC the verification of temporal logic formulae is based on model checking [28].

TABLE I
SATISFACTION OF A CLOSED FORMULA BY A STATE

$p \not\models \texttt{ff}$		
$p \models \texttt{tt}$		
$p \models \varphi \wedge \psi$	iff	$p \models \varphi$ and $p \models \psi$
$p \models \varphi \vee \psi$	iff	$p \models \varphi$ or $p \models \psi$
$p \models [K]_R \varphi$	iff	$\forall p'. \forall \alpha \in K. p \xrightarrow{\alpha}_{K \cup R} p'$ implies $p' \models \varphi$
$p \models \langle K \rangle_R \varphi$	iff	$\exists p'. \exists \alpha \in K. p \xrightarrow{\alpha}_{K \cup R} p'$ and $p' \models \varphi$
$p \models \nu Z. \varphi$	iff	$p \models \nu Z^n. \varphi$ for all n
$p \models \mu Z. \varphi$	iff	$p \models \mu Z^n. \varphi$ for some n

where for each n, $\nu Z^n.\varphi$ and $\mu Z^n.\varphi$ are defined as:

$$\nu Z^0.\varphi = \texttt{tt} \qquad \mu Z^0.\varphi = \texttt{ff}$$
$$\nu Z^{n+1}.\varphi = \varphi[\nu Z^n.\varphi/Z] \qquad \mu Z^{n+1}.\varphi = \varphi[\mu Z^n.\varphi/Z]$$

with $\varphi[\psi/Z]$ denoting the substitution of ψ for every free occurrence of the variable Z in φ.

IV. IDENTIFICATION OF ATTACKS

Our analysis starts from a Simulink model of the system under analysis. The system is simulated under operating conditions and logs of sensed data and data sent to actuators are generated. Safety requirements are identified on the system, as expected to be true. For example, if a safe distance between car is violated, brakes must be activated.

Then the design of the system is instrumented with the effect of a possible attack and the logs of the simulation are generated. The effect of the attack will result in a log with some different values if compared with the legitimate behavior.

The idea behind our proposal is that it is possible to discriminate behind a driving session with an attack (for instance, targeting the adaptive cruise control) from the legitimate expected behavior by simply analyzing the log retrieved from the vehicle under analysis i.e., without additional hardware and/or sensors. To this aim, we resort to formal verification techniques, that demonstrated to be effective in software verification and testing in the last years.

Then the method we propose is composed of two steps: *Formal model generation* (depicted in Figure 1) and *Potential attack detection* (depicted in Figure 3).

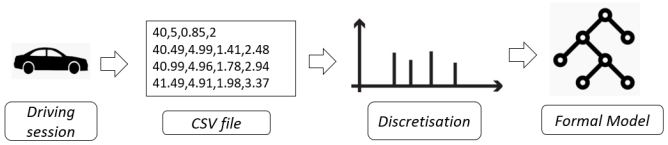

Fig. 1. Formal model generation.

As hinted in Figure 1, the values of relevant variables (*features*) are stored in comma-separated-values (CSV) format. The logs are then converted to a so-called discretized form, where each actual value is mapped to a symbolic value in the set $\{low, medium, high\}$, with respect to given thresholds. For example, the values of some feature could be mapped to low if less than or equal to -0.1, $medium$ if between -0.1 and 0.1, and $high$ if greater than or equal to 0.1.

The LTS is then built from the discretized log. Each log entry is the (abstract) state of the system at each simulation step, represented by the tuple of feature values. The set of values for each feature is taken as the set of actions of the LTS. Each state at step i is then defined by a CCS expression consisting in the parallel composition of processes, each composed of a sequence of feature values, all terminated by the process at step $i + 1$. More details about the model construction can be found in [29]. An example of formal model generation from a log with two features (i.e., F1 and F2) is shown in Figure 2.

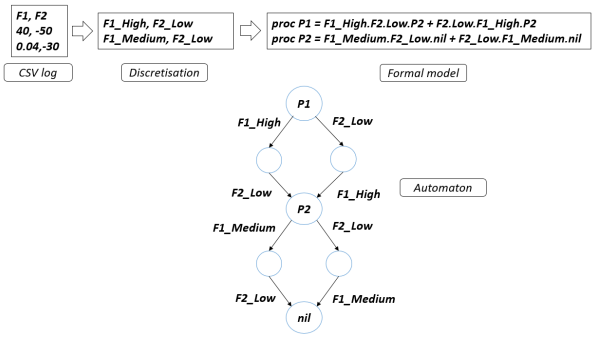

Fig. 2. An example of formal model generation.

