2025 NDIA MICHIGAN CHAPTER
GROUND VEHICLE SYSTEMS ENGINEERING
AND TECHNOLOGY SYMPOSIUM
MODULAR OPEN SYSTEMS APPROACH TECHNICAL SESSION
AUG. 12-14, 2025 - NOVI, MICHIGAN

ADVANCING AUTOMOTIVE SOFTWARE SUPPLY CHAIN SECURITY: A BLOCKCHAIN-REPRODUCIBLE BUILD APPROACH

Iwinosa Aideyan[1], **Mert D. Pesé, PhD**[2], **and Richard Brooks, PhD**[1]

[1] Holcombe Department of Electrical and Computer Engineering, Clemson University
[2]School of Computing, Clemson University

ABSTRACT

The automotive industry's systems and over-the-air (OTA) updates have vulnerabilities in its software supply chain (SSC). Although frameworks like Uptane have improved OTA security, gaps remain in ensuring software integrity and provenance. In this paper, we examine challenges securing the automotive SSC and introduce a framework, ***GUIXCHAIN****, that integrates version control, reproducible builds, blockchain technology, and software bills of materials (SBoMs) for transparency, auditability, and resilience. Reproducible builds guarantee identical resulting binaries when compiling the same source code in different environments, as any deviation in the final output indicates a potential compromise in the build process, such as malware injection. Our preliminary study shows Guixchain's use of reproducible builds ensures consistent and integrity-secured software across various build environments. The blockchain provides forensic capabilities, offering a history of the what, who and where of discrepancies within the SSC process. SBoMs provide an inventory of the software components used. Our preliminary study demonstrates that Guixchain effectively mitigates risks such as ransomware, unauthorized modifications, and build server compromises, reinforcing the system's integrity and resilience throughout the software life cycle. Future work will focus on the full implementation of Guixchain and a comprehensive evaluation of its performance in real-world automotive software supply chain scenarios.*

Citation: I. Aideyan, M. Pesé, R. Brooks, "Advancing Automotive Software Supply Chain Security: A blockchain-reproducible build Approach," In *Proceedings of the 2025Ground Vehicle Systems Engineering and Technology Symposium (GVSETS)*, NDIA, Novi, MI, Aug. 12-14, 2025.

1. INTRODUCTION

Modern vehicles contain over 100 million lines of code spread across numerous Electronic Control Units (ECUs), improving functionality and introducing a complicated network of suppliers [1]. Original Equipment Manufacturers (OEMs) must coordinate a diverse ecosystem of tier 1, tier 2, and tier 3 suppliers, where a single fault from any vendor could compromise the entire system.

Historically, updating ECUs required physical visits to dealerships, but the advent of over-the-air (OTA) updates revolutionized this process. The Uptane framework [2], the most widely implemented OTA solution, secures the delivery of updates using advanced cryptography and secure communication protocols. However, it does not address SSC security of the updates themselves particularly their integrity before delivery.

To address these gaps, the in-toto framework was introduced in 2019 to protect software from its initial creation to end-user installation [3]. Its integration with Uptane led to the Scudo framework [4], which offers end-to-end protection for automotive ECUs using Uptane's pre-distributed keys. However, it still has limitations: compromised keys can enable unauthorized actions, replay attacks remain possible, metadata overhead can affect scalability, and integrating with legacy systems can be difficult. Moreover, in-toto does not safeguard the build environment itself, leaving the compilation process, tools, or environment vulnerable if they become compromised. These constraints underscore the need for an alternative approach.

This paper presents GuixChain, a novel approach that integrates Git, blockchain, reproducible builds and Software Bill of Materials (SBoM). It leverages:

Git and Blockchain Integration: Git's distributed version control ensures precise tracking of changes in the source code [5], while the blockchain adds an immutable, transparent ledger that permanently records every change. This combination addresses security and auditability requirements. While Git excels at version tracking, its history can be altered and lacks immutability. The blockchain ledger, therefore, serves as an immutable audit trail, ensuring that every code change is permanently recorded and verifiable, which is crucial for tracing the origins of malicious components. This dual approach mitigates risks like a single point of failure, Distributed Denial of Service (DDoS), ransomware, and unauthorized modifications, and enhances transparency in multi-stakeholder environments, effectively creating a verifiable and tamper-proof record of the software supply chain.

