Secure Vehicle Software Updates: Requirements for a Reference Architecture

Citation for the original published paper (version of record):
Strandberg, K., Arnljung, U., Olovsson, T. et al (2023). Secure Vehicle Software Updates: Requirements for a Reference Architecture. IEEE Vehicular Technology Conference, 2023-June.
http://dx.doi.org/10.1109/VTC2023-Spring57618.2023.10199410

Secure Vehicle Software Updates: Requirements for a Reference Architecture

Kim Strandberg*†, Ulf Arnljung*, Tomas Olovsson†, Dennis Kengo Oka‡
*Volvo Car Corporation, Sweden {firstname.lastname}@volvocars.com
†Chalmers University of Technology, Sweden, {firstname.lastname}@chalmers.se
‡Synopsys, Japan, {dennis.kengo.oka}@synopsys.com

***Abstract*—A modern vehicle is no longer merely a transportation vessel. It has become a complex cyber-physical system containing over 100M lines of software code controlling various functionalities such as safety-critical steering, brake, and engine control. The amount of code is anticipated to rise to around 300M lines of code by 2030. Furthermore, even well-tested code will contain more than one bug per 1000 lines of code. Thus, it can be expected that there will be around 100k bugs in a modern vehicle and around 300k bugs in a few years, where some might have a safety-critical impact. Automotive companies are transforming into software companies with more software developed in-house. The ability to hastily and securely patch vulnerabilities has become vital and is a prerequisite when securing modern cars. The UN Regulation No. 156 and the ISO 24089 emphasize the ability to update vehicle software securely.**

Consequently, we focus on securing the vehicle software update process. Our contributions include defining an attacker model and general security requirements. We further map these requirements to common security goals and directives to ensure broad coverage. Additionally, we present UniSUF, a secure and versatile approach to vehicle software updates. We identify entities involved during vehicle software updates, perform a threat assessment, and map the identified threats to security goals and requirements. The results highlight a secure framework with high industrial relevance that can be used as a reference architecture to guide securing similar software update systems within automotive and related areas such as cyber-physical systems, internet-of-things, and smart cities.

***Index Terms*—vehicle security, vehicle resilience, vehicle software updates**

I. Introduction

The complexity of software within the automotive domain is increasing at a high pace, and as a result, the number of potential software bugs increases as well. Hence, software updates mitigating vulnerabilities need to be applied regularly. The automotive industry has requirements for numerous software deployment scenarios, such as over-the-air, in workshops, and in factories, with or without Internet access [1]. However, there is a risk that if the software update process is vulnerable, it might leverage the potential for malicious code reaching in-vehicle systems causing life-threatening hazards such as manipulated brakes, steering, and engine control. Therefore, UN Regulation No. 156 [2] and ISO 24089 [3] provide demands on secure vehicle software updates.

A vehicle is a distributed system with dependability and real-time requirements and can contain over 150 computers running various operative systems, further interconnected using many different communication buses and protocols. Thus, performing secure software updates to vehicles requires a rather complex approach.

The software update process can be divided into *data distribution* and *data execution*, where the former concerns the difficulties of securely distributing all needed data, such as software files, installation instructions (diagnostic commands), and cryptographic material, to the entity responsible for the installation process for all in-vehicle computers, i.e., Electronic Control Units (ECUs). The latter can be divided into pre-, peri- and post-state for the installation process. Pre-state refers to the preparation necessary to execute before the installation process can start, including a version control validation for all ECUs by comparing vehicle-unique software versions in a database with the actual vehicle and handling deviations. Additionally, it may be necessary to disable firewalls and Intrusion Detection/Prevention Systems to enable unlocking and placing ECUs in programming mode and verify that the vehicle is in a state that allows software updates, e.g., parking mode with a secure offline state.

The peri-state involves potential validation, decryption, and installing and transferring the new software to affected ECUs. Post-state is to perform the necessary validation of the update, such as securely creating installation reports and logs that may affect upcoming installation processes. For example, the installation report can include information about corrupt in-vehicle data, such as invalid cryptographic keys or faulty ECUs, which must be solved by downloading additional updates or replacing hardware. Alternatively, the installation report can serve as proof of a successful update.

Furthermore, every vehicle is unique and needs to have unique *data distribution* and *data execution*. Thus, a unique vehicle configuration, multiple software files for every ECU, many unique cryptographic keys, and ECU-specific diagnostic requests are required. For instance, special cryptographic keys to turn off security functionality that might otherwise block the installation process.

The ECU installation process typically uses Unified Diagnostics Services (UDS), a communication protocol specified in the ISO 14229-1 standard [4]. UDS is also used for various tasks such as reading out fault codes, activating or deactivating firewalls, changing operations mode (e.g., driving or passive), and testing functionality. Additionally, there are different security levels related to diagnostics. To execute at a particular level, the entity responsible for the installation needs access to the corresponding key to unlock this level. These keys need to be securely distributed and used securely in the execution. Securing the distribution, storage, pre-, peri-, and post-execution processes w.r.t. the infrastructure and all in-vehicle processes is challenging.