Once the formal model has been generated and safety requirements for the expected behavior have been identified, a formal verification environment is used to verify whether an attack is potentially happening.

Let $\mathcal{S}(P)$ be the automaton corresponding to the log for a driving session and φ be the formalization of a safety requirement. If $\mathcal{S}(P)$ does not satisfy the safety requirement,

we suspect that a potential attack is happening. Figure 3 describes the potential attack detection step.

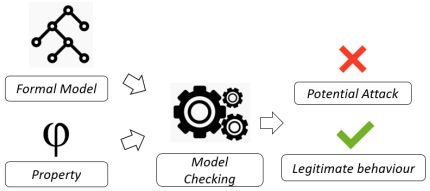

Fig. 3. Potential attack detection.

In this step, a formal model and a property are considered as input for the model checker. If the model checker returns *true*, the safety requirement is satisfied, and this is symptomatic that the formal model exhibits a legitimate behavior, otherwise if the model checker returns *false*, an attack is potentially happening.

Let us assume that actions b and c must never happen when the a action happen, we have to prove that the formal model satisfies the φ property, where:

$$\varphi = \nu X.[b][c]\ \varphi_1 \wedge [-]\ X$$
$$\varphi_1 = [a]\ \texttt{ff}$$

To verify a formal model in the CWB-NC environment, we use the *chk* command, with the formal model $\mathcal{S}(P)$ to verify and the φ property as inputs.

V. A Case Study: Adaptive Cruise Control System

As a case study, an adaptive cruise control system is considered, taken from the Matlab/Simulink documentation, which includes several examples in the automotive field[2]. The Simulink scheme is shown in Figure 4.

The *Lead car* (the box on the left of the figure) is parametric with respect to the acceleration function. The *Ego car* reads the actual position and the actual velocity of the Lead car. Then these parameters are provided as input to the Controller, which together with the actual position and velocity of the Ego car, computes the new acceleration of the Ego car.

Actually, the schema in Figure 4 is a variant of the original model. In particular, an attack has been added that changes the value of the position of the Lead car, after this value has been read by the Ego car sensors and before this value has been provided as an input to the Controller. As a proof of concept, we assume that the attack increases the position of the Lead car by 60 units. Graphically, the attack is modeled by a block named `Attack` in the figure.

Figure 5 shows the system behavior, under some operating conditions. In particular, D_{safe} is the safe distance between the Lead and the Ego car. The figure shows that the relative position is always greater than the safe distance and that in the steady state the Ego car has the same behavior of the Lead car.

[2]https://it.mathworks.com/help/mpc/ref/adaptivecruisecontrolsystem.html

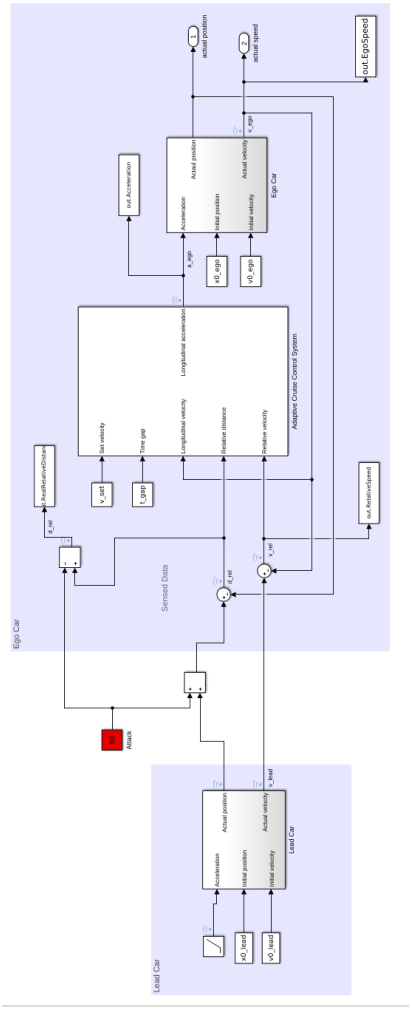

Fig. 4. Adaptive cruise control system (adapted from Matlab/Simulink).

Figure 6 shows the system behavior in case of attack. At time 14.5 sec, the relative position becomes negative, which means that there is a crash; at time 4.2 the safety requirement described below is violated.

Assume the log stores the following information: `RelativePos`, the difference between the actual positions of the two cars (read by sensors); `RelativeSpeed`, the relative velocity of the two cars; `EgoAcc`, the acceleration of the Ego car and `RelativePos - Dsafe`, the difference between the actual distance and the safety distance of the two cars. An excerpt of the Log file is shown in Table II, where `State` is the timestamp. The duration of a step is $0,1$. State 145 corresponds to $14,5$ sec in the simulation.

An example of safety requirement is the following:

> *if the distance between the cars is less than the safety distance and the difference betwee the speeds of the two cars is negative, then the Ego car must decelerate.*

Below we show the representation of the formula in mu-

TABLE II
LOG FILE (AN EXCERPT).

State	RelativePos	RelativeSpeed	EgoAcc	(RelativePos - Dsafe)
...	...	...	...	...
98	45.2629324696	-0.1734871587	-0.0543565760	0.0200504475
99	45.2459171683	-0.1668540851	-0.0544967007	0.0123214492
100	45.2295540452	-0.1604368351	-0.0538242986	0.0049424761
101	45.2138233922	-0.1542013153	-0.0526345966	-0.0020584492
102	45.1987079325	-0.1481315931	-0.0511222645	-0.0086762978
103	45.1841914158	-0.1422221882	-0.0468068141	-0.0149196477
104	45.1702569607	-0.1364976159	-0.0441163354	-0.0208397016
...	...	...	...	...

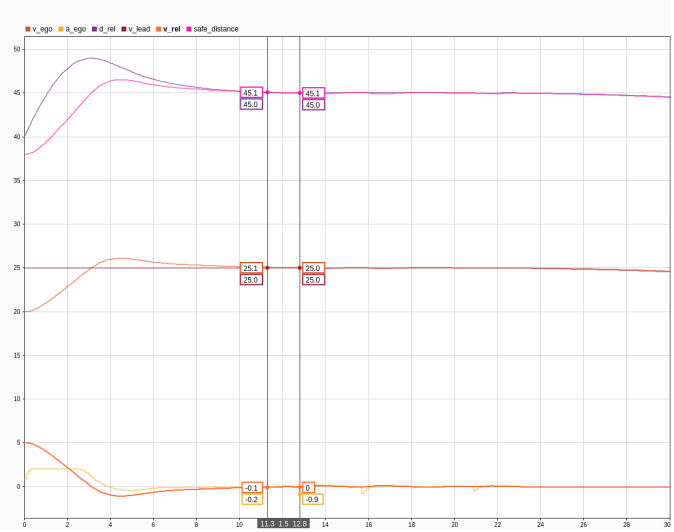

Fig. 5. Simulink output.

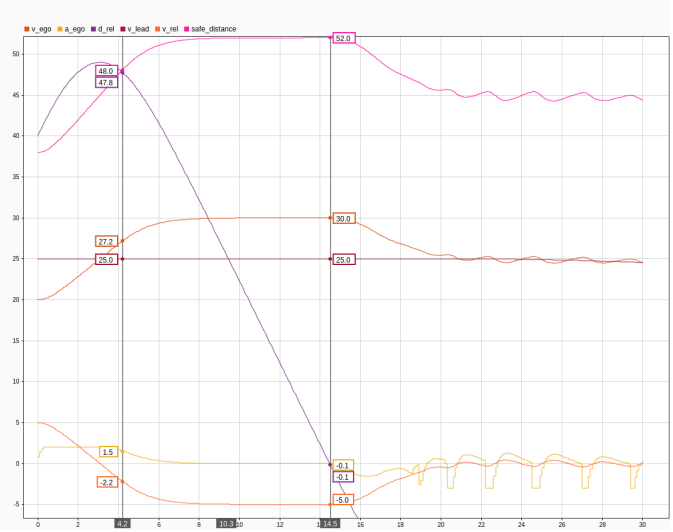

Fig. 6. Simulink output in case of attack.

calculus temporal logic:

$$\varphi = \nu X.[low\_RelativeSpeed]$$
$$[low\_RelativePos\_Dsafe]\ \varphi_1 \wedge [-]\ X$$
$$\varphi_1 = [high\_EgoAcc]\ \texttt{ff}$$

The φ formula describes the following property: it must never happen that the relative speed and the (relative position - Dsafe) assume a low value and the acceleration in the next state exhibits a high value.

Figure 7 shows the output of the model checker when the legitimate and the attack models are verified. When the legitimate model is evaluated, the φ property results *true* (i.e., the system is safe) while, when the attack model is evaluated, the CWB-NC outputs *false*, symptomatic of an unsafe behavior (i.e., a possible attack).