Reproducible Builds: Protect against malicious code injection and build server compromises. By integrating Blockchain's smart contracts that trigger parallel, deterministic compilation across multiple nodes, GuixChain guarantees that identical source code produces byte-for-byte consistent binaries. Smart contracts are tamper-resistant programs that run on blockchain technology, automatically enforcing agreements between parties without intermediaries. This reinforces build integrity and makes any tampering immediately evident.

Software Bill of Materials: SBoM provides detailed traceability of every software component [6]. When artifacts such as images, metadata, and SBoMs are hashed and cross-verified across nodes before being transmitted to the Uptane repository, any discrepancy prevents unauthorized modifications from being deployed. Additionally, SBoMs used to find Common Vulnerabilities and Exposures (CVE) entries in software libraries.

These integrated technologies secure the automotive software supply chain by ensuring the provenance and reproducibility of software artifacts. The immutable ledger, deterministic build processes, and enhanced traceability combined counteract threats such as arbitrary edits, insider threats, or build server compromises. This approach significantly reduces the risks of supply chain

attacks. This paper makes the following main contributions:

- An analysis of automotive SSC challenges.
- An evaluation of automotive industry SSC frameworks.
- The Guixchain framework to provide end-to-end security from code development to artifacts repository.
- Blockchain-enabled reproducible builds for decentralized verification to ensure the software compiled from the same source code results in identical outputs across different nodes, increasing resilience against malware injection.
- SBoM not merely as a compliance requirement but as a proactive security mechanism for real-time tracking, verification, and vulnerability assessment of software components throughout the supply chain.
- A detailed threat model and implementation strategy addressing critical vulnerabilities.

We explore automotive software supply chain security, beginning with background and challenges in Sections 2-3. Section4 reviews existing frameworks, and Section 5 discusses OTA updates. Sections 6-7 present and evaluate our GuixChain solution. Section 8 summarizes our findings.

2. BACKGROUND

Manufacturers use computer systems to advance performance, value, comfort, entertainment, maintenance, diagnostics, and safety in the automotive industry. These embedded systems run on firmware/software. They include input/output components, peripherals, real-time operating systems, and communication networks. The evolution of automotive software development began in the 1980s and 1990s with hand coding

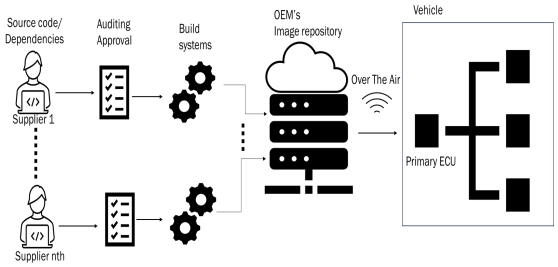

Figure 1: Vehicle Software Update Supply Chain

and has progressed through model-based development (MBD), MBD with automatic code generation [7], and now to update capabilities via OTA.

Before the advent of OTA updates [8], vehicle owners had to visit dealerships for ECU software updates, a process that was inconvenient, costly, and time-consuming. Technicians at dealerships would connect specialized equipment to the vehicle's diagnostic port, typically an onboard diagnostics (OBD) system, to download new firmware onto the ECUs. This controlled environment ensured safe updates but lacked flexibility and customer convenience.

OTA updates allow ECUs to be updated remotely without physical access, significantly enhancing the flexibility and immediacy of software deployments, as seen in Figure 1. In this setup, *primary* ECUs such as the Telematic Control Unit (TCU) receive updates over the air. They can either distribute the update package to *secondary* ECUs or instruct them to fetch and install updates directly, minimizing downtime and service interventions.

As mechanical systems are replaced with embedded systems in vehicles, it is crucial to consider the characteristics of automotive software, which include functional safety, reliability, dependability, security, real-time capabilities, complexity, fault tolerance, and fault recovery. Throughout the software development lifecycle (SDLC)

[9], code is created using the V-cycle model to ensure complete verification and validation of all processes, from requirement analysis to coding, testing, and deployment. Ensuring complete implementation of these characteristics during the update process is vital.

As the automotive sector progresses towards electric vehicles, autonomous or semi-autonomous fleets, self-driving cars, and software-defined vehicles (SDVs), the complexity involved in vehicle development escalates. Consequently, a greater reliance on software within ECUs, such as safety systems, maps and navigation software, entertainment and communication systems, and connectivity with other consumer devices, significantly influences vehicle quality and performance [7]. Alongside this evolution and automotive software development, concerns about cybersecurity and vulnerabilities that malicious actors could exploit have intensified.

OEMs depend on a network of suppliers for critical components and software, either from tier 1 or tier 2 providers. This dependence persists even when OEMs develop software internally, frequently utilizing codebases or templates from these suppliers as a foundational element. This practice leverages the supplier's expertise and pre-existing solutions, tailored to meet the specific requirements of different OEMs [10]. Ironically, while OEMs may procure similar products from the same suppliers, these products are customized to align with each OEM's unique branding, functionality, and performance specifications. Consequently, companies that serve multiple OEMs across various programs and product lines face a complex web of differing configuration management requirements, change control policies, and data management procedures, many of which are inadequately documented and communicated [11], as seen in Figure 2.

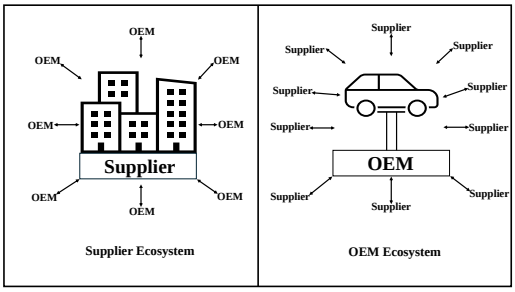

Figure 2: Supplier and OEM ecosystems

The relationship between OEMs and suppliers is characterized by collaboration and customization, which presents several software challenges regardless of in-house development. The increasing complexity and interconnectedness of software supply chains renders them susceptible to significant security threats, as the SolarWinds supply chain attack vividly illustrates. In 2020, malicious actors compromised SolarWinds' Orion software, widely used for network management [12]. By injecting malicious code into the software updates, the attackers accessed the networks of thousands of SolarWinds customers, including major corporations and government agencies.

Another example is the *log4j* vulnerability [13], known as Log4Shell, which was disclosed in December 2021. It affected the Apache *log4j* library, a Java-based logging utility; the vulnerability allowed log messages to contain placeholders that were dynamically replaced at runtime because the Java Naming and directory interface(JNDI) lookup in *log4j* was implemented insecurely. An attacker could trigger *log4j* to execute arbitrary Java code remotely. The supply chain issue was that many versions of *log4j* had enabled the JNDI lookup feature by default, making a wide range of applications and services vulnerable to remote code execution attacks.

These concerns include recent attacks on Okta[14], a widely used identity and access management provider. In this

incident[15], the hacking group Lapsus$ gained unauthorized access to a third-party support vendor, Sitel, which Okta relies on for customer support services. The attackers compromised a Sitel employee's account, allowing them to view internal Okta support tools and systems for about five days. This breach, affecting fewer than 3% of Okta's customers, raised serious concerns because Okta's identity and access management solutions lie at the heart of many organizations' authentication and single sign-on processes[16].

Imagine if a similar attack were to target a top-tier agency, such as the Directorate of Defense Trade Controls (DDTC), part of the U.S. Department of State or the Centers for Medicare Medicaid Services (CMS), an agency within the Department of Health and Human Services or other critical governmental bodies. These organizations depend on Okta's secure identity management to protect highly sensitive data and maintain national security. A breach of this magnitude could expose confidential information, disrupt essential services, and compromise national security. Although the Okta incident did not involve the direct injection of malicious code into software updates or the build process, it underscores the broader risks posed by third-party vendor compromises. This incident highlights the critical importance of securing every link in the supply chain, from the software build environment to third-party service providers, to protect against traditional supply chain attacks and emerging threats that target privileged access and support channels.

These incidents show the risk posed by widely-used software components or libraries that, when compromised, can lead to cascading effects across numerous systems and organizations globally. They are particularly relevant to the automotive industry, where integrating components and software from various suppliers can create similar vulnerabilities in multiple vehicles from multiple OEMs.

3. CHALLENGES OF SECURING THE AUTOMOTIVE SOFTWARE SUPPLY CHAIN

According to the 9th Annual State of the Software Supply Chain Report [17], there has been a noticeable increase in the creation of custom packages designed specifically to download and execute harmful payloads automatically, without any interaction from developers. This trend poses significant security risks, particularly in industries like automotive, where software integration is deeply embedded.

The automotive industry faces multifaceted challenges in securing its software supply chain. These challenges span from increasing system complexity and high-quality demands to intense time and cost pressures. Notably, as Grimm [18] pointed out, the rapid expansion of software integration in vehicles introduces several organizational, technical, and strategic challenges. Furthermore, Broy [19] emphasized the difficulties posed by rapid technological changes, increased competition, shortened product life cycles, operational complexity, and risk management.

In addition, the rise of connected vehicles has raised concerns about potential threats from enhanced connectivity. Soriano *et al.* [20] highlight that this increased connectivity necessitates robust security measures, particularly for IoT components and ECUs. To navigate these challenges, automotive companies must develop specific competencies, including robust development processes, quality management, supplier collaboration, and system integration capabilities. In addition, it is vital to

maintain security throughout extended vehicle lifespans and implement continuous vulnerability management strategies to ensure long-term protection within the automotive software supply chain.

In the following, we will look at eight challenges in securing vehicle software supply chains. This examination will draw on insights from two key sources: the observations of 30 industry and government organizations on securing the software supply chain [21] and a proposal to resolve software supply chain vulnerabilities in vehicles [4].

Challenge 1 - Updating vulnerable dependencies: Different vendors have disparate policies on when to update vulnerable dependencies, leading to coordination challenges across the software supply chain. Organizations often hesitate to be the first or the last to implement updates, with some preferring isolated build environments to limit reliance on dependency updates. This lack of standardization complicates the uniform application of security updates. The growing complexity of automotive software and the constant need for updates highlight the importance of effectively managing these dependencies to ensure security over extended lifespans [18], [20], [22].

Challenge 2 - Leveraging SBoM for security: SBoM is mandated by legislation to increase transparency and enhance security within supply chains [23]. However, its potential to improve security is often underutilized and is treated more as a compliance checklist than a proactive security tool. Effective use involves creating meaningful, verifiable security metrics that demonstrate adherence to security policies [21], [24].

Challenge 3 - Choosing trusted supply chain dependencies: Selecting secure and reliable dependencies is essential yet challenging, especially under the zero-trust principle. Although the Open Source Security Foundation (OpenSSF) [25] provides resources to mitigate the risks associated with rogue packages, the balance between thorough vetting and timely project completion remains critical [26].

Challenge 4 - Securing the build process: The security of the build process is paramount, as vulnerabilities in this stage can undermine all previous security measures. One of the critical aspects of securing the build process is ensuring reproducible builds, which means that the builds should be bit-for-bit identical under the same conditions, which helps verify the integrity of the build system. Achieving this across platforms and programming languages presents ongoing challenges[27]–[30] such as:

- Build environment variability: Different build environments may yield different outputs from the same source code. Operating system differences, available system libraries, specific compiler versions, build paths, etc., may vary and affect the result.
- Platform and language differences: Reproducible build techniques vary depending on the language and platform. In return, some languages/tools are designed for reproducibility, while others are not and will need to change or make a specific toolchain.
- Toolchain variability: The tools responsible for the build process (like compilers and linkers) can cause non-determinism. Changes to or updates of these tools can then have an effect on the reproducibility of the builds. It can

be a logistical headache to ensure that different environments use these tools' same versions and configurations.
- Time and date stamps: Many build processes will automatically stamp a current time and date in build outputs, which could create incompatibility between builds done later or earlier.

Challenge 5 - Size and diversity of codebases: The extensive and diverse code bases of modern vehicles, often exceeding 100 million lines of code, complicate the implementation of effective supply chain security strategies [18]. Trust in external suppliers to ensure code quality and security is paramount.

Challenge 6 - Getting industry-wide participation: Securing the software supply chain requires industry participation. Despite initiatives like Sigstore[31] and in-toto[3] laying the groundwork for collective security efforts, widespread adoption is necessary. Bridging the gap between individual company policies and industry-wide best practices requires concerted effort and time [22].

Challenge 7 - Enhancing the use of supply chain management software: Although earlier studies [26] have shown that supply chain management software can reduce costs and work times, its full potential for cooperative integration and collaboration among stakeholders remains largely unexplored. In today's automotive industry, solutions like SAP Integrated Business Planning (IBP)[32], Oracle SCM Cloud[33], Siemens Teamcenter[34], and PTC Windchill[35] are employed to streamline operations, enhance collaboration, and integrate data-driven decision-making across global networks. This indicates a significant opportunity for further integration and innovation to maximize benefits in the automotive supply chain.

Challenge 8 - Addressing rising connectivity and security concerns: The rise in connectivity through connected car technologies, such as 5G, has raised security concerns. Continuous vulnerability management and secure firmware/software updates are crucial. The complexity of SSC, with numerous dependencies, poses significant security challenges with an increasing number of attacks, necessitating robust security measures [21], [24].

4. SSC FRAMEWORKS APPLICABLE TO AUTOMOTIVE INDUSTRY

This section discusses various SSC frameworks that, while commonly employed across multiple industries, hold significant applicability to the automotive sector. These frameworks are instrumental in addressing the unique challenges automotive software supply chains face. Recent advancements in software solutions have aimed to address these challenges through various technologies and methodologies, starting with:

The FISHY platform[24] is a sophisticated cyber-resilient framework explicitly designed to address the myriad of security challenges in the automotive software supply chain, focusing mainly on connected and autonomous vehicles. Its main objective is to forge trusted supply chains by bolstering cybersecurity throughout the vehicle lifecycle. This includes a comprehensive suite of tools for continuous monitoring and proactive vulnerability management, which continuously scans for potential security issues, allowing for the rapid deployment of updates or patches to mitigate risks.

Moreover, FISHY promotes a secure

SDLC that embeds security considerations from design to deployment. This methodical inclusion of security processes, rigorous risk assessments, and software attestation ensures that the integrity and security of each component is verified before integration. The platform emphasizes the importance of data management and privacy, ensuring that sensitive information, such as biometrics and location data, is anonymized and securely managed to maintain user privacy while supporting connected services functionality.

SBoMs provide transparency by offering a machine-readable inventory of software components, allowing faster identification and remediation of vulnerabilities [36]. SBoMs can enhance transparency and traceability in software supply chains as they are a comprehensive list of software components, dependencies, and firmware metadata or a centralized inventory of third-party components and dependencies [37]. SBoM adoption is limited due to differing incentives and concerns among key business stakeholder groups involved in production and consumption.

Sigstore [31] represents a significant advancement in software signing technology, integrating OpenID Connect (OIDC) authentication, ephemeral keys, and transparency logs to bolster security. Developed to streamline the software signing process, developers can quickly authenticate their work without navigating complex security protocols. This system is designed to be straightforward: Developers push their code, and Sigstore manages the authentication and cryptographic processes in the background.

The framework uses transparency logs to record critical information about each piece of software, including details about who created it and where it was built. This ensures that any software verified through Sigstore can be traced back to its origin, assuring that it has not been altered post-signing. Moreover, the data stored within these logs is designed to be easily auditable, laying a robust foundation for integrating monitoring tools and enhancing security workflows. Despite its innovative approach and the security enhancements it brings, Sigstore has faced criticism for its dependency on trusted third parties like OIDC providers such as Google, Microsoft, and other identity providers.

Critics like Chang *et al.* [38] argue that existing command-line interface (CLI) tools could offer similar functionalities without relying on Sigstore's key components, suggesting that the system might benefit from a more decentralized approach to reduce reliance on external validators who are third-party service that independently verifies the authenticity and integrity of the signed software artifacts. These validators check the transparency logs to confirm that the signatures and associated metadata have not been altered since they were recorded. This critique points to an ongoing debate in the software security community about the best balance between ease of use and the minimization of trust assumptions in security architectures

Grafeas/Kