Securing Automotive Software Supply Chains

Marina Moore
New York University
marinamoore@nyu.edu

Aditya Sirish A Yelgundhalli
New York University
aditya.sirish@nyu.edu

Justin Cappos
New York University
jcappos@nyu.edu

***Abstract*—Software supply chain attacks are a major concern and need to be addressed by every organization, including automakers. While there are many effective technologies in both the software delivery and broader software supply chain security space, combining these technologies presents challenges specific to automotive applications. We explore the trust boundaries between the software supply chain and software delivery systems to determine where verification of software supply chain metadata should occur, how to establish a root of trust, and how supply chain policy can be distributed. Using this exploration, we design Scudo, a secure combination of software over the air and software supply chain security technologies. We show that adding full verification of software supply chain metadata on-vehicle is not only inefficient, but is also largely unnecessary for security with multiple points of repository-side verification.**

In addition, this paper describes a secure instantiation of Scudo, which integrates Uptane, a state of the art software update security solution, and in-toto, a comprehensive supply chain security framework. A practical deployment has shown that Scudo provides robust software supply chain protections. The client side power and processing costs are negligible, with the updated metadata comprising 0.504% of the total update transmission. The client side verification adds 0.21 seconds to the total update flow. This demonstrates that Scudo is easy to deploy in ways that can efficiently and effectively catch software supply chain attacks.

I. Introduction

There has been a massive increase in software supply chain attacks recently [3], [21], [7], [32]. These attacks target the software development process and missing validation of artifacts used in this process. In response to these attacks, President Biden issued an Executive Order that requires the use of supply chain metadata [1]. Other nations are also developing regulations to reduce such supply chain threats [25]. In addition to government regulations, several technologies have been proposed to address this threat. These technologies each solve pieces of the broader software supply chain security problem [30], [20], [8], [33], [24], [35], [38]. Such measures provide the devices installing software added assurance that the software was correctly made. After the software is made, there are several technologies that allow secure software over-the-air (SOTA) to update or distribute software on a vehicle [34], [14], [37], [15], [12], [36], [19].

However, the combination of software supply chain technologies with SOTA technologies has not been explored, and combining them in a way that maintains their security properties and is efficient has some added complexity. Specifically, there is no existing solution for secure distribution of the software supply chain root of trust (or why the software supply chain security system is trusted), dissemination of security policy (often from multiple supply chain vendors), all while reconciling limited storage and runtime capabilities of automobile Electronic Control Units (ECUs).

First, the root of trust for a secure software supply chain system needs to be securely distributed to vehicles. Software supply chain systems and SOTA systems each have a root of trust that is used in verification. While using one root of trust to distribute the other simplifies verification, it introduces a risk that the distribution of the root of trust could be compromised. This is especially true if the root of trust uses online keys, which are more likely to be compromised.

In addition to the roots of trust, software supply chain policy also needs to be distributed to verifiers. This policy needs to specify which actors were supposed to perform which actions in the supply chain, so that this information can be used in verification. As automotive supply chains are complex with many suppliers contributing to a single update, this policy may involve verification of supplier-sourced updates. Also, there may be different policy to verify depending on where the verification occurs. For example, verification later in the process could additionally validate that previous verifiers signed off on the update. It is also vital to securely establish which actors are allowed to define different policies. All of this policy needs to be created by a trusted party and securely distributed to the verifier to prevent tampering.

Finally, the verification of metadata required by software supply chain security solutions requires additional storage and runtime on-device. For many automotive ECUs, this makes use of these technologies impossible on devices with limited capabilities.

In this work, we model the automotive software supply chain, from the production of the software all the way through its distribution. We enumerate the actors involved in each part of this pipeline, and we describe the capabilities they must be granted for the tasks they are responsible for.

We present Scudo, an abstract framework that builds on this model to describe how software supply chain technologies can be securely composed with automotive SOTA technologies taking into account the unique constraints faced by automotive software and hardware. Scudo proposes utilizing secure SOTA systems to perform much of the software supply chain verification, transmitting cryptographically verifiable attestations to

Symposium on Vehicles Security and Privacy (VehicleSec) 2024
26 February 2024, San Diego, CA, USA
ISBN 979-8-9894372-7-6
https://dx.doi.org/10.14722/vehiclesec.2024.23015
www.ndss-symposium.org

the vehicle to guarantee the verification occurred. By performing this verification in three, isolated environments, we achieve the same compromise-resilience as on-vehicle software supply chain verification while minimizing the on-vehicle overhead. We further define security properties needed to distribute the supply chain root of trust and policy via the SOTA system including the use of offline keys, protection from rollback attacks, and a system of delegation. These properties ensure both compromise resilience and per-image, up-to-date policy. Scudo also describes how the various actors that make up the supply chain must be classified, and the trust boundaries each actor is responsible for.

Finally, we describe a practical implementation of Scudo that uses Uptane [37], [14], [15], [18], a popular secure SOTA system, and in-toto [35], [10], a software supply chain security framework. We collaborated with Toradex, an Internet of Things company, to evaluate this implementation of Scudo. Toradex manufactures embedded devices that are deployed in key applications such as medical devices, industrial human machine interfaces, and gateways. The company also develops Torizon, an OS platform, for their boards that customers use to deploy their applications. As Toradex were prior adopters of Uptane, we were also able to determine if Scudo can be integrated into existing systems. We believe this model also applies well to automotive use cases, with ECUs having similar capabilities to the Toradex embedded devices. We used the metadata captured by Scudo when building Torizon to measure the overhead imposed by Scudo in comparison with just using Uptane. We found that with Scudo, the updated metadata retrieved by a device comprises 0.504% of the total transmission as the change in metadata size is dwarfed by the size of the image itself. Scudo also adds extra verification steps at various points in the repository. Each of these checks incurs a one-time cost of an average of 1.33s per image. On the vehicle itself, Scudo adds limited verification that ensures the repository side verifiers all approved the image. This verification has an overhead of 0.21s. These findings demonstrate the practicality of Scudo.

In summary, our contributions are:

- We analyze the challenges of combining SOTA technologies with software supply chain security technologies in the automotive domain.
- We introduce Scudo, a secure system for combining a SOTA system with a software supply chain security system by taking advantages of specific security properties of each.
- We implement Scudo using Uptane and in-toto, and evaluate this implementation with Toradex, determining that Scudo provides a manageable overhead on-vehicle while achieving the security of end-to-end software supply chain verification.

II. Understanding Software Supply Chain Security

To better understand how to secure the software supply chain, this section discusses the common components used and then discusses issues that arise when composing them. To keep this section general, the components and their properties are described in an abstract manner.