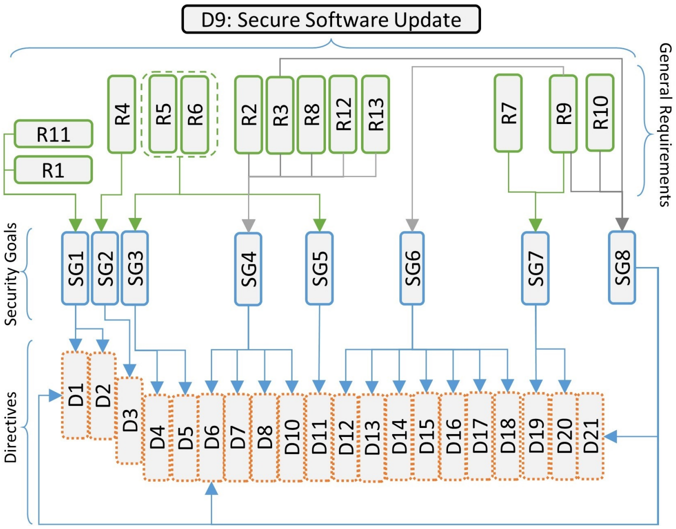


Fig. 1. A Goal Structuring Notation over Secure Vehicle Software Updates

TABLE I
General Requirements

Requirement R1: infrastructure and communication. The infrastructure, cryptographic algorithms, and key material shall follow best security practices. For instance, communication between backend entities shall encrypt communication and use proper authentication between entities. The same requirements shall be considered for in-vehicle entities directly related to the update framework.

Requirement R2: code review, testing and validation. When possible, external and internal code reviews shall be performed to detect vulnerabilities and deviations. Follow secure programming guidelines like the Secure Software Development Lifecycle (SSDLC). Continuous testing and validation shall be performed, such as positive/negative, vulnerability, fuzzing, penetration testing, and validation of, e.g., security controls.

Requirement R3: secure storage. Key management, such as generation and storage, shall be protected according to a high-security level within HSMs. Access to such should be highly restricted and only accessible to authorized users.

Requirement R4: redundancy. Relevant redundancy shall be used to switch to redundant entities during failures and compromises of entities or processes that are part of the update framework. For instance, traffic shall be redirected to redundant systems if appropriate during a denial of service attack.

Requirement R5: least privilege. Each backend entity shall be implemented according to the least privilege principle. i.e., each entity is responsible for securing its data (output), whereas data from other entities are validated and visible only on a need-to-know basis (input).

Requirement R6: separation of duties. The separation of duties shall be considered within the software update framework, where many separate entities shall be required to complete the distribution and execution processes.

Requirement R7: state awareness. Employed mechanisms and functions need to be robust against anomalies. The system shall be aware of its state and shall be able to switch to other states when anomalies are detected. For instance, when a cyber attack is detected, and if appropriate, the software updates system shall be able to abort, roll back and perform a retry from redundant entities.

Requirement R8: secure boot. State-of-the-art secure boot protection mechanisms shall be used where applicable, e.g., the installation responsible entity (the diagnostic client), including TEE environments. For example, the first image in a Chain of Trust has a ReadOnly Memory (ROM) and contains an immutable Hardware Trust Anchor (HTA), i.e., a Root of Trust code, including a root certificate. Hence, this image can be used to verify keys and signatures for upcoming loaded images such as TEEs.

Requirement R9: Intrusion Detection/Prevention Systems. IDSs/IPSs shall be used to detect and react to anomalies from normal communication patterns and known attacks.

Requirement R10: fault-tracing and forensics. Events related to software installation and security events (e.g., turning off the firewall) shall be securely stored to enable fault tracing and forensics investigations. For instance, traceability regarding time and source for events shall be possible.

Requirement R11: secure algorithms. The type and lifetime of key material concerning its context, future maintainability, and implementation shall be considered, e.g., using quantum-resistant algorithms and a set validity time for key material. However, the author intends not to recommend the type of algorithms to use.

Requirement R12: authenticity software. All encrypted software files shall be validated for authenticity before decryption by the installation responsible entity. Furthermore, supported end nodes shall perform another validation for decrypted software files. These two separate validation steps shall be based on different signing keys. The intermediate certificates shall be fetched, received, or pre-stored and always validated via OCSP or CRL requests and against a specific root certificate before being used.

Requirement R13: secure storage, freshness and authenticity within TEE. a) The private vehicle unique decryption key at the consumer (receiver) side shall be secured according to best security practices and only accessible inside the TEE. Furthermore, the public counterpart, i.e., the encryption key, shall be validated for authenticity before use towards Root CA, expiration date and revocation. b) The public key in R13a shall only be used to encrypt session keys. The validity time of these session keys shall be connected to the validity time of an update package. c) The input data (function calls) to TEE shall be validated for authenticity.

Contributions. We have identified general requirements to ensure a secure software update process. These requirements fulfill common security goals for cyber-secure vehicles. Moreover, we present a reference architecture named UniSUF based on previous work [1]. We validate the usability and security of our reference architecture by identifying an attacker model and performing a threat assessment. Finally, we identify mitigation mechanisms and map the specific threats to security goals and requirements to strengthen the robustness and design of UniSUF for a broad industry adaption with UN Regulation No. 156 in mind.

II. Attacker Model

There can be various threat actors, such as cyber terrorists, foreign countries, hacktivists, and vehicle owners [5]. However, we assume a common agenda where someone aims to manipulate the software update process or the software itself at any entity or during communication between entities throughout the software update process. For instance, the intent can be to recover and exploit secret signing or obtain encryption keys used during the software update process. The latter might enable disabling firewalls or switching ECUs into programming mode to enable update capabilities. Additionally, attackers might want to decrypt software files to reverse engineer and gain insight into its contents affecting the intellectual property and try to find vulnerabilities, e.g., through analysis of safety-critical systems. Thus, the attacker's ultimate goal is to exploit the software update system so that malicious or unauthorized software providing additional or altered functionality reaches the in-vehicle system, for instance, to gain and maintain remote persistence.

III. Methodology

Related work by us on the resilience of vehicles against security threats called Resilient Shield [5] provides a list of common security goals (SG) and directives (D), shown in

TABLE II
Mapping of Requirements to Security Goals, Directives and Threats

[Requirement]	[Security Goal]	[Directive]	[Threat]
[R1,R11]	[SG1: Secure Communication]	[D1,D2]	[P1-P6, R, C1-C3]
[R4]	[SG2: Readiness]	[D3]	[P1-P6, R]
[R5,R6]	[SG3: Separation of Duties]	[D4, D5]	[P1-P6, R, C1-C3]
[R2,R3,R8,R12,R13]	[SG4:Secure Software Techniques]	[D6-D8,D10]	[P1-P6,R,C1-C3]
[R5,R6]	[SG5: Separation/Segmentation]	[D11]	[P1-P6, R, C1-C3]
[R9]	[SG6: Attack Detection and Mitigation]	[D12-D18]	[P1-P6, R, C1-C3]
[R7,R9]	[SG7: State Awareness]	[D19,D20]	[P1-P6, R, C1-C3]
[R3,R9,R10]	[SG8: Forensics]	[D1,D6,D21]	[P1-P6, R, C1-C3]

Directives

D1:Authentication, D2:Encryption, D3:Redundancy/Diversity, D4:Access Control, D5:Runtime Enforcement, D6:Secure Storage, D7:Secure Boot, D8:Secure Programming, D10:Verification & Validation, D11:Separation, D12:Specification/ Anomaly-based Detection, D13:Prediction of Faults/Attacks, D14:Adaptive Response, D15:Reconfiguration, D16:Migration/Relocation, D17:Checkpoint & Rollback, D18:Rollforward Actions, D19:Self-X, D20:Robustness, D21:Forensics

Table II, developed from an analysis of cyber attacks on vehicles over a ten-year timespan to create a common baseline for a cyber-secure vehicle. We have identified general security requirements for software updates summarized in Table I and mapped them to the SG and D from Resilient Shield as security claims in Table II. We further use Goal Structuring Notation (GSN) [6] to present proofs for claims in a graphical manner to map these claims (i.e., SG and D) to the general requirements in Table I, as illustrated in Figure 1. Additionally, we map threats and elaborate on the fulfillment of requirements in Section IV-B. Thus, we achieve broad coverage by ensuring requirements covering established security goals and enhanced security by fulfilling these requirements. For instance, as shown in Figure 1 and Table II, SG1 is fulfilled by requirements R1 and R11 and reinforced by implementing the detailed directives D1 and D2, mitigating threats P1-P6, R, and C1-C3.

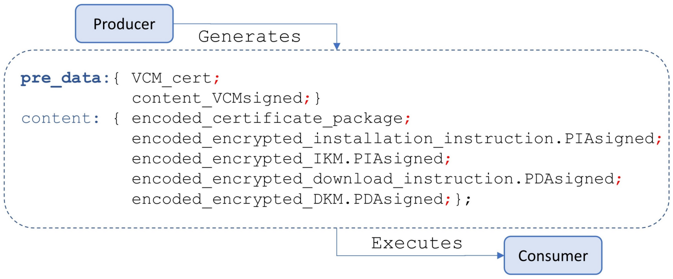
We further establish that the following security properties and principles are fulfilled by enforcing the following general requirements (cf. Table 1): *Confidentiality* by encryption and authorization [R1,R3,R5,R13]. *Integrity* and *authenticity* with hashes and signing [R1,R8,R12,R13]. *Authorization* and *isolation* between entities and their data by signing, containerization, virtualization, and trusted execution environment (TEEs) [R1,R3,R5,R13]. *Freshness* by a set validity time for the update package and associated session keys [R11-R13]. *Availability* and *reliability* using redundancy in the update system and support for implementing updates to the update framework when requirements change [R1-R13]. Principles such as the *least privilege* and *separation of duties* for entities are ensured on a need-to-know basis, and many separate entities are needed to complete vehicle updates [R5,R6]. Forensic capabilities with traceability by providing secure storage of events in logs [R10].