We show also a variant of safety requirement than can be used to anticipate the potential attack:

$$\chi = \nu X.[low\_RelativeSpeed]$$
$$[low\_RelativePos\_Dsafe]\ \chi_1 \wedge [-]\ X$$
$$\chi_1 = [medium\_EgoAcc]\ \texttt{ff}$$

```
CWB-NC
cwb-nc> load F:\legitimate_model.ccs
Execution time (user,system,gc,real):(0.254,0.000,0.073,0.254)
cwb-nc> chk P1 F
Invoking alternation-free model checker.
Building automaton...
.........1000.........2000.........3000.........4000.........5000
.........6000.........7000.........8000.........9000.........10000
.........11000.........12000...
States: 12342
Transitions: 19264
Done building automaton.
TRUE, the agent satisfies the formula.
Execution time (user,system,gc,real):(0.560,0.000,0.039,0.560)
cwb-nc> load F:\attack_model.ccs
Execution time (user,system,gc,real):(0.296,0.000,0.102,0.296)
cwb-nc> chk P1 F
Invoking alternation-free model checker.
Building automaton...
.........1000.........2000.........3000.........4000.........5000
.........6000.........7000.........8000.........9000.........10000
.........11000.........12000...
States: 12342
Transitions: 19264
Done building automaton.
FALSE, the agent does not satisfy the formula.
Execution time (user,system,gc,real):(0.545,0.000,0.028,0.545)
cwb-nc>
```

Fig. 7. The output of the model checker.

The χ formula describes the following property: it must never happen that the relative speed and the (relative position - Dsafe) assume a low value and the acceleration in the next state exhibits a medium value. This second formula can be helpful to alert that a potential attack can happen in a short

time (if the acceleration continues to increase). When the legitimate model is evaluated, the χ property results *true* while, when the attack model is evaluated, the CWB-NC outputs *false*, symptomatic that an unsafe behavior can happen in the near future. One of the interesting point of the proposed approach is that it adopts variables currently measured in vehicles.

VI. Conclusions and Future Work

This paper has introduced a method whereby simulated cyber-attacks to networks of ECUs in automotive systems can be identified from patterns in simulation logs. An algorithm has been designed and implemented that abstracts a system's behavior by building an LTS defined in CCS. A temporal logic formula to check if the attack can drive the system into an unsafe condition, written in the mu-calculus language, can then be proved or disproved with a model-checking prover. This procedure is arguably useful at an early design stage, in the phase of risk analysis, when simulation can be used to assess the consequences of various types of attacks. Further, it makes it possible to characterize the conditions revealing a possible attack and have them detected at run-time with a simple monitoring algorithm.

ACKNOWLEDGMENT

Work partially supported by the Italian Ministry of Education and Research (MIUR) in the framework of the CrossLab project (Department of Excellence) and by the PRA 2018_81 project entitled Wearable sensor systems: personalized analysis and data security in healthcare funded by the University of Pisa.

References

[1] F. Martinelli, F. Mercaldo, V. Nardone, A. Orlando, and A. Santone, "Cluster analysis for driver aggressiveness identification." in *ICISSP*, 2018, pp. 562–569.

[2] E. Massaro, C. Ahn, C. Ratti, P. Santi, R. Stahlmann, A. Lamprecht, M. Roehder, and M. Huber, "The car as an ambient sensing platform," *Proceedings of the IEEE*, vol. 105, no. 1, pp. 3–7, 2017.

[3] 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 *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE, 2010, pp. 447–462.

[4] M. F. Carfora, F. Martinelli, F. Mercaldo, V. Nardone, A. Orlando, A. Santone, and G. Vaglini, "A "pay-how-you-drive" car insurance approach through cluster analysis," *Soft Computing*, vol. 23, no. 9, pp. 2863–2875, 2019.

[5] F. Martinelli, F. Mercaldo, V. Nardone, and A. Santone, "Car hacking identification through fuzzy logic algorithms," in *Fuzzy Systems (FUZZ-IEEE), 2017 IEEE International Conference on*. IEEE, 2017, pp. 1–7.

[6] F. Martinelli, F. Mercaldo, A. Orlando, V. Nardone, A. Santone, and A. K. Sangaiah, "Human behavior characterization for driving style recognition in vehicle system," *Computers & Electrical Engineering*, 2018.

[7] F. Martinelli, F. Mercaldo, V. Nardone, A. Santone, and G. Vaglini, "Real-time driver behaviour characterization through rule-based machine learning," in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2018, pp. 374–386.

[8] F. Martinelli, F. Mercaldo, V. Nardone, A. Orlando, and A. Santone, "Who's driving my car? a machine learning based approach to driver identification," in *ICISSP*, 2018.

[9] B. I. Kwak, J. Woo, and H. K. Kim, "Know your master: Driver profiling-based anti-theft method," in *PST 2016*, 2016.

[10] F. Martinelli, F. Mercaldo, V. Nardone, A. Santone, A. K. Sangaiah, and A. Cimitile, "Evaluating model checking for cyber threats code obfuscation identification," *Journal of Parallel and Distributed Computing*, vol. 119, pp. 203–218, 2018.

[11] M. S. Al-Kahtani, "Survey on security attacks in vehicular ad hoc networks (vanets)," in *Signal Processing and Communication Systems (ICSPCS), 2012 6th International Conference on*. IEEE, 2012, pp. 1–9.

[12] G. Samara, W. A. Al-Salihy, and R. Sures, "Security issues and challenges of vehicular ad hoc networks (vanet)," in *New Trends in Information Science and Service Science (NISS), 2010 4th International Conference on*. IEEE, 2010, pp. 393–398.

[13] "Simulink web site." 2016, http://www.mathworks.com/products/simulink.

[14] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, T. Kohno *et al.*, "Comprehensive experimental analyses of automotive attack surfaces." in *USENIX Security Symposium*. San Francisco, 2011.

[15] S. Woo, H. J. Jo, and D. H. Lee, "A practical wireless attack on the connected car and security protocol for in-vehicle can," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, pp. 993–1006, 2015.

[16] A. Taylor, S. Leblanc, and N. Japkowicz, "Anomaly detection in automobile control network data with long short-term memory networks," in *Data Science and Advanced Analytics (DSAA), 2016 IEEE International Conference on*. IEEE, 2016, pp. 130–139.

[17] K. M. A. Alheeti, A. Gruebler, and K. D. McDonald-Maier, "An intrusion detection system against malicious attacks on the communication network of driverless cars," in *2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*. IEEE, 2015, pp. 916–921.

[18] W. Liu, H. Zhang, and W. Zhang, "An autonomous road side infrastructure based system in secure vanets," in *2009 5th International Conference on Wireless Communications, Networking and Mobile Computing*. IEEE, 2009, pp. 1–6.

[19] N. Lyamin, A. V. Vinel, M. Jonsson, and J. Loo, "Real-time detection of denial-of-service attacks in ieee 802.11 p vehicular networks." *IEEE Communications letters*, vol. 18, no. 1, pp. 110–113, 2014.

[20] P. Golle, D. Greene, and J. Staddon, "Detecting and correcting malicious data in vanets," in *Proceedings of the 1st ACM international workshop on Vehicular ad hoc networks*. ACM, 2004, pp. 29–37.

[21] G. Yan, S. Olariu, and M. C. Weigle, "Providing vanet security through active position detection," *Computer Communications*, vol. 31, no. 12, pp. 2883–2897, 2008.

[22] C. Bernardeschi, A. Domenici, and M. Palmieri, "Modeling and simulation of attacks on cyber-physical systems," in *Proceedings of the 5th International Conference on Information Systems Security and Privacy, ICISSP 2019, Prague, Czech Republic, February 23-25, 2019*, 2019, pp. 700–708.

[23] M. Palmieri, C. Bernardeschi, and P. Masci, "Co-simulation of semi-autonomous systems: The line follower robot case study," in *Software Engineering and Formal Methods - SEFM 2017 Collocated Workshops: DataMod, FAACS, MSE, CoSim-CPS, and FOCLASA, Trento, Italy, September 4-5, 2017, Revised Selected Papers*, 2017, pp. 423–437.

[24] C. Bernardeschi, A. Domenici, and P. Masci, "A pvs-simulink integrated environment for model-based analysis of cyber-physical systems," *IEEE Trans. Software Eng.*, vol. 44, no. 6, pp. 512–533, 2018. [Online]. Available: https://doi.org/10.1109/TSE.2017.2694423

[25] R. Milner, *Communication and concurrency*, ser. PHI Series in computer science. Prentice Hall, 1989.

[26] D. Kozen, "Results on the propositional mu-calculus," *Theor. Comput. Sci.*, vol. 27, pp. 333–354, 1983. [Online]. Available: http://dx.doi.org/10.1016/0304-3975(82)90125-6

[27] R. Cleaveland and S. Sims, "The NCSU concurrency workbench," in *Computer Aided Verification, 8th International Conference, CAV '96, New Brunswick, NJ, USA, July 31 - August 3, 1996, Proceedings*, 1996, pp. 394–397. [Online]. Available: https://doi.org/10.1007/3-540-61474-5_87

[28] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model checking*. MIT Press, 2001. [Online]. Available: http://books.google.de/books?id=Nmc4wEaLXFEC

[29] L. Brunese, F. Mercaldo, A. Reginelli, and A. Santone, "Formal methods for prostate cancer gleason score and treatment prediction using radiomic biomarkers," *Magnetic resonance imaging*, 2019.