A. Software Supply Chain Security systems

A software supply chain is a collection of systems, devices, and people which produce a final software product [22]. Any defense mechanism that protects a software supply chain, thus, encompasses the supply chain itself as well as the people responsible for defining the security policies for the defense mechanism. This allows us to derive, broadly speaking, two groups of actors for the production of automotive software: **software supply chain owners** who are responsible for defining security policies (and therefore the root of trust for the software supply chain security technology), and **software supply chain functionaries** who perform the necessary operations to produce the software, such as testing and building.

As each software supply chain functionary performs operations, they record verifiable metadata about these steps and who performed them. This metadata can capture information about the build process [30], [20], [8], [35], dependencies [35], [33], [24], version control information [35], [30], and other holistic information [35] about the software supply chain. All of this metadata is then used to ensure that the policy set by software supply chain owners was followed.

Verification of software supply chain metadata varies greatly by system. Today, in most cases, the information gathered is not independently verifiable by external parties except to verify that the software supply chain system has signed to indicate it is correct [30], [8], [33], [24]. However, systems do exist which enforce policies over the generated supply chain security metadata [35], and others are adding this functionality now [30], [31].

As a result, there are a few key properties to consider with software supply chain security systems:

1) *root of trust.* How should the root of trust, i.e. the mechanism used to establish trust for the software supply chain system, for the software supply chain security technology be managed? Should it be burnt into the system at manufacture time or established by the SOTA system? The root of trust defines the trusted keys and policy for the software supply chain system, and so a compromised or mis-distributed root of trust can negate the security of the system.
2) *trusted components and actors.* Which software supply chain functionaries are trusted and for what aspects of security? Are the keys associated with these systems or actors able to be better secured than they were in a pure SOTA setup? Is the impact of a compromise lessened in some way?
3) *verification.* Where should software supply chain verification be performed and how does this impact security and efficiency? For efficiency, is it possible to avoid transmitting some information to vehicles while retaining security?

The answers to these questions are paramount to understanding what composition of SOTA and software supply chain technologies makes sense.

B. SOTA systems

After a software artifact is produced, it must be distributed securely to consumers. Software Over the Air (SOTA) systems deliver software updates to vehicles [34], [15], [37], [14], [18], [12], [19]. Their goal is to allow a manufacturer to securely update vehicles in the field, without requiring that they be returned to a dealership. In the automotive context, the consumers are one or more ECUs on a vehicle. Each software artifact is stored in a repository, from which it is distributed to automobiles. Like before, this allows us to derive groups of actors who handle the distribution of software: **repository owners** who are ultimately responsible for the software repository, **artifact uploaders** who upload built artifacts to the repository, and **repository directors** who dictate which software artifacts must be distributed to specific vehicles.

Note that some of these groups may overlap in their personnel or systems. For example, a software supply chain functionary, having built some piece of software, is typically tasked with uploading it to the software repository, meaning they also serve as an artifact uploader.

SOTA systems often have the following properties:

1) *root of trust.* ECUs which work with a SOTA system typically have a root of trust loaded into them at manufacture time that corresponds to the repository owners [12], [15]. All trust in the system derives from these keys and so their security is paramount.
2) *key management.* The SOTA system in many cases includes mechanisms to rotate and revoke keys for both repository directors and artifact uploaders [15]. This is essential to make a system resilient against attacks long term and to make key management feasible and secure.
3) *update ingestion.* The system has a means to determine the validity of an update and decide whether it should be provided to ECUs. This is an opportunity for verification that is typically done via a signature on either an update or the update's metadata.
4) *update selection.* The repository director (typically a manufacturer) decides which update should be applied to a specific ECU in a vehicle. This is another opportunity for update verification.

An attacker who is able to influence or attack any of these steps can cause substantial harm in practice.

C. Vehicle

Finally, the last actor involved in this chain from the production of software to its consumption is the vehicle itself. The vehicle's ECU(s) must fetch and install new software when directed to do so by the repository directors. The vehicle is responsible for SOTA verification, and in some cases may be expected to perform software supply chain verification. It is important to note that most vehicle ECUs have limited memory and processing capabilities, and anything sent to an ECU over-the-air may travel over a cellular network or other bandwidth-constrained network. Due to these realities, efficient on-device performance is critical.

III. Threat Model

We consider the following actors in our system:

- **Software supply chain functionaries:** The entities performing, and attesting to, each step in the software supply chain. Some functionaries who also submit the software for ingestion at the end of the supply chain will act as artifact uploaders.
- **Software supply chain owners:** The entity setting policy for the software supply chain including which steps should be performed and which functionaries should perform them.
- **Artifact uploaders:** The entity that uploads software to a repository.
- **Software update repository:** The repository that contains software updates to be sent to the vehicle.
- **Software repository owners:** The entity establishing the software repository that distributes artifacts to vehicles.
- **Software repository director:** The entity determining which software updates should be sent to each vehicle and ECU.
- **SOTA gateway:** The gateway that sends updates and metadata to the vehicle over the air.
- **Software supply chain root of trust:** The root of trust for the software supply chain that indicates software supply chain owners. This root of trust must be provided to supply chain verifiers in a secure manner. The keys used to sign the root of trust are typically stored offline and used rarely to minimize the chances of compromise.
- **Software update root of trust:** The root of trust for the software update system that indicates the trusted software update repository and director. This root of trust is included on device ECUs at manufacture time and is managed by the software repository owners. As with the software supply chain root of trust, the keys used to sign this root of trust are stored offline and used rarely to minimize the chances of compromise.
- **Vehicle ECUs:** The devices that will receive and verify software updates.

We assume that an attacker can do the following:

- Compromise one or more keys including keys on the software update repository or keys controlled by the supply chain functionaries, supply chain owners, or software update director.
- Compromise one or more supply chain functionaries.
- Compromise the software update repository, the software update director, or the SOTA gateway (but not all three).

The following are out of scope:

- Complete compromise of software update system's root of trust, which is backed by multiple, offline keys.
- Compromise of any ECU on the vehicle before the software update, including with physical access.

In this environment, we aim to ensure the integrity of the supply chain producing an artifact and the installation of the intended software update. This means that all steps in the software supply chain were performed as described by the software supply chain owners, and the correct update is installed on the vehicle as described by the software update director. As different elements of the system are compromised, our goal is a graceful degradation of security properties.

IV. Scudo: Composing SOTA and Software Supply Chain Technologies

We present a sketch of our design, Scudo, that addresses the threat model in section III. This section first explores a simple combination of SOTA and software supply chain security technologies, then examines how Scudo is an improvement. Some of these improvements may require different properties of the SOTA and software supply chain technology systems which are being utilized. The key insight behind our design is that a sufficiently robust SOTA system can perform verification of software supply chain metadata at multiple points, then send proof of this verification to the vehicle. This reduces the overhead on the vehicle while still performing all verification. Later, we will present a concrete architecture for this design in section V.

A. Software supply chain root of trust

How should the root of trust for the software supply chain security technology be managed? Should it be burnt into the system at manufacture time or established by the SOTA system?

The simple way to set up the root of trust for the software supply chain security technology is to burn the root of trust into the automobile ECU at manufacture time. During regular operations, the ECU receives updates and supply chain information, which is verified against the root of trust. However, this technique has disadvantages. Key rotation is a necessary and important hygiene practice for reasons including later discoveries of weak key generation [5], algorithm weakness [11], accidental key exposure [9], or things like the emergence of quantum algorithms for cracking existing systems. Given the long lifespan of vehicles, some occurrence of this type is fairly likely and needs to be part of an organization's plan. The ability to rotate even the root of trust's keys also furthers *compromise resilience*, or the ability to recover from a compromise of even the software supply chain system's root of trust keys.

The alternative is to use a secure SOTA system to distribute the root of trust. Here, a secure SOTA system is one that has strong protections in update selection (avoiding rollback attacks where the SOTA system presents an older version of an artifact in place of the latest) [29], [16], a system of delegation [17], and the ability to manage its own root of trust. We observe that distributing the supply chain security system's root of trust is fundamentally an artifact distribution problem as the root of trust is another artifact sent to the vehicle. The SOTA system is designed expressly for that, as opposed to the supply chain security system, thus we assign this responsibility to the SOTA system.

Scudo selects a compromise resilient SOTA system and combines the roots of trust for the SOTA and software supply chain systems. The SOTA system can use the (offline) SOTA root of trust to delegate to the supply chain root of trust. Therefore, the security of the supply chain root of trust is equivalent to that of the SOTA root of trust, which can update the delegation if the supply chain root of trust in the event of a compromise.

In addition to the compromise resilience properties, Scudo requires the SOTA system to be deployed with *offline keys* for the root of trust. An offline key such as a hardware token, will not be compromised if the repository is compromised. This allows for *secure recovery* if any other element of the system is compromised. A SOTA system that uses an online root of trust should not be used to distribute a software supply chain root of trust as such keys are more likely to be compromised.

B. Verification

Where should software supply chain verification be performed and how does this impact security and efficiency? For efficiency, is it possible to avoid transmitting some of this information to vehicles while retaining security?

There are many places where one could validate the software supply chain metadata. This includes before ingestion to the SOTA system, before selecting the update to be installed on a vehicle, before transmitting the update to a vehicle, on the vehicle ECU which receives the transmission, and on the vehicle ECU which the update is destined for. A natural question is how different choices for the verification location impact the security and efficiency of the system.

One key point to note is that for a single update one could decide to perform verification in *all* of these locations. There is no harm (at least from a security standpoint) to verifying the same metadata in multiple places. However, each point of verification adds a performance cost. So, where is it sensible to verify metadata, in what cases, and why?

The obvious point of verification is the vehicle, as it is ultimately the consumer of the software. This has the benefit of full end-to-end verification. However, secure SOTA systems [37] have previously found that verification on the vehicle is not viable given resource constraints, and they have had to settle for reduced verification workflows without all the same guarantees.

Scudo mitigates the resource constraint issue by a combination of two features that allow most software supply chain verification to move off device with minimal impact on security. First, Scudo adds software supply chain verification at multiple, disparate points at the repository, similar to the strategy proposed by the Reproducible Builds project [28]. Each verifier must be hosted on a distinct server to ensure that the compromise of one does not result in the compromise of the others. Second, Scudo captures evidence of all of these verifications, known as verification summaries, which are then sent to the automobile. The summaries are signed by the verifier and attest that verification happened at each point on the repository without imposing the full resource overhead of the software supply chain security metadata on-vehicle. In this way, Scudo ensures end-to-end verification with the caveat that it relies on there being at least one honest verifier at the repository. As long as this one honest verifier exists, this strategy is equivalent to doing software supply chain verification on-vehicle.

While these factors help with the resource constraint issue, they also enable additional checks that ensure verifiers are behaving correctly. In addition to verifying the full software supply chain metadata for an artifact, each repository-side verifier can also validate the verification summaries of the verifiers that come before it. Thus, a verifier can detect other misbehaving verifiers early.

But, how many points of verification are sufficient to ensure that the software supply chain policy was met? To avoid a single point of failure (i.e., a solitary verifier), Scudo requires at least two distinct verifiers at the repository. Obviously, the more distinct verifiers there are, the more secure the system is. Scudo does not pick a specific number, but instead the design adds verification at several logical points: at update ingestion into the SOTA system, at update selection by the software repository director, and at update distribution by the SOTA gateway. Any additional verifiers only strengthen Scudo's compromise resilience, but each verifier adds complexity and cost both at the repository and on the vehicle (as it must validate more verification summaries).

In summary, out of the box, Scudo ensures that the supply chain security metadata is verified at three points at the repository. Each of these verifiers generate authenticated summaries of the verification. In addition to the image itself, the vehicle receives all of the verification summaries, which are validated by the vehicle's policy to verify the result of repository-side verification. This vastly reduces the metadata sent to the vehicle, while still ensuring end-to-end verification.

C. Software supply chain policy

Who defines policy for the software supply chain? Is this policy the same for all verifiers? How is policy distributed?

Software supply chain security technologies require verification, and this verification must follow a specific policy. This policy could define software supply chain functionaries, what these steps these functionaries should perform, where they must be performed, and even ensure that artifacts are not altered between steps.

However, this policy can be very complex, especially in an automotive environment with multiple vendors, each responsible for some steps in the supply chain. A tier 1 supplier for a particular ECU may create a software update for that ECU, which is verified by the manufacturer before being transmitted to the vehicle alongside other updates. This tier 1 supplier may even have their own sub-suppliers (tier 2, etc), each with a different software supply chain. The policies for each supplier need to be vetted and combined into a single, comprehensive policy that can be checked at verification time.

Further, as Scudo has multiple points of verification, we need to support distinct policies for each verifier. This is because each verifier is responsible for verifying the software supply chain security metadata **and** the integrity of the verifiers that come before it. Thus, the policy that will be verified is unique to both the supply chain for a particular image, and the point at which verification occurs. This also enables specifying policies that may only need to be applied at a particular verifier, without affecting the others.

All of these factors complicate the distribution of policy. Any malicious alterations to the policy has implications for the effectiveness of the supply chain security technology. Thus, the secure distribution of policy is paramount.