IV. A Reference Architecture for Secure Vehicle Software Updates

This section introduces a reference architecture named UniSUF [1] that fulfills the general requirements presented in Section III. We describe the reference architecture components, provide a threat assessment and elaborate on possible mitigations. UniSUF is made to cover the whole software deployment process, starting with securing the software update files to the installation process and finally creating post-process installation reports. UniSUF covers online and offline updates without dependencies on the data distribution model or the software update storage location. Additionally, it supports updating of 3rd party components, which is an increasingly important requirement. For instance, more companies are using vehicle platforms extended with 3rd party components and need to update such components ideally using the same software update framework.

Three primary entities are involved in the update process: the producer, the consumer, and the repository. The first entity produces the software, which includes the automaker and 3rd party suppliers. The second consumes or uses the software and comprises the vehicle and its users. The third entity is a storage for the software before installation, such as various cloud sources and local storage points, e.g., local network workshop drives.

In UniSUF, all data required for a unique vehicle software update is encapsulated using encryption and signatures into layers of data producing one single update file, the Vehicle Unique Update Package (VUUP). As shown in Figure 2, VUUP contains *pre_data* and *content*. The first part, i.e., pre-data, contains a validation certificate and a signed hash of the rest of the content of the VUUP file. The actual content includes a package containing the required certificates to validate all signatures in the VUUP file. Moreover, the content includes installation and download instructions and all the required cryptographic keys. However, due to the complexity and cost impact, modifying the E/E to all ECUs to accommodate a new specific software update framework is rarely an option. Instead, UniSUF allows securing the software update processes without adding new functions to all in-vehicle ECUs. Potential security mechanisms in use, such as secure boot and software signing, in individual ECUs remain intact. UniSUF uses isolation techniques such as containers, virtual machines, or combinations of such for producer entities. Depending on the context, producer entities shall be implemented as isolated modules with controlled and secure communication, secured on-premises or in the cloud. Thus, the communication shall be encrypted and authenticated. Each entity is responsible for its



```
pre_data:{ VCM_cert;
           content_VCMsigned;}
content: { encoded_certificate_package;
           encoded_encrypted_installation_instruction.PIAsigned;
           encoded_encrypted_IKM.PIAsigned;
           encoded_encrypted_download_instruction.PDAsigned;
           encoded_encrypted_DKM.PDAsigned;};
```

Fig. 2. The Vehicle Unique Update Package (adapted from [1])

TABLE III
ABBREVIATIONS

Producer Security Agent (PSA) handles cryptographic material in the backend systems.

Secure Key Generator (SKG) is used by PSA to secure key material generation.

Vehicle Identification Number (VIN) is a vehicle unique fingerprint, and is composed of 17 characters.

Version Control Manager (VCM) manages software versions related to unique vehicles and creates the software list and repackage data into the VUUP file.

Producer Download Agent (PDA) creates the instructions for downloading software for a specific VIN.

Producer Installation Agent (PIA) makes the diagnostic instructions for software installation for a certain VIN, including retrieving necessary cryptographic material.

VIN Database (VD) stores necessary VIN unique data related to software.

Cryptographic Material Storage (CMS) is a secure storage of cryptographic material.

Consumer Download Agent (CDA) executes download instructions and retrieves all necessary software files to local storage.

Consumer Installation Agent (CIA) is a diagnostic client responsible for running installation instructions and requesting the execution of needed cryptographic algorithms.

Consumer Security Agent (CSA) has a trusted execution environment (TEE) with pre-stored certificates between vehicle manufacturers and the CSA, enabling secure transfers and execution of cryptographic material.

data security, i.e., data is encrypted and signed at the source. Consequently, producer entities can validate the data. Still, only the intended consumer entity can read the information on a need-to-know basis via interaction with trusted applications running processes securely in isolation within a TEE on the consumer side.

A. Key management

UniSUF uses multiple signatures where producer entities have their specific signing keys. Entities are prevented from reading sensitive data created by other entities. Session keys are secured by encapsulation (i.e., asymmetric key wrapping) into layers within a final single file and transferred to the consumer. Encapsulation into layers implies that producer entities encrypt and sign their data (at the source). When data is retrieved, it is validated by the receiving producer entities that append their own encrypted and signed data. Finally, all information is collected and appended into a single VUUP file (cf. Figure 2 and 3). On the consumer side, the VUUP file is validated in its entirety, and further decapsulated, where after each internal component is validated in itself with its corresponding certificate. Policy-based keys are directed and bound to trusted applications by installation processes (pre-state) and isolated within the TEE. Thus, policies dictate how specific trusted applications executing in a trusted execution environment are allowed to use these keys. Decryption processes and keys are therefore isolated within these trusted applications and can be used for, e.g., decryption of software files, unlocking ECUs for software updates, and signing installation reports and logs.

B. Threat Assessment

This section lists the potential threats to UniSUF. The abbreviations used are listed in Table III. We look at threat probability and give examples of mitigating these threats. Security goals, directives, and requirements are considered for each threat, according to Table II. For instance, for threat R, SG4 and SG5 are most relevant, where R2 and R5 are mentioned as examples for mitigation. Although most requirements are applicable for all threats, there are some distinctions, e.g., R4 is typical for the producer and R13 for the consumer. We start with the producer, i.e., the backend side, then the repository, and end with the consumer, i.e., the vehicle side. We start with the perspective of isolated compromise, one entity at a time. Section IV-C, looks at a few examples of the consequences when the attacker has gained complete control over more than one isolated entity. Note that all attack vectors depend on the localization and implementation details. The probability is graded on a scale from low, medium and high and is based on the required time, expertise, tools, and proximity for the potential attack.

1) The Producer: We start with the perspective of isolated compromise on the backend side as visualized in Figure 3.

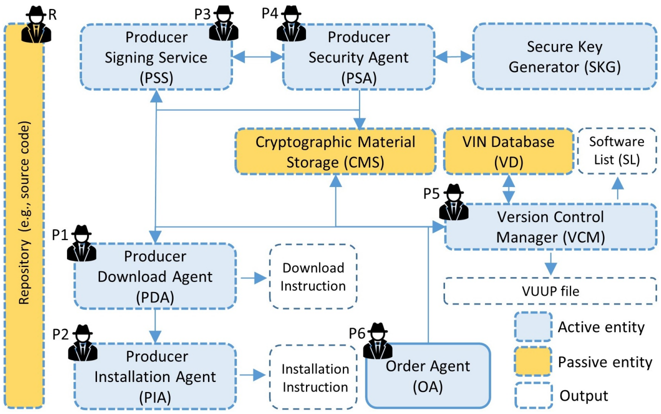

R
P3
P4
Producer Signing Service (PSS)
Producer Security Agent (PSA)
Secure Key Generator (SKG)
Repository (e.g., source code)
Cryptographic Material Storage (CMS)
VIN Database (VD)
Software List (SL)
P5
Version Control Manager (VCM)
P1
Producer Download Agent (PDA)
Download Instruction
VUUP file
P2
Producer Installation Agent (PIA)
Installation Instruction
P6
Order Agent (OA)
Active entity
Passive entity
Output

Fig. 3. Overview of the Producer and Repository threats

Threat P1: Producer Download Agent (PDA). *Location:* Within the automaker's internal infrastructure (on-premises) or via 3rd party suppliers, e.g., the cloud. *Functionality:* PDA receives a signed Software List (SL), validates the signature, and creates download instructions (URLs to software files). *Assumption:* PDA is compromised and under the complete control of the attacker. The attacker can manipulate the SL with instructions to download malicious data from an external source. Furthermore, the attacker can encrypt and sign the malicious download instructions with the correct keys. *Probability:* If PDA is secured according to the stated requirements, the probability of compromise is low. If located on-prem, it will likely be insider actors, and if located in the cloud, external attacks will likely be detected by intrusion detection systems. *Consequence and Mitigation:* Manipulated download instructions can be included in the VUUP file. Still, since all software files shall be encrypted and signed with keys not accessible to the PDA, the update process will abort and disrupt the software update process. A mismatch between the installation instructions and available software files will be detected. Redundant system shall be available to take over, e.g., when an attack is detected, switch to another server [R4].

Threat P2: Producer Installation Agent (PIA). *Location:* Same as in *Threat P1*. *Functionality:* PIA receives the SL, validates its authenticity, and creates installation instructions based on SL. PIA also receives all required cryptographic keys in an encrypted form. Thus, not readable to PIA. *Assumption:* The attacker controls the PIA and can manipulate the diagnostic instructions. For instance, decide what parts of the software files should be installed, creating inconsistencies in the vehicle software. Possibly, the attacker intends to keep known exploitable vulnerabilities from being updated. *Probability:* Same as in *Threat P1*. *Consequence and Mitigation:* The consequence can be severe, depending on the existing type of vulnerabilities. The results of inconsistencies in vehicle software are difficult to anticipate. Post-installation processes will detect inconsistencies via a complete vehicle software readout, and additional updates can be issued using a redundant, non-compromised system [R4]. A warning message shall be given about a potential malfunction and compromised PIA. The signed SL file can be included in the VUUP file, validated, and used for a consistency check before performing installation processes on the consumer side. The SL file is composed and signed by another entity, i.e., the VCM [R6].

Threat P3: Producer Signing Server (PSS). *Location:* Within automaker premises if possible. *Functionality:* PSS consists of three sub-entities isolated to the PDA, PIA, and the PSA. After a mutual authorization process, PDA, PIA, or PSA request signatures of hash values from PSS. *Assumption:* The complete PSS is under control by an attacker who can freely decline or accept signing requests and return invalid or valid signatures. *Probability:* Same as in *Threat P1 and P2*. *Consequence and Mitigation:* A disruption of the software update can occur. Invalid or non-existing signatures will abort the update process. If signing keys are compromised, these need to be revoked [R12]. Anomalies can be detected, and signing requests redirected to redundant systems [R4,R9].

Threat P4: Producer Security Agent (PSA). *Location:* Within automaker premises if possible. *Functionality:* PSA has access to the CMS and the SKG, where PSA retrieves and generates the required keys for the actual vehicle software installation process. *Assumption:* Secret keys are compromised and the threat actor has complete control over PSA, CMS, and SKG. *Probability:* Same as in *Threat P1-P3*. *Consequence and Mitigation:* The threat actor can disrupt the software update process by blocking key transferals or communicating faulty keys. IDS/IPS shall detect anomalies and react, e.g., switch to a redundant entity [R4,R9]. However, suppose the threat actor has physical access via the OBD-II port, debug ports, or connected directly to the communication bus. In that case, keys can be used to gain extended diagnostic privileges, for instance, turning off a firewall. We must then rely on end-nod security [R8, R12].

Threat P5: Version Control Manager (VCM). *Location:* Same as in *Threat P1-P4*. *Functionality:* VCM verifies the update request containing a fully signed software version readout from the vehicle, i.e., the current software composition for a unique vehicle. VCM retrieves the latest available software versions from the VIN database and creates a signed SL, which PDA, PIA, and PSA further process. Moreover, VCM validates input data from PDA and PIA and repackages the data into the VUUP file. *Assumption:* A threat actor can create an SL not corresponding with an approved combination of software versions and add faulty data into the VUUP file. *Probability:* Same as in *Threat P1-P4*. *Consequence and Mitigation:* Considering the amount of ECUs running various operating systems, aligned with the extensive amount of code, different software versions between ECUs might not be compatible and can cause unpredictable behavior. Thus, extensive testing is performed within the automotive industry towards different baselines, i.e., combinations of software versions [R2,R7].

A threat actor can prevent a specific software version with known vulnerabilities from being updated by stating that the ECU already has the correct software version installed. However, it will be detected by post-installation processes since a complete vehicle readout will be included in the logs and can thus be solved by issuing an additional update. Moreover, since VCM only repackage already signed data, any manipulation of such data will be detected [R5,R6]. If the issue remains, a redundant system shall be used, and a warning message of a potentially faulty VCM will be issued [R4]. Additionally, a consistency check towards an approved baseline can be performed by PDA, PIA, and PSA before the installation process and give a warning message when deviations between SL and an approved baseline are detected, and further allow or block the update depending on the detected deviations [R5,R6].

Threat P6: Order Agent (OA). *Location:* Same as in *Threat P1-P5*. *Functionality:* OA is responsible for managing the queue of vehicle software update requests. Verify that the update requests are authentic and initiate the update process via the VCM. *Assumption:* A threat actor has control over OA. *Probability:* Same as in *Threat P1-P5*. *Consequence and Mitigation:* Threat actors can block or allow update requests. Blocking updates can be performed by claiming that the signature validation of software update requests failed or supplying a malicious URL to a faulty VUPP file. In the first case, no VUUP file will be created, and in the second case, the VUUP file will be made. However, the signature validation of an incorrect VUUP file will fail, and the update will be aborted [R5,R6]. A warning message shall be issued when there are signs of update failures, whereafter, a redundant system can be used [R4].

2) The Repository: As shown in Figure 3, a passive entity that contains, e.g., source code.

Threat R: Repository, supply chain and insider threats. *Location:* 3rd party suppliers and within automakers premises. *Functionality:* Software controls various parts of the vehicle, including safety-critical systems. *Assumption:* Attacks/threats on the supplier and internal side, e.g., manipulation of source code and attacks on servers. *Probability:* Low for automakers' own developed software and medium for 3rd party suppliers due to the complexity of the supply chain and the limited potential for code reviews. *Consequence and Mitigation:* Consequences can be severe, depending on the code. For instance, code might affect safety-critical ECUs directly or indirectly via other ECUs not being satisfactorily isolated. Automakers and external suppliers shall limit access to codebase [R5] and perform software code reviews [R2]. Code validation processes shall be established through the supply chain and internal automaker processes.

3) The Consumer: As shown in Figure 4, the focus of this section is on the threats concerning an individual vehicle. The threats are still dependent on implementation details, such as the location of the various modules [1]. For instance, the diagnostic client, i.e., CIA, can be part of an ECU in the vehicle but also executed externally via a diagnostic update tool connected to the OBD-II port.

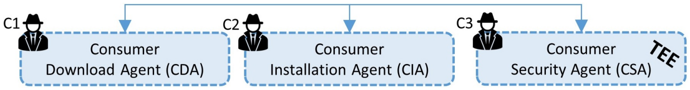

C1
Consumer Download Agent (CDA)
C2
Consumer Installation Agent (CIA)
C3
Consumer Security Agent (CSA)
TEE

Fig. 4. Overview of the Consumer Threats

Threat C1: Consumer Download Agent (CDA). *Location:* In vehicle or within an external tool. *Functionality:* CDA checks if updates are available and receive a signed URL to the unique VUUP file if that is the case. CDA then validates the URL's authenticity and downloads the VUUP file to local storage. Further, the VUUP file is validated and decapsulated. Whereafter all internal contents are validated with their respective certificate. The next step is for CDA to request the initiation of keys by the CSA. If successful, CDA can request a decryption process of the download instructions from CSA. *Assumption:* Threat actor has complete control of the CDA and can block the update process and manipulate the download URL to retrieve additional malicious files to local storage. *Probability:* We consider a medium probability for external tools and a low probability for in-vehicle implementations.

Consequence and Mitigation: From a CDA perspective, malicious data can only be downloaded, not executed. The attacker can allow CDA to finalize all steps with accurate data. In that case, still, the CIA will never execute any data from local storage without validating the signatures of the data. Secure boot protection mechanisms shall detect manipulation of CDA functionality [R8]. CDA and CIA can be integrated into the same ECU and be part of the same secure boot mechanisms. However, CDA can also be integrated into an external download tool, whereafter data is pushed to local vehicle storage, e.g., via mutual SSH [R1]. If the external CDA has been compromised, malicious data might reach local storage. Still, the CIA will not execute any data without the successful validation of signatures of the data [R6,R12]. Detected comprise of tools shall lead to revocation. Thus, mutual authentication shall fail for these devices concerning downloads from external sources and pushing data to the vehicle [R1,R11,R13].

Threat C2: Consumer Installation Agent (CIA).

Location: Same as C1. *Functionality:* The responsibility of the CIA is to execute installation instructions, thus installing decrypted and validated software files. Moreover, after the CIA successfully established a communication interface with the CSA, i.e., with the secure applications within a TEE, the CIA has the potential to send encrypted keys to unlock ECUs to deactivate firewalls, perform decryption of software and allow software updates. *Assumption:* A threat actor controls the CIA and might manipulate installation processes. *Probability: Same as C1. Consequence and Mitigation:* The decryption of keys only takes place inside the TEE; therefore, not visible to the CIA. However, the actual function calls to trusted applications might be manipulated, for instance, by switching encrypted keys between functions. Thus, function calls to the CSA shall be authenticated, e.g., with an authentication tag. We assume an isolated compromise; therefore, the CDA is still intact, and we only have authentic data at local storage [R13].

Threat C3: Consumer Security Agent (CSA). *Location:* Same as C1 and C2. Note that the VUUP file needs to be created for a particular VIN when located in an external tool. *Functionality:* CSA has a TEE and offers a secure execution of cryptographic mechanisms and data. *Assumption:* We assume a complete compromise of CSA. *Probability*: Low, since secured with, e.g., R3, R8 and R13. *Consequence and Mitigation:* Secret keys are compromised. Examples of usage include turning off firewalls, unlocking ECUs for software updates, and software decryption. A secure boot shall protect CSA, and thus, manipulations shall be detected [R3, R8, R13]. However, suppose a threat actor has gained physical control, and the decryption/execution of keys within the TEE is compromised. Keys can be used when connecting to a communication bus to gain extended privileges to ECUs.

However, many ECUs have inherent protection mechanisms such as secure boot and signed software [R8, R12]. Thus, malicious updates will not be approved even when valid diagnostic keys are used to put them in a state for software updates. However, a few legacy ECUs that do not fulfill security requirements might be vulnerable.

C. Examples of Multilevel Compromise

UniSUF is made to cause the least possible harm when entities are compromised. Each entity shall be implemented as a separate module with the potential to be localized differently (e.g., locally on-prem or containerized in the cloud), along with necessary redundancy. This section will take a few examples of when multiple entities are compromised.

1) The Producer: ***Threat P2, P3: PDA and PIA***. Assume that a threat actor has gained control over the download of data and the diagnostic instructions to install data. ECUs without end-node/secondary validation of signatures might be compromised in that case. However, the PSA is not under attacker control in this scenario. Thus, there will be deviations from the output data from PSA concerning the PDA and the PIA. The PDA, the PIA, and the PSA generate data based on the signed SL file. However, if an attacker can manipulate the normal process flow, e.g., perform ECUs unlock by using authentic data from PSA and then use malicious data for ECUs without end-node protection, these might be compromised [R8, R12]. Post-installation processes shall detect these deviations, and additional updates shall be issued from a redundant non-compromised system [R4,R9,R10].

Threat P2, P3, P5: PDA, PIA and PSA. In the previous multi-level compromise, PSA is not compromised. Thus, cryptographic keys can be sent and processed encrypted to TEE, e.g., to enable extended diagnostic privileges within manipulated installation instructions. Still, the keys are not exposed outside the TEE since they are encrypted. However, if PSA is compromised, we can also assume that secret keys are leaked because PSA can access CMS. In such a case, legacy ECUs without end-node protection are at risk by physical access, e.g., over the OBD-II port. PDA, PIA, and PSA shall all create their data based on the signed SL file. However, if all three entities are compromised, we can assume that the SL file is not enforced. Thus, additional data validation and comparison towards the SL file shall be made on the consumer side, and warning messages issued for deviations [R9,R10]. The primary mitigation is end-node protection, such as using secure boot and signed software [R8, R13]. Legacy systems not adhering to basic security requirements are always at risk.

