UNCOVER: Data-Driven Design Support through Continuous Monitoring of Security Incidents

Matthias Stammler*, Julian Lorenz*, Eric Sax*, Jürgen Becker*, Matthias Hamann‡, Patrick Bidinger‡, Andreas Dewald‡, Paraskevi Georgouti§, Alexios Camarinopoulos§, Günter Becker§, Klaus Finsterbusch¶, Maximilian Kirschner†, Laurenz Adolph†, Carl Philipp Hohl†, Maria Rill†, Daniel Vonderau† and Victor Pazmino Betancourt†

*Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany, Email: {stammler, julian.lorenz, sax, becker}@kit.edu

‡ERNW Research GmbH, Heidelberg, Germany, Email: {mhamann, pbidinger, adewald}@ernw.de

§RISA Sicherheitsanalysen GmbH, Berlin, Germany, Email: {p.georgouti, alexios.camarinopoulos, guenter.becker}@risa.de

¶EnCo Software GmbH, Berlin, Germany, Email: k.finsterbusch@enco-software.com

†Forschungszentrum Informatik, Karlsruhe, Germany, Email: {kirschner, adolph, hohl, m.rill, vonderau, pazmino}@fzi.de

***Abstract*—The seamless and secure integration of subsystems is a pivotal requirement within contemporary automotive development, necessitating the application of design methodologies like the Vee model. While this approach includes dedicated verification steps for the included contexts and provides a high level of assurance that the system will operate correctly under specified conditions, formalizing specifications outside its operational design domain is per definition not included. Additionally, black-box systems like machine learning based functions prove difficulty to test by these traditional methodologies.**

In this project, we introduce and demonstrate a design workflow combining the Vee model design paradigm with continuous data-driven software engineering. Our workflow assists the continuous, safe and secure development and improvement of consumer vehicle functionality over the product lifecycle. This is achieved through the continuous monitoring of anomalies, as well as system states that deviate from the established design domain. The UNCOVER methodology consists of a continuous reduction in the amount of necessary monitored messages and presents a methodology throughout the entirety of the product lifecycle.

We demonstrate our methodology through a simulation and show our automatic generation of monitoring components, and an automated preselection of identified safety or security incidents.

***Index Terms*—Automotive Development, Continuous Data-Driven Software Engineering, Safety- and Security Incidents, In-Vehicle Monitoring**

I. Introduction and Project Overview

The Vee model proves to be an effective design method for non-functional properties and their implementation in a given system [1]. One methodology for this is the usage of the X-by-construction paradigm during design time [2] and is employed in different public funded projects [3]. There exist a plethora of formal models to generate isolation [4] or guarantee predictable multicore execution [5]. These methods necessitate an abstraction of real life systems to fit inside the respective predefined abstraction and can be leveraged by tools like ENCOs tool SOX. These tools allow developers to find potential safety hazards or security vulnerabilities.

This work was funded by the German Federal Ministry of Education and Research (BMBF) under grant number 16KIS1414 (UNCOVER). The responsibility for the content of this publication lies with the authors.

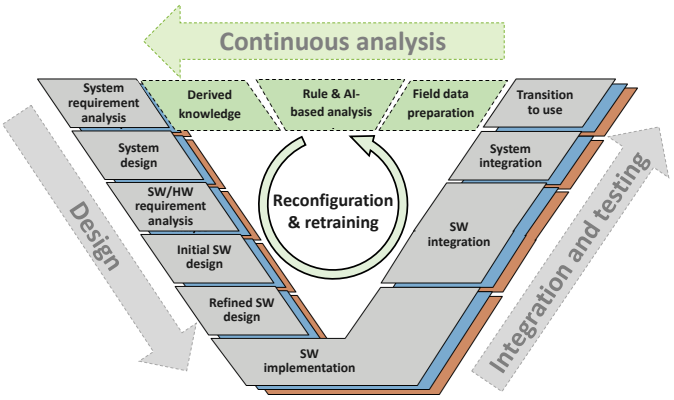

Fig. 1. Proposed development process from the UNCOVER project. The classic Vee model (gray) is complemented by a design loop of continuous analysis and development (green), leveraging field data monitored from selected vehicles.

However, during the operational phase, additional vulnerabilities and security gaps may arise, which were not known during system design or arise from the necessary abstraction. Changes to requirements are expected during the operation of autonomous systems, which can affect the system itself, its environment, or updates to existing functions in the vehicle. Each change creates the chance for a new path for information flow inside the considered systems communication system.

These continuous changes pose potential risks to safety and security, which can lead to considerable risks for road users by affecting the functional safety and the fulfillment of the safety of the intended functionality (SOTIF), either through malfunctions resulting from security incidents or from safety incidents directly. Moreover, it is essential to continuously ensure safety and security over the entire product lifecycle of the vehicles and systems until they are taken out of service.

This is not supported in the traditional Vee model design, where the development is split hierarchically from the broad system architecture down to the software architecture and software design. During integration, this abstraction is reversed from unit tests over integration test and systems integration

979-8-3503-4859-0/DATE24/© 2024 EDAA

to validation [1]. Therefore, exclusive development-supporting measures are no longer sufficient and must be continuously supported by feedback from operational experience.

This project aims to overcome challenges in developing and operating safe and secure driving functions, by complementing a systematic feedback loop of field data, including detected security incidents into the development or update phase, depicted in Fig. 1. These security incidents include detected cyberattacks manifesting themselves in information flow not covered by the original system design, as well as detected changes in the operational design domain.

Detected security risks, such as attack paths uncovered during operation, can be identified, analyzed, and linked to security analyses, requirements, models, and measures from the design phase. This enables our workflow to adjust the development of new safety and security measures, allowing continuous treatment of malfunctions, closure of security gaps, and the safe update of functionality.

The main contribution of this project and its goal consist of designing, implementing and validating a workflow for a *continuous feedback loop* of *monitored data* into the *development process* over the product lifecycle, while simultaneously reducing the amount of needed monitored data to detect the occurrence of incidents.

The rest of the paper is structured as follows:

1) Section II: a monitoring platform with automatically generated rules and conditions, selectively collecting information flow inside a vehicle or a fleet of vehicles.
2) Section III: a workflow and tool set for continuously reducing the amount of needed human classification of detected security incidents through automatic preselection and accompanying knowledge management.
3) Section IV: a simulation system, allowing for insertion and replayability of security incidents validating the presented methods.

UNCOVER is funded by the German Federal Ministry of Education and Research (BMBF), with a project time from June 2021 until December 2023. The consortium consists of ERNW Reaseach, RISA, ENCO as well as the Forschungszentrum Informatik and the Karlsruhe Institute of Technology.

II. Monitoring Platform

Connected to an automotive CAN bus and listening to the presented vehicle data transmitted via CAN mutliframe messages, a monitoring platform is placed. This component represents an onboard monitoring unit inside o an automotive vehicle. Section II-A1 describes the model employed for automatic code generation, including configuration of monitoring components in the form of hardware descriptions, as well as generating accompanying software. These components need to fulfil the requirements of subsequent and preceding system parts, which include the specification of information flow deemed suspicious as well as the read-out format of collected data. Fig. 2 shows the platform components. Using the encryption and decryption modules, described in section II-B, end-to-end encryption is facilitated. The platform is connected to the aforementioned CAN bus, where incoming messages are promptly encrypted.

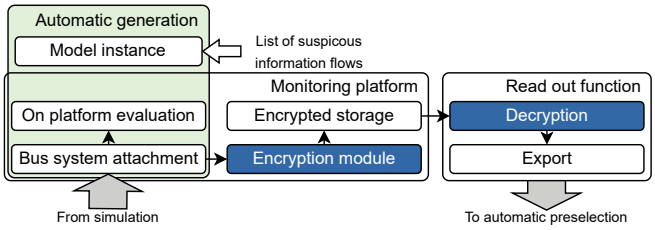

Fig. 2. Components of monitoring platform. Components inside the green box are generated based on a given system model. Arrows denote information flow of monitored data inside the component. Blue modules show encryption and decryption. Large arrows show interfaces to other parts of the UNCOVER workflow.