In Scudo, we use delegation from the SOTA system to tie different policies to specific images for each point of verification. Software supply chain owners must define different policies for each of the SOTA points of verification, and vehicle verification for each image. So, in a deployment of Scudo with the three repository-side verifiers detailed above, the supply chain owner defines a policy for each of those verifiers. In addition, the owner creates a policy for the vehicle that consumes the summary of each of the repository-side verifiers. These policies and summaries are associated with the image using the SOTA system's delegations.

D. Summary

In summary, Scudo combines a secure SOTA system and a software supply chain security system while balancing the unique constraints of the automotive ecosystem. First, Scudo recognizes the problem of distributing the root of trust for the supply chain security system is fundamentally that of software distribution, and leverages a secure SOTA system as a solution. Next, it allows for more efficient on-vehicle verification by requiring multiple repository-side points of verification, and enforcing this requirement on the vehicle. Finally, Scudo leverages the delegation capability of secure SOTA systems to manage multiple vendor and verifier specific policies for a single image.

V. Implementing Scudo

In this section, we describe the architecture of a real world implementation of Scudo. We discuss our selection of a secure SOTA system and software supply chain security system. We then describe how this implementation achieves the design of Scudo described in section IV. Figure 1 presents the architecture of this Scudo implementation. The analysis of Scudo's security (Section VI) and overhead (Section VIII) are deferred to later in the paper.

A. Components

For a real world implementation of Scudo, we selected the Uptane Standard [14], a widely deployed robust SOTA system, and in-toto [35], a state of the art comprehensive software supply chain security technology that is part of many other popular frameworks like SLSA [30] and Sigstore [20]. Parts of these technologies that are relevant to their integration are discussed briefly here but for brevity, detailed information about the projects are left to the project documentation [14], [37], [18], [35], [10]. Implementation of Scudo is not limited to these technologies, but can be done with any system that has the properties discussed in section IV. Some other technologies are discussed in section IX. These technologies were selected for our implementation based on their current adoption and security properties.

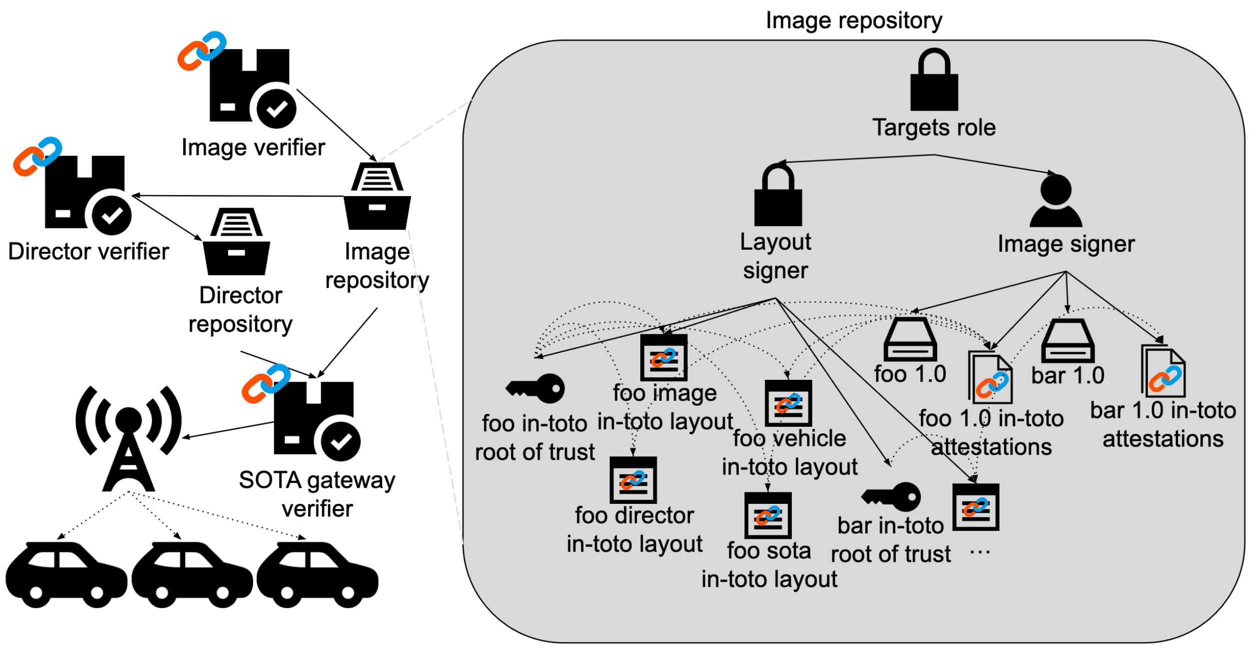

Fig. 1. An overview of Scudo's design. Scudo stores in-toto metadata on the Image repository using secure roles for the in-toto root of trust, layouts, and attestations. Scudo then introduces Image and Director verifiers that perform in-toto verification before images are ingested by the repositories. Then, in-toto verification is performed before metadata is sent from the Image repository to a vehicle by the SOTA Gateway. Finally, the vehicle performs verification of the summary attestations from each of the previous verifiers. The components that store or verify in-toto metadata are identified using its logo, a link in a chain.

1) Uptane: Uptane is a well established SOTA system used in automotive and Internet-of-Things (IoT) contexts that has a few key properties. Uptane asserts that a specific image is trusted to be installed on an ECU via the use of a type of signed metadata called targets metadata. This provides a means to attest to update ingestion and also that an update has been selected. However, to provide better compromise resilience, update ingestion and update selection are provided by different repositories, the Image repository and Director repository, respectively. Update ingestion from artifact uploaders is done using a key that is kept offline (not stored on a server) to minimize the damage potential in the event of a compromise. Both the Image and Directory repository have offline root keys that form their root of trust. These separate repositories will form two of our points of verification in Scudo.

Note also, that targets metadata is flexible and can be annotated to indicate further information or restrictions about an image. This may include what models of ECUs an image can be installed on, what other keys should be trusted to provide information about an image through *delegation*, where a specific image must be installed, and so on. This feature, especially the use of targeted delegation, will be used by Scudo to specify the in-toto root of trust and policy for a specific image.

Highly relevant to Scudo's security properties is the fact that Uptane provides strong timeliness and rollback protection [16], due to the way in which roles in Uptane sign metadata. Every time a vehicle checks to see if new updates exist, it receives metadata that makes a future attacker unable to convince ECUs to trust older images or metadata than what exists on the repository at this point. The mechanism by which this works are unchanged in Scudo and so interested readers should read the Uptane paper for details [14].