2) The Consumer: We assume that CDA, CIA, and CSA are localized in the vehicle for this specific case. These can be physically or remotely compromised by, e.g., the driver with the intent to chiptune or by other threat actors to cause harm. Directly compromising entities not part of UniSUF is out of the scope. However, indirect compromises via UniSUF are considered.

Threat C1 and C2: CDA and CIA. A threat actor that has gained control of the CDA can control the process of downloading data. Still, the CSA is not exposed in this case. Thus, the threat actor can only interact with CSA using authentic data since validation of function calls and sent data detects manipulations [R13]. However, a threat actor could manipulate the instructions to download malicious data from other sources. In typical cases, the only way to initiate the installation process is to send authentic installation instructions to the diagnostic client, i.e., CIA [R6]. In this case, the CIA is also compromised but must still interact with CSA using authentic data to start the installation [R13]. As long as CSA is secured, the required keys to perform software installation via UniSUF are not exposed, and no installation can be completed using invalid data. Still, blocking the actual update might be possible [R4,R9].

Threat C1, C2, and C3: CDA, CIA and CSA. In this case, the threat actor has also managed to control CSA. If secret data is leaked from the TEE, such as cryptographic material, we can assume that these keys can be misused for one specific vehicle [R6]. For instance, decrypt software files, disable firewalls and unlock ECUs to enable software updates. Mitigations for these cases are based on end-node protection mechanisms, such as an additional layer for signed software and secure boot [R8, R12]. Critical ECUs, e.g., safety-related, shall always use end-node security.

D. Comparison to other approaches

Previous solutions such as [7]–[11] lack a unified approach for the various software update scenarios required within the automotive. Moreover, they do not consider vehicles needing unique updates regarding specific configurations, installed software versions, and unique cryptographic keys. These solutions are missing necessary details, e.g., installation instructions such as handling pre-, peri- and post-state diagnostics, secure transport, and secure execution of ECU-specific cryptographic keys. They also consider changes to all in-vehicle ECUs, which usually is not feasible. UniSUF aims to fill these gaps by proposing a secure and versatile software update framework with previously mentioned considerations in mind.

V. Conclusion

Modern vehicles are complex systems containing more than 100M lines of software code controlling various functionality including safety-critical functions and get increasingly vulnerable when adding connectivity. Thus, ensuring hastily and secure software updates to patch vulnerabilities is imperative. We have introduced UniSUF, identified entities involved in the distribution and execution during vehicle software updates, provided an attacker model, performed a threat assessment, and elaborated on mitigation mechanisms. We have identified general security requirements for vehicle software updates and mapped them to common security goals and directives further visualized with the Goal Structuring Notation (GSN). The results show that UniSUF fulfills the stated security goals and provides a secure and unified vehicle software update framework that can serve as a detailed reference architecture. We believe our results are valuable not only for automotive software update architects. We also see high relevance for engineers in related areas, such as cyber-physical systems, internet-of-things, and smart cities, guiding the design of secure software update solutions.

References

[1] K. Strandberg, D. K. Oka, and T. Olovsson, "UniSUF: a unified software update framework for vehicles utilizing isolation techniques and trusted execution environments," 2021.

[2] United Nations Economic Commission for Europe (UNECE), "Un regulation no. 156 - software update and software update management system," 2022.

[3] "ISO 24089, Road vehicles - Software update engineering," International Organization for Standardization (ISO), Tech. Rep., 2023.

[4] "ISO14229, Road vehicles - Unified Diagnostic Services (UDS)," International Organization for Standardization (ISO), Tech. Rep., 2020.

[5] K. Strandberg, T. Rosenstatter, R. Jolak, N. Nowdehi, and T. Olovsson, "Resilient shield: Reinforcing the resilience of vehicles against security threats," 04 2021, pp. 1–7.

[6] T. Kelly and R. Weaver, "The goal structuring notation–a safety argument notation," *Proc Dependable Syst Networks Workshop Assurance Cases*, 01 2004.

[7] S. Mahmud, S. Shanker, and I. Hossain, "Secure software upload in an intelligent vehicle via wireless communication links," in *IEEE Proceedings. Intelligent Vehicles Symposium, 2005.*, 2005, pp. 588–593.

[8] S. Idrees, H. Schweppe, Y. Roudier, M. Wolf, D. Scheuermann, and O. Henniger, "Secure automotive on-board protocols: A case of over-the-air firmware updates," 03 2011, pp. 224–238.

[9] D. K. Nilsson and U. E. Larson, "Secure firmware updates over the air in intelligent vehicles," in *ICC Workshops - 2008 IEEE International Conference on Communications Workshops*, 2008, pp. 380–384.

[10] J. Samuel, N. Mathewson, J. Cappos, and R. Dingledine, "Survivable key compromise in software update systems," 12 2010, pp. 61–72.

[11] T. Karthik, Kuppusamy, and D. McCoy, "Uptane : Securing software updates for automobiles," 2016.