A. Model based generation

Generation of monitoring components is facilitated by using a multileveled model for monitoring hardware to trace information flow over automotive communication systems [6]. Table I shows a short description of the respective model elements.

1) Mathematical Software Model: A model instance is described as:

$$M = (M_P, M_I, M_F),$$

with the first layer being defined as:

$$M_P = (U, L, \varphi_L, \delta_R),$$

the second layer being defined as:

$$M_F = (F, I_S, I_{Norm}, R, \sigma_F, \varphi_R),$$

and the third layer being defined as:

$$M_I = (V, C, \varphi_V, \varphi_C, \omega_V).$$

The first layer, M_P, describes the platform architecture and contains: the set of units U; the set of links L; as well as $\varphi_L : L \to \mathbb{P}(U)$, mapping each link to a set of connected units; and $\delta_R : L \to \{0, 1\}$ specifying if monitoring components can be generated for this link.

The second layer, M_F, describes the functional implementation and contains the set of features F, describing abstract information sources and sinks; the set of suspicious information flow $I_S \subseteq F \times F$, which models unwanted system behavior; the set of normal system behavior $I_{Norm} \subseteq F \times F, I_S \cap I_{Norm} = \varnothing$, which specifies information flow during normal operation; a set of rules R, defining when information flow is deemed suspicious; as well as $\sigma_F : F \to U$, mapping each feature to a corresponding execution unit and $\varphi_R : R \to \mathcal{P}(I_S)$ and the mapping of every rule to one or more elements in I_S.

The third layer, M_I, describes the functional implementation and contains: the set of (system) variables V; the set of conditions C; as well as $\varphi_V : V \to F$, mapping each variable to its produced feature; $\varphi_C : C \to \mathcal{P}(V)$, mapping to

TABLE I
MODEL USED FOR MONITORING GENERATION [6]

Layer	Model element	Description
M_P	$U, L, \varphi_L : U \to L$	Units, links & mapping
	$\delta_R : U \cup L \to \{0, 1\}$	Unit reachability
M_F	F_{in}, F_{out}	Input and output features
	$I \subseteq F_{in} \times F_{out}$	Intended information flow
	I_S	Suspicious information flow
	$R, \varphi_R : R \to \mathcal{P}(I_S)$	Suspicious rules & mapping
	$\delta_F : F \to U$	Mapping of features to units
M_I	$V, \varphi_V : V \to F$	System variables and mapping to F
	$C, \varphi_C : C \to \mathcal{P}(V)$	Conditions & mapping to V
	$\omega_V : V \to F \times F$	Assignment of V to network paths

each condition one or more variables; and $\omega_V : V \to F \times F$ specifying, where every system variable is sent to and from. This work has been presented in [6]. Exchange of information is modeled via the so-called information flow over multiple features.

One instance represents one automotive system and is then used for the monitoring of communication systems and to describe the used hardware components. Detected information flow, not specified in $I_{\text{norm}} \in M_S$ as well as system variable in the monitored software system, can be used as an element in the set of rules $R \in M_F$ or the set of conditions $C \in M_I$. This allows for a monitoring strategy, that observes the software as well as the hardware of an automotive system, in respect of the flow of information not previously specified or anticipated by the designer.

2) Generation of Monitoring Components: Generation of monitoring components is facilitated for each model instance as shown in fig. 2. The lower part is parameterizing functions for a given software system. Resulting from this, a software frame is the input into the next step, where possible monitoring functions generated. These functions are then inserted in a given code base.

For a communication system, an instance of M needs to be provided to the instance evaluation. Out of those, a list of suspicious addresses and bus system messages is generated. This list is then used to generate monitoring hardware description files using the Chisel framework [7]. Additionally, functions parameterized during the generation of the software monitoring system are registered as elements in the sets $R \in M_F$ as well as $C \in M_I$. Lastly, the monitoring subsystems for software and communication systems are combined in the deployment stage, resulting in a bit stream ready for programming an FPGA.