2) in-toto: in-toto [35] is a software supply chain security framework with a few security properties which are relevant to Scudo. To specify the sets of steps which should occur when making software, in-toto uses a metadata file called a *layout*. The "root layout" corresponds to the primary supply chain policy file and is also the supply chain root of trust as it is issued and signed by the software supply chain owners. The layout specifies policy about which entities are authorized to perform an step, what their corresponding keys are, and how this step relates to other steps (for example if the output from one step should be the input to another). For each step, one or more authorized software supply chain functionaries create and sign in-toto *attestations* that record attributes about the step such as the artifacts used and the environment where the step was executed. Finally, any party can use an in-toto layout and the corresponding attestations to validate the steps were correctly followed by the appropriate parties. Even in the case of a compromise or malicious interference, in-toto provides strong resilience and assurance properties. This is all unchanged from the typical use, implementation, and deployment of in-toto [35].

3) Implementing our model using in-toto and Uptane: The model of the software supply chain and distribution processes can be implemented using in-toto and Uptane. We discuss how the concepts discussed above can be applied to these technologies.

in-toto perceives the software supply chain as a series of steps to be performed by one or more actors. This aligns with the **software supply chain functionaries** role. in-toto also defines a policy known as a "layout", that must be issued by the **software supply chain owners**.

The mapping of the repository-side actors declared in section II map less cleanly to Uptane in comparison to in-toto. For starters, Uptane defines two separate repositories that together handle the tasks assigned to each group of actors. Each has a collection of **repository owners** who are ultimately trusted for that repository. They are the holders of the repository's root keys. The SOTA root of trust consists of both sets of root keys. **Artifact uploaders** are those that can upload an artifact to the repositories. Typically, this overlaps with a software supply chain functionary. Uptane defines two repositories precisely to direct software updates. The **repository director** role is backed by the "director" repository. This repository chooses specific updates for each vehicle.

B. Establishing the root of trust and policy for in-toto using Uptane

Scudo uses Uptane's targets metadata to specify the root of trust and policy for in-toto. This is done by having the image signer role, which previously handled image ingestion, additionally track in-toto attestation information. It also involves the creation of a new Uptane targets role, the layout signer, which specifies the root of trust and the in-toto layout, i.e. supply chain policy, for the images. Each of these are described in more detail now.

Layout signer role. The layout signer role, managed by the software supply chain owners, establishes in-toto's layout and root of trust. The layout and the root of trust, which includes one or more signing keys, are recorded as artifacts in the image repository. Further, the role associates each layout with its root of trust and ties the layout to particular images and ECUs. For example, in Figure 1, the foo in-toto layout can be specified to all `foo-*` images. Similarly, the foo root of trust is specified there and associated with the foo in-toto layout. Therefore, a verifier uses the layout signer role to identify the correct layout to use for an image and the layout's root of trust.

The layout signing key is only used when a layout or its root of trust changes. Given the importance of this role and its infrequent use, in practice this role's key is protected using measures like requiring a threshold of multiple physically held cryptographic keys stored in separate geographic locations.

Image signer role. The image signer role, on the other hand, is used whenever an image is ingested to sign metadata about the image and attach in-toto attestation information. As such, it is controlled by the artifact uploaders. This role from Uptane is adapted so that in addition to this role's metadata listing every image in the repository, it now also lists their in-toto attestations and associates the image with its attestations. A verifier which has the correct layout and root of trust from the layout role can then use the image signer role's attestations and image to verify an image. in-toto attestations are stored in the namespace of the image they apply to, so that verifiers always receive the correct attestations, and not some from older versions of the image. The in-toto attestations for an image also identify the image they apply to by its hash, as the image is recorded in the attestations as the final product of the supply chain. These factors prevent a mix-and-match attack [29] on the in-toto attestations that uses some current, and other out-of-date attestations.

This role, while much more frequently used than the layout role, is also important to protect because it gatekeeps when ingestion of an image may occur. Thus in practice, its keys are protected using measures like storing them offline using hardware keys which are only plugged in when needed.

C. Verifying in-toto metadata at the repository

In Uptane, we have two repositories, the Image and Director. In addition, there is typically a gateway component that transmits the software artifact to the target automobiles. In order to achieve the multiple points of verification discussed in section IV, we perform verification of software supply chain security policy conformance at each of these points in the distribution pipeline. Further, each successive verifier is expected to verify the summary verification of each predecessor as well. Thus, each verifier has a unique policy defined in a layout signed by the layout signer role, and verifies in-toto attestations as well as previous verifications for the image. As discussed in section IV, using three points of prevents a single point of failure while reducing cost and complexity.

All together, this means that Scudo adds in-toto verification at three points: at image ingestion (Image repository), image selection (Director repository), and at update time (SOTA Gateway). Section VI explores the implications and properties of performing verification at these phases. The Image repository verifier only signs updated Uptane metadata if the in-toto verification is successful using a layout specifically written for the Image verifier and root of trust previously established in the layout signer role. In doing so, the verifier generates a signed summary of verification which it associates with the image in the repository's Uptane metadata. The Director repository performs in-toto verification using the layout corresponding to the Director verifier and root of trust from the layout signer role. This layout requires the Director to veryify the Image repository verifier's summary in addition to verifying all the software supply chain metadata. The Director repository also generates a summary of verification, which is recorded only in the Director repository's Uptane metadata, associated with the image. The SOTA Gateway performs in-toto verification when an update is sent to automobile ECUs. Like the Director repository, the SOTA gateway uses the corresponding layout which indicates that it will check that prior verifiers (both the Image and Director repositories) validated the image by verifying their summary verifications in addition to performing a full check of the artifact's in-toto metadata.

D. Verifying supply chain integrity at the vehicle

In place of the full supply chain metadata, Scudo dispatches three verification summaries to the vehicle along with the image to install and SOTA metadata. The supply chain owner signs an in-toto layout indicating a policy that the vehicle must verify these three summaries. This policy is bootstrapped much like the repository-side policies, using Uptane's delegations.

Specifically, the vehicle checks that the summaries were created for the artifact being installed. Further, it also verifies that each of the repository-side verifiers performed full verification, which is indicated by their signatures on the corresponding summaries. Thus, the vehicle is able to verify the validity of the artifact being installed without shouldering the overhead of performing full verification.

VI. Security Analysis

This section examines the security properties of Scudo by studying the implications of a compromise of each actor, as allowed by our threat model. We assume that software supply chain policies are reasonably secure and check the input and output of each step to detect tampering.

We consider a compromise of each actor to analyze the security impact in the context of our threat model. In order of impact, this is how each actor can influence the software supply chain and its verification. Recall that our goal is to install the software update indicated by the software update director, with all steps described by the supply chain policy manager occurring.

A. Repository owners

First, the most important role is the **repository owners**. For Uptane, this may be two distinct groups: the root of trust for the Image repository and the root of trust for the Director repository. If these repository owners are compromised, they could replace the root of trust for the software supply chain, or direct vehicles to install the incorrect images. Such a compromise would require compromising a threshold of offline keys, and is unlikely to occur in practice. A complete root of trust compromise is out of scope in our threat model.

B. Software supply chain owners

On the software supply chain side, the **software supply chain owners** can weaken policies by distributing weak layouts. This role is critical with a lot of responsibility, and it is hard to stop misbehavior. After such an attack, in-toto's root of trust needs to be revoked and rotated by Uptane's root of trust. Then, new layouts must be issued that are signed with new, securely stored keys. The layout signer role at the Image repository is updated to revoke the old layouts and root of trust entirely in favor of the new ones. Note that if multiple layouts exist for different versions of some image, all of these layouts must be re-issued using the new in-toto root of trust. Here, we see the value of using a secure SOTA system like Uptane to bootstrap in-toto's root of trust. The use of a weaker SOTA system that results in in-toto's root of trust being flashed into ECUs requires *every vehicle* to be independently updated, not just those affected by a malicious image. Scudo's use of Uptane allows for recovery when the compromise is detected. Also note that a compromise of in-toto's root of trust means compromising a *threshold* of keys that are stored securely, much like the keys used for other secure roles at the repository. Such a compromise is unlikely.

C. Software supply chain functionaries

Next, the **software supply chain functionaries** can impact the software that is produced and / or the metadata generated for the software build. However, it may be possible to detect this misbehavior with the right software supply chain policies. For example, if a policy requires that the output of one step matches the input to another, any changes at that stage would be detected. Further, supply chain functionaries are defined in the in-toto layout, and can be replaced once their compromise has been detected by replacing the definitions in the layout.

D. Artifact uploader

An **artifact uploader** can tamper with the software or metadata at upload time. Trivial instances of such tampering ought to be detected using the software supply chain policies. Once such tampering is detected, the artifact uploader's key can be revoked using the Uptane root of trust. The root of trust does so by signing a new delegation removing the compromised key. Rollback attack prevention mechanisms ensure that old metadata listing the old key will not be used.

E. Verifiers

We previously described verifiers at the Image repository, Directory repository, and the SOTA gateway. Each of these perform full in-toto verification of all attestations as well as validate the signatures from prior verifiers. To understand how a verifier can misbehave, we consider the following cases. First, a buggy artifact upload process may bypass the verifier altogether, meaning a verification summary does not exist. This is caught by all subsequent repository-side verifiers and the vehicle. Second, a malicious verifier may sign an invalid verification summary claiming in-toto verification was successful even when it was not. This is again detected by other repository-side verifiers (though not the vehicle) as they *also* perform full in-toto verification for an image. When in-toto verification fails at one verifier and a successful verification summary from another verifier is observed, it is clear one of the verifiers is misbehaving. Our threat model assumes that some, but not all, verifiers can be compromised. As long as one verifier remains uncompromised, any attack can be detected by this uncompromised verifier. Malicious verifiers can misbehave in other ways such as performing a denial of service attack, but these are general problems for the image ingestion and distribution process and not unique to Scudo. As such, we do not discuss them.

VII. Collaboration with Toradex

Scudo adds software supply chain security verification to Uptane at several points. A natural question is how the addition of in-toto impacts security and performance. The answer is highly dependent on how in-toto's policy is configured and attestations are generated. So, to answer this, we collaborated with Toradex to generate in-toto metadata for Torizon images, which could be verified against the corresponding in-toto layout.

Torizon is built using Yocto Project [39] tools like BitBake [2] and OpenEmbedded Core [23]. We generated in-toto attestations recording the sources used to build an image, the build configuration, the external dependencies, and finally the build process itself that uses all of these artifacts. These attestations are verified using an in-toto layout with strong rules governing which artifacts each step could manipulate and the types of manipulations allowed. For example, the dependency fetch step is only allowed to write to certain directories in the builder where dependency artifacts are stored. This step is not allowed to make changes to the build cache. The layout is also configured to detect unauthorized modifications to artifacts between two steps. For example, it validates that all the artifacts used during the final build step are identical to those recorded during the source, configuration, and dependency fetch steps.

Configuration	Uptane	in-toto	Total	Proportion
Uptane only	297.65 KiB	0	297.65 KiB	0.155%
Scudo	958.76 KiB	6 KiB	964.76 KiB	0.504%

TABLE I. WE COMPARE THE METADATA OVERHEAD ON VEHICLE OF SCUDO AND UPTANE. THE PROPORTION INDICATES THE METADATA OVERHEAD COMPARED TO OUR REFERENCE IMAGE. UPTANE INCLUDES THE METADATA FROM BOTH THE IMAGE AND DIRECTOR REPOSITORIES. SCUDO'S UPTANE METADATA SEES A MINOR INCREASE DUE TO ASSOCIATING EACH IMAGE WITH ITS ATTESTATIONS. THE IN-TOTO METADATA INCLUDES THE SUMMARY ATTESTATIONS AND VEHICLE LAYOUT.

All of these checks performed by in-toto are not part of a standard Uptane deployment. Note that Scudo cannot guarantee all these checks are performed as they must be declared for each supply chain in the in-toto layout. However, Scudo enables performing these checks, and thus can be leveraged to achieve a security posture that is a clear improvement over a standard Uptane deployment, while also ensuring the security of in-toto's root of trust keys and layouts.

Toradex's existing use of Uptane protects them from repository compromises. By integrating in-toto via Scudo, Toradex extend their software supply chain security to cover the build pipeline, to ensure an image is built using the right set of inputs, enforce the integrity of the build steps, and to verify that cached artifacts are not tampered with. Over time, the in-toto policy can be updated to include protections against compromise of the version control system [4], [27] and to incorporate security mechanisms like Reproducible Builds [28], [13] to protect against a build server compromise [6]. For now, the in-toto layout described above presents a sane, security forward position, ahead of industry trends.

VIII. METADATA OVERHEAD

To evaluate Scudo's metadata overhead, we used data gathered from Toradex's deployment. All performance numbers were gathered using a computer with an Intel i7-1185G7 processor and 32GB of RAM as a representative of a repository-side verifier. For ECU verification, we measured the runtime overhead on a Toradex Colibri iMX7D v1.1B board. Additionally, we use release 5.7.0 of Torizon as a reference image. This image has a compressed size of 186.90 MiB. Finally, the Toradex Uptane metadata lists 204 images, each of which is associated with seven in-toto attestations. It is likely that Toradex's frequent release cycle results in a greater number of total images on the repository than typical automotive deployments, so this represents an upper bound for Uptane metadata size. Also, while the Uptane metadata evaluated here lists each attestation for an image as an independent artifact, this could be optimized to use a single entry for a compressed archive of all attestations. Note that all runtime numbers are the average of 10 runs.

A. What proportion of transmitted data is the in-toto and Uptane metadata?

To answer this question, we first calculated the compressed size of Uptane metadata alone, which Toradex had deployed prior to adopting Scudo. As Table I shows, the metadata makes up about 0.155% of the total transmission (i.e., the metadata and the image itself). Then, we updated the Uptane metadata to include in-toto for all 204 images. This includes in-toto's layout, root of trust, and attestations for all images. We found that the Uptane metadata and summary in-toto verification with Scudo makes up 0.504% of the total transmission to end devices. On the repository, we see a larger increase in the Uptane metadata from 297.65 KiB to 958.76 KiB in addition to 41.437 MiB of in-toto metadata. Note that while the total metadata sent to these verifiers is larger, this cost is only paid once for each image as the result of in-toto verification is cached. Table I also shows the contribution of in-toto and Uptane metadata to the overall increase in each of these scenarios. Significantly, we see that Scudo has a negligible impact on the data transmitted to end devices. A device fetching an update from the repository with 204 images fetches about 649.11 KiB extra, with the metadata comprising 0.504% of the total transmission size. The impact is even smaller in repositories with fewer images, and can be further reduced by using a single entry for all image attestations as described above. However, as the overhead is already quite trivial, this optimization was omitted in this deployment.

Verifier	Average Runtime (s)
Image Repository	1.34
Director Repository	1.33
SOTA Gateway	1.32

TABLE II. OVERHEAD IMPOSED BY IN-TOTO VERIFICATION AT EACH REPOSITORY SIDE VERIFIER. ALL NUMBERS ARE AN AVERAGE OF 10 VERIFICATION RUNS PERFORMED ON AN INTEL CORE I7-1185G7 MACHINE WITH 32 GB OF RAM.

B. What is the runtime overhead of in-toto verification?

The time taken to perform in-toto verification for a single image at three verifiers at the repository is about **3.99s** in total, as shown in Table II. The Image and Director repository verifiers perform this verification during image ingestion, meaning the overhead is not imposed on the image distribution to devices. The SOTA gateway verifier takes about **1.32s** for its verification, which happens when the image is sent to the device. However, for any one image, this verification can be cached by the verifier to minimize the overhead. Therefore, we find that in-toto verification during the image ingestion, image selection, and update processes does not add significant overhead.

We also measured the runtime overhead of performing in-toto verification of the summary attestations on Toradex's embedded device. The verification takes **0.21s** per image, on hardware representative of an automobile ECU. To confirm the infeasibility of full in-toto verification on device, we repeated the repository side verification on the Toradex device, and we found that this took an average of **37.23s**. The amount of metadata transmitted is also far higher, equivalent to what is used by the repository side verifiers. This is clearly impractical and demonstrates the benefit of using Scudo.

IX. RELATED WORK

We examine other work on software supply chain security for automobiles.

We first discuss the problem of securing the delivery of artifacts from a repository to vehicles. Scudo is designed to work with Uptane, the widely deployed solution addressing the delivery problem. An alternative approach to Uptane as a firmware over-the-air system is described by Mbakoyiannis et al [19]. The research team focus on in-vehicle implementation aspects not previously delineated by Uptane to propose a prototype system. Their system does not address real world deployability of software supply chain security solutions as Scudo does. Plappert et al [26] also create an on-vehicle design for software updates, using TPMs as a trust anchor in a way that is compatible with Uptane and other automotive software supply chain standards.

Several other research teams have also proposed software supply chain solutions for the automotive sector. Kent et. al. [12] propose a PKI Infrastructure Scheme for verifying software updates that allows ECUs to verify signatures from both suppliers and OEMs. *UniSUF* [34], developed by Strandberg et al., securely encapsulates multiple signing and encryption keys and all data needed for a complete software update into one single file. The authors claim that this Vehicle Unique Update Package (VUUP) can provide updates using a number of delivery systems. Metadata for these software supply chain solutions could be distributed by a SOTA system using Scudo.

X. Conclusion and Future Work

This paper examines the practicalities and pitfalls of deploying software supply chain security solutions in the automotive space. Although combining SOTA and software supply chain security systems may seem straightforward, our analysis has shown there are many potential issues that may arise if this is not carefully done.

To show an exemplar of this process, this paper presents Scudo, a solution that integrates Uptane and in-toto. Scudo carefully considers how to establish a root trust for in-toto through Uptane metadata in a way that provides security and flexibility. Scudo also demonstrates that software supply chain verification may actually be quite done efficiently and that only a negligible amount of added information needs to be sent to vehicles. Scudo's design is shown to make it more resilient and adaptable than Uptane or in-toto alone.

Finally, our collaboration with Toradex demonstrates that using Scudo is efficient in practice and practical for existing Uptane users. Scudo adds a one-time cost of about 4 seconds across all repository verifiers, and is able to eliminate the need for costly on-device verification through the use of multiple repository verifiers. Scudo also imposes a minimal metadata overhead that comprises 0.504% of the data devices need to fetch during an update and a per-image verification time overhead of 0.21 seconds.

Future work could build on the work described here to assess how in-toto can be configured so as to reduce the overhead at repositories. While attestations can be optimized in several ways, we must carefully study their impact on the security guarantees as some optimizations may result in weaker policies. In addition, research could study how Scudo's properties change with the constraints in other deployments such as the cloud and open source community repositories.

XI. Acknowledgements

We would like to thank Toradex for their support in this work, especially Brandon Shibley, Jon Oster, and Samuel Bissig. In addition, we are grateful to Lois Anne DeLong and Trishank Karthik Kuppusamy for their help and feedback on Scudo.

This material is based upon work supported by the National Science Foundation under Grant No. 2054692. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect those of the NSF. This work was also supported by a GAANN Fellowship awarded by the US Department of Education under grant P200A210096.

References

[1] J. R. Biden, "Executive order on improving the nation's cybersecurity," https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/, 2021.

[2] "BitBake," https://docs.yoctoproject.org/bitbake.html.

[3] "CNCF: Catalog of supply chain compromises," https://github.com/cncf/tag-security/tree/main/supply-chain-security/compromises.

[4] Codecov, "Bash uploader security update," https://about.codecov.io/security-update/, 2021.

[5] "Dsa-1571-1 openssl – predictable random number generator," "https://www.debian.org/security/2008/dsa-1571", 2008.

[6] FireEye, "Highly evasive attacker leverages solarwinds supply chain to compromise multiple global victims with SUNBURST backdoor," https://www.fireeye.com/blog/threat-research/2020/12/evasive-attacker-leverages-solarwinds-supply-chain-compromises-with-sunburst-backdoor.html, 2020.

[7] D. Geer, B. Tozer, and J. S. Meyers, "For good measure: Counting broken links: A quant's view of software supply chain security," in *USENIX; Login:, Vol. 45, no. 4*, 2020.

[8] Grafeas Authors, "Grafeas," https://grafeas.io/.

[9] M. Hanley, "We updated our rsa ssh host key," "https://github.blog/2023-03-23-we-updated-our-rsa-ssh-host-key/".

[10] "in-toto specification," https://github.com/in-toto/docs/blob/master/in-toto-spec.md.

[11] Infosecurity Magazine, "Flame attackers used cryptographic collision attack," https://www.infosecurity-magazine.com/news/more-from-microsoft-flame-attackers-used/, 2012.

[12] D. Kent, B. Cheng, and J. Siegel, "Assuring Vehicle Update Integrity Using Asymmetric Public Key Infrastructure (PKI) and Public Key Cryptography (PKC)," *SAE International Journal of Transportation Cybersecurity and Privacy*, vol. 2, 08 2020.

[13] kpcyrd, "Rebuilderd," https://github.com/kpcyrd/rebuilderd.

[14] T. Kuppusamy, A. Brown, S. Awwad, D. McCoy, R. Bielawski, C. Mott, S. Lauzon, A. Weimerskirch, and J. Cappos, "Uptane: Securing Software Updates for Automobiles," *Escar Europe*, vol. 14, 2016.

[15] T. Kuppusamy, L. DeLong, and J. Cappos, "Securing software updates for automotives using uptane," *Usenix login*, vol. 42, pp. 63–67, 2017.

[16] T. K. Kuppusamy, V. Diaz, and J. Cappos, "Mercury: Bandwidth-effective prevention of rollback attacks against community repositories," in *Proceedings of the 2017 USENIX Conference on Usenix Annual Technical Conference*, ser. USENIX ATC '17. USA: USENIX Association, 2017, p. 673–688.

[17] T. K. Kuppusamy, S. Torres-Arias, V. Diaz, and J. Cappos, "Diplomat: Using delegations to protect community repositories," in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. Santa Clara, CA: USENIX Association, Mar. 2016, pp. 567–581. [Online]. Available: https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/kuppusamy

[18] Kuppusamy, Trishank and DeLong, Lois and Cappos, Justin, "Uptane: Security and Customizability of Software Updates for Vehicles," *IEEE Vehicular Technology Magazine*, 02 2018.

[19] D. Mbakoyiannis, O. Tomoutzoglou, and G. Kornaros, "Secure over-the-air firmware updating for automotive electronic control units," in *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, ser. SAC '19. New York, NY, USA: Association

for Computing Machinery, 2019, pp. 174–181. [Online]. Available: https://doi.org/10.1145/3297280.3297299

[20] Z. Newman, J. S. Meyers, and S. Torres-Arias, "Sigstore: Software signing for everybody," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 2353–2367. [Online]. Available: https://doi.org/10.1145/3548606.3560596

[21] M. Ohm, H. Plate, A. Sykosch, and M. Meier, "Backstabber's knife collection: A review of open source software supply chain attacks," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2020, pp. 23–43.

[22] C. Okafor, T. R. Schorlemmer, S. Torres-Arias, and J. C. Davis, "SoK: Analysis of Software Supply Chain Security by Establishing Secure Design Properties," in *Proceedings of the 2022 ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses*, ser. SCORED'22. New York, NY, USA: Association for Computing Machinery, 2022, p. 15–24. [Online]. Available: https://doi.org/10.1145/3560835.3564556

[23] OpenEmbedded, "Openembedded core," https://www.openembedded.org/wiki/OpenEmbedded-Core.

[24] OWASP Foundation, "CycloneDx," https://cyclonedx.org/, 2021.

[25] E. Parliament, "Eu cyber-resilience act," https://www.europarl.europa.eu/RegData/etudes/BRIE/2022/739259/EPRS_BRI(2022)739259_EN.pdf.

[26] C. Plappert and A. Fuchs, "Secure and lightweight ecu attestations for resilient over-the-air updates in connected vehicles," in *Proceedings of the 39th Annual Computer Security Applications Conference*, ser. ACSAC '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 283–297. [Online]. Available: https://doi.org/10.1145/3627106.3627202

[27] N. Popov, "Changes to git commit workflow," https://news-web.php.net/php.internals/113838, 2021.

[28] "Reproducible builds," https://reproducible-builds.org/.

[29] J. Samuel, N. Mathewson, J. Cappos, and R. Dingledine, "Survivable key compromise in software update systems," in *Proceedings of the 17th ACM conference on Computer and communications security*. ACM, 2010, pp. 61–72.

[30] "Supply chain levels for software artifacts," https://slsa.dev/.

[31] "SLSA Verifier," https://github.com/slsa-framework/slsa-verifier.

[32] Sonatype, "State of the software supply chain," https://www.sonatype.com/resources/2023-software-supply-chain-report, 2022.

[33] SPDX Workgroup, "The Software Package Data Exchange," https://spdx.dev/, The Linux Foundation, Tech. Rep., 2021.

[34] K. Strandberg, D. K. Oka, and T. Olovsson, "Unisuf: a unified software update framework for vehicles utilizing isolation techniques and trusted execution environments," in *19^{th} escar Europe : The World's Leading Automotive Cyber Security Conference (Konferenzveröffentlichung)*, 2021.

[35] S. Torres-Arias, H. Afzali, T. K. Kuppusamy, R. Curtmola, and J. Cappos, "in-toto: Providing farm-to-table guarantees for bits and bytes," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 1393–1410. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/torres-arias

[36] UNECE-29, "Concerning the adoption of harmonized technical united nations regulations for wheeled vehicles, equipment and parts which can be fitted and/or be used on wheeled vehicles and the conditions for reciprocal recognition of approvals granted on the basis of these united nations regulations, addendum 155 – UN regulation no. 156," https://unece.org/sites/default/files/2021-03/R156e.pdf.

[37] Uptane Community, "Uptane series 2.0.0," Joint Development Foundation Projects, LLC, Standard, 2022. [Online]. Available: https://uptane.github.io/papers/uptane-standard.2.0.0.html

[38] D. Waltermire, "Software Identification (SWID) Tagging," https://csrc.nist.gov/projects/Software-Identification-SWID, NIST, Tech. Rep., 2022.

[39] "Yocto Project," https://www.yoctoproject.org/.