The generated monitoring code is deployed onto a MicroBlaze soft core processor. Every monitored CAN message ID is associated to a locally assigned buffer, where messages are stored until all multi-frame messages are received. Incoming messages are checked for their expected multi-frame index. In case the multi-frame index mismatches with the expectation, already present message parts are deleted, and the buffer is cleared.

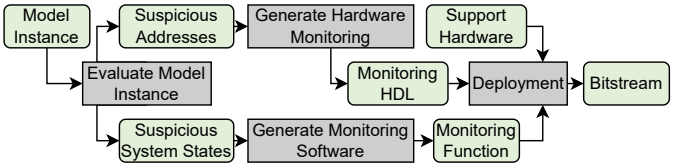

Fig. 3. Generation workflow. Top: generation for monitoring software components. Bottom: Generation of hardware components.

B. Encryption Module

While the data stored by the monitoring platform for further analysis in the backend is likely not to contain canonical examples of sensitive data such as credit card or social security numbers, respective data needs to be thoroughly protected, nonetheless. Through the easily resolvable *car-driver* link, even seemingly innocuous data such as start-up time or acceleration profiles might reveal sensitive personal information up to uncovering a travel route. Not only the data at rest on the monitoring platform, but also the data in transit during communication with the backend, needs to be protected.

In order to gain an improved view of the overall car state and processes, a future variant of the monitoring platform might employ additional sensors at particular car components. These sensors might not transmit their data via conventional bus systems, but, e.g., wirelessly on an RFID basis, making the transmitted data a potentially straightforward target for eavesdropping and manipulation.

Especially, this preparation for further extension through RFID-based sensors imposes harsh restrictions on the tolerable resource demands of a corresponding cryptographic scheme. Therefore, a local hardware encryption module based on the DRACO stream cipher [8] is deployed in the monitoring platform.

The cipher's key size is 128 bits and its IV size is 96 bits. In an ultra-lightweight scenario like a low-cost RFID tag that has the secret key burned into the device or stored in an EEPROM and whose transmission module holds non-repeating frame numbers that can be used as IVs when encrypting the respective data packets, DRACO needs 23 percent less chip area and 31 percent less power than the established lightweight stream cipher Grain-128a [9] at 10MHz. In this work, the DRACO hardware module is further optimized to resist the cryptanalytic approach of Banik [10] and extended to feature the "authenticator generator" of Grain-128AEADv2 [11].

In consequence, the onboard monitoring platform developed features a model based generation scheme as well as a state-of-the-art lightweight authenticated encryption module, which protects data at rest and in transit from eavesdropping and manipulation, and allows for straightforward expandability of the monitoring domain via low-cost RFID sensors.

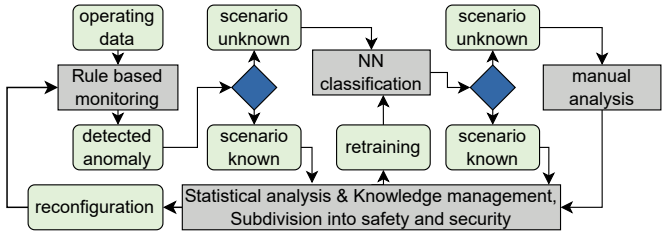

Fig. 4. Identification flowchart of a given anomalous event.

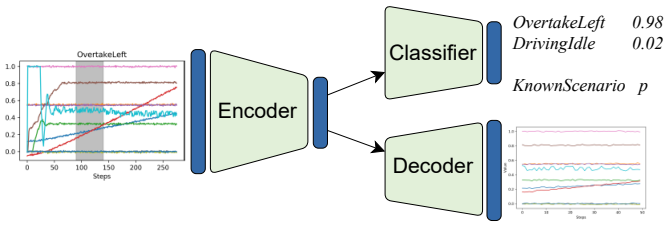

Fig. 5. Architecture of the NN for anomaly detection.

III. Continuous Analysis and Development

Consider a fleet of vehicles each equipped with one instance of the monitoring platform as described in Section II, likely to produce enormous amounts of data, which have to be categorized automatically in order to make them useful for the developing process.

There are many states and execution environments having an impact on the categorizing by the monitoring platform. Furthermore, a considerable part of the events produced will contain discrepancies or contradictions between different sources of information (e.g. speed information from GPS or from speedometer). In such cases, interaction with human experts is required to obtain additional information. Thus, a central and partial manual classification of elements is necessary to reduce future human load.

The continuous analysis and development therefore consists of three parts:

1) A neural network application determining whether the considered event has occurred previously and a check against an accompanying database of occurred events.
2) Human intervention, if a detected event could not be classified into an existing class.
3) Statistical model to calculate characteristics of the events of a given type including a failure rate estimation using a hierarchical model to reflect different conditions.

Fig. 4 shows a flowchart of the processes as described in this section. The automated parts are implemented in Python and the COODEXX database is used for knowledge management.

A. Event Identification

Detection of an anomaly by one instance of the monitoring platform results in a trace of n monitored messages. This trace is sent to a backend application and identified using the following process.

Before deployment, a whitelist of nominal behavior of the driving function and the corresponding bus messages is created during testing of the driving function. Using this whitelist, the set of rules R for the monitoring platform is derived, and the Neural Network (NN) classifier is initially trained.

Because of limited decision time and available computing resources inside the monitoring platform, a finalized decision can't be achieved onboard only using a rule-based approach, resulting in the usage of the NN based classifier. This approach is described in further detail in section III-A1. If the NN is able to classify the anomaly the information is added to the database, if not, human assessment (section III-B), is needed and the event is added to the database as well as to the whitelist of known events. This growing whitelist is then used to retrain the NN classifier and reconfigure the monitoring platform (section II). The database is able to store vehicle and event metadata like observation period or driven kilometers. Additionally, legal norms as well as regional, national or international laws are registered and automatically presented to corresponding events. These include the Vienna Convention, StVG and AFGBV or DSGVO, because of the sensitivity of the personal data involved.

1) Neural Network based Classification: The NN used for classifying bus traces consists of an autoencoder and a classifier. The neural network architecture is shown in Fig. 5.

The encoder and the decoder each consist of three fully connected layers with dropout layers. The training data for this NN is the initial whitelist of nominal behavior of the driving function from the simulation in Section IV. Using Monte-Carlo Dropout sampling, the uncertainty of the classifier can be calculated. With this uncertainty, the anomaly can be classified as either being similar to another signal trace of the initial whitelist or as a new scenario. As the uncertainty is calculated for the whole trace, the parts that contribute most to the deviation from the whitelist can be highlighted, simplifying human assessment of unknown anomalies.

B. Human Event Assessment and Reuse of Data Analysis

In case the monitoring platform detects an unknown behavior or event, further human analysis is needed to distinguish critical from uncritical events. The identification of a critical event is the first step in an analysis process, potentially revealing a security incident. Depending on the amount and quality of data, knowledge can be gained on attack methods and exploited weaknesses, supporting future reuse and continuous analysis. Analyzing monitored events depends on a safety & security analysis of the monitored system. For our work, we use SOX, a model-based safety & security analyzing tool supporting system modelling using both Unified Modeling Language (UML) and Systems Modeling Language (SysML). The system model is used to describe the structure and intended behavior of the system including activities, state machines and interactions as well as constraints.

Event data from the monitoring platform provide a time series of data flow and system states. By visualizing time series data and accounting for any constraints of the monitoring parameters, unintended behavior can be identified, and the corresponding event marked as potentially critical. Furthermore, from analyzing the unintended behavior, a failure category is derived. As a second step, the monitored behavior can be analyzed in the context of the system model. Suspicious signals or information flow is related to components based on the type of data, and a decision can be made whether the system design supports the analyzed behavior. In case a specific failure mode can be identified in the monitoring data, this can be combined with an initial safety and security analysis to get a hint of what might have caused the unintended behavior, including catalogs of known cybersecurity attacks to support continuous analysis like [12].

Analyzing monitoring traces followed by a detailed system analysis may reveal a cybersecurity attack. The results can be used in SOX via a thread analysis and risk assessment (TARA) to iteratively analyze a categorized event. Depending on the level of detail, the knowledge gained might be represented as a threat scenario associated with a certain cybersecurity property of an asset, or as one or more attack steps with a detailed description of affected components and vulnerabilities. Finally, a cybersecurity control can be defined to mitigate the discovered risk. All cybersecurity-related elements like single attack steps and attack paths as well as discovered vulnerabilities and countermeasures are stored in catalogs for reuse and to support the continuous analysis and development.

C. Statistical Methods

Classic frequentest and Bayesian estimation methods, as they have been known for many years, are based on the assumption that all events summarized in a population are homogeneous with regard to the characteristic under consideration, i.e. in the case of failures of technical components, that all components have the same failure rate. In the case of barriers against cyberattacks, it might be assumed that all considered barriers have the same failure rate. Because a fleet of cars can not be considered homogeneous, other hierarchical estimation methods are used, which have become known as *2-Stage Bayes* or *super population approach*.

The super population method was first introduced by Kaplan [13] and was established [14], [15]. The added subpopulations serve to estimate an a priori distribution, on the basis of which a Bayesian evaluation is carried out for the subpopulation under consideration. For each of the total of k subpopulations there is a pair of values $E = (n, T)$ or $E = (n, N)$, namely the number of failures experienced therein n and the time base T or the frequency of demands N [1]. The subpopulation for which an evaluation is to be carried out is denoted by E_1, the others are denoted by $E_2, ..., E_k$, and is denoted in their entirety by $\underline{E}$. In order to take into account the variability of the rate or probability per request, this variable is taken as a random variable that satisfies a given distribution law with unknown parameters $\underline{\Psi}$. The unknown parameters $\underline{\Psi}$ of this population distribution are determined by Bayesian estimation as follows.

[1]Subsequently the method is described for failure rates (parameter of the exponential distribution.). A similar argument can be derived for the failure probability per demand (parameter of the binomial distribution)

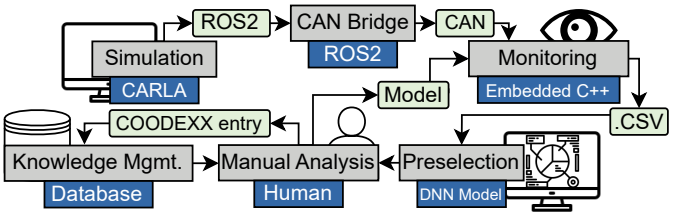

Fig. 6. Project component overview. Starting with a CARLA simulation Section IV, the vehicle data is forwarded over a ROS2 [16] bridge to the developed monitoring platform Section II. Afterward, the analysis platform (Section III) performs an automatic preselection using a continuously refined DNN model. The developer has the opportunity and knowledge to refine the monitoring platform. Detected incidents are entered in a knowledge management database. Steps are shown in gray, used tools in blue and exchange formats are marked in green.

$$g(\underline{\psi}|\underline{E}) = \frac{g(\underline{\psi})L(\underline{E}|\underline{\psi})}{\int_{(\underline{\psi})} g(\underline{\psi})L(\underline{E}|\underline{\psi})d\underline{\psi}}$$

$g(\underline{\psi})$ is an a priori distribution density reflecting any existing prior information about the unknown. The likelihood function $L(\underline{E}|\underline{\psi})$ results given pair-wise independence of the subpopulations as:

$$L(\underline{E}|\underline{\psi}) = \prod_{i=2}^{k} L(E_i|\underline{\psi})$$

Subsequently, the target value λ or p shall be denoted by x. Applying the law of total probability and the factors in this product yields

$$L(E_i|\underline{\psi}) = \int_{(x)} L(E_i|x)g(x|\underline{\psi})dx$$

If the target is to calculate a (failure) rate, the likelihood for subpopulation i with n_i failures in T_i time is

$$L(E_i|x) = L((n_i, T_i)|\lambda) = \frac{\lambda^{n_i}}{n_{i!}} e^{-\lambda T_i}$$

Given this, the distribution density function g is completely defined. Again, applying the law of total probability results in

$$f(x|\underline{E}) = \int_{(\underline{\psi})} g(x|\underline{\psi})g(\underline{\psi}|\underline{E})d\underline{\psi}$$

which is the resulting prior distribution density for x. A posterior taking into account the specific evidence E_1 is then according to the law of Bayes

$$f(x|E_1, \underline{E}) = \frac{L(E_1|x)f(x|\underline{E})}{\int_{(x)} L(E_1|x)f(x|\underline{E})dx}$$

Reasoning has been produced to use the lognormal distribution as a population distribution.

TABLE II
FORMAL DEFINITION OF THE LOGICAL SCENARIO *OvertakeLeft*

OvertakeLeft	Road lanes	≥ 2
	Right lane width	$3\,\mathrm{m}$
	Left lane width	$3\,\mathrm{m}$
	Ego speed	$v_e \sim U(20\,\mathrm{km/h}, 30\,\mathrm{km/h})$
	Other car speed	$v_o \sim U(50\,\mathrm{km/h}, 70\,\mathrm{km/h})$
	Other car lane	*is left of* ego car lane
	Other car pos	*is behind of* ego car pos
	Other car distance	$\leq 200\,\mathrm{m}$

IV. SIMULATION

Demonstrating our approach, the Car Learning to Act (CARLA) [17] environment is used to simulate a real world consumer vehicle. CARLA is an open-source simulation for vehicles and allows for sensor data, such as cameras, LIDAR, or radar sensor outputs to be simulated. The CARLA Scenario Runner is used to simulate and record specific driving scenarios. It allows for the definition of logical scenarios and generating concrete scenarios with randomized parameters. These include scenarios like *OvertakeRight*, and *DrivingIdle*. Table II exemplary shows the definition of the logical scenario *OvertakeLeft*. In this scenario, the ego vehicle drives on the highway and is overtaken by another actor on the left lane. The LIDAR data can be used to detect the distance between the ego vehicle and a preceding vehicle. This information is used by an Adaptive Cruise Control (ACC) application, maintaining a safe predefined distance from the preceding vehicle if, it is travelling slower than the maximum speed limit.

The Robot Operating System 2 (ROS2) [16] framework is used for communication between simulation and ACC application, as well as for recording data, paralleling an automotive onboard communication system. A special ROS2 node is present inside the network for sending ROS2 messages from the simulation and ACC commands to CAN frames, and transmitting them via a USB-CAN adapterto the monitoring platform. Messages are transmitted as CAN multi-frame messages according to ISO 15765-2 [18]. The CAN messages are then monitored and encrypted by an instance of the presented monitoring platform. Connected to this instance, the evaluation, and database is placed, completing the workflow.

V. CONCLUSION

In this project, a workflow to monitor automotive communication inside active vehicle fleets and use the monitored traces to refine the future development process as well as further analysis is presented and researched. By making use of continuous monitoring, the workflow allows for an adaptation of the configuration of the monitoring platform as well as the subsequent anomaly detection to lessen the need for human intervention or classification of subsequent events.

The goal of classifying every scenario into known or unknown as well as malicious or benign is achieved by retraining a neural network model to classify monitored scenarios upon detecting an event. Scenarios deemed benign will be excluded from triggering future monitoring. Scenarios detected as malicious or classified as unknown are reinforced into the model after classification into benign or malicious. An accompanying knowledge management is added. Our approach is supplemented and validated by a demonstrator. With this workflow as the project's result, we extend the Vee model by a continuous monitoring of safety and security incidents, allowing the development and software engineering process to encompass continuous data-driven design support.

REFERENCES

[1] K. Forsberg and H. M. Co-Principals, "4 system engineering for faster, cheaper, better," in *INCOSE International Symposium*, vol. 9, pp. 924–932, Wiley Online Library, 1999.

[2] M. H. ter Beek, L. Cleophas, I. Schaefer, and B. W. Watson, "X-by-construction," in *Leveraging Applications of Formal Methods, Verification and Validation. Modeling* (T. Margaria and B. Steffen, eds.), (Cham), pp. 359–364, Springer International Publishing, 2018.

[3] L. Masing *et al.*, "Xandar: Exploiting the x-by-construction paradigm in model-based development of safety-critical systems," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1–5, 2022.

[4] T. Dörr, T. Sandmann, H. Mohr, and J. Becker, "Employing the concept of multilevel security to generate access protection configurations for automotive on-board networks," in *2021 24th Euromicro Conference on Digital System Design (DSD)*, pp. 107–114, 2021.

[5] A. Biondi and M. Di Natale, "Achieving predictable multicore execution of automotive applications using the let paradigm," in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 240–250, 2018.

[6] M. Stammler, T. Harbaum, and J. Becker, "Mitigating masking in automotive communication systems: Modeling and hardware generation," in *2023 26th Euromicro Conference on Digital System Design (DSD)*, 2023.

[7] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović, "Chisel: Constructing hardware in a scala embedded language," in *DAC Design Automation Conference 2012*, pp. 1212–1221, 2012.

[8] M. Hamann, A. Moch, M. Krause, and V. Mikhalev, "The DRACO stream cipher A power-efficient small-state stream cipher with full provable security against TMDTO attacks," *IACR Trans. Symmetric Cryptol.*, vol. 2022, no. 2, pp. 1–42, 2022.

[9] M. Ågren, M. Hell, T. Johansson, and W. Meier, "Grain-128a: a new version of grain-128 with optional authentication," *Int. J. Wirel. Mob. Comput.*, vol. 5, no. 1, pp. 48–59, 2011.

[10] S. Banik, "Cryptanalysis of draco," *IACR Trans. Symmetric Cryptol.*, vol. 2022, no. 4, pp. 92–104, 2022.

[11] M. Hell, T. Johansson, A. Maximov, W. Meier, and H. Yoshida, "Grain-128aeadv2: Strengthening the initialization against key reconstruction." Cryptology ePrint Archive, Paper 2021/751, 2021. https://eprint.iacr.org/2021/751.

[12] S. K. Khan *et al.*, "Cyber-attacks in the next-generation cars, mitigation techniques, anticipated readiness and future directions," *Accident Analysis & Prevention*, vol. 148, p. 105837, 2020.

[13] S. Kaplan, "On a "two-stage" bayesian procedure for determining ailure rates from experimental data," *IEEE Transactions on Power Apparatus and Systems*, vol. PAS-102, no. 1, pp. 195–202, 1983.

[14] K. Poern, *On empirical Bayesian inference applied to poisson probability models*. Sweden: Linkoeping Univ, 1990.

[15] S. Hora and R. Iman, "Bayesian modeling of initiating event frequencies at nuclear power plants1," *Risk Analysis*, vol. 10, pp. 103 – 109, 05 2006.

[16] S. Macenski, T. Foote, B. P. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Sci. Robotics*, vol. 7, no. 66, 2022.

[17] A. Dosovitskiy *et al.*, "CARLA: an open urban driving simulator," in *1st Annual Conference on Robot Learning, CoRL 2017, Mountain View, California, USA, November 13-15, 2017, Proceedings*, vol. 78 of *Proceedings of Machine Learning Research*, pp. 1–16, PMLR, 2017.

[18] "Road vehicles — Diagnostic communication over Controller Area Network (DoCAN) — Part 2: Transport protocol and network layer services," standard, International Organization for Standardization, Geneva, CH, 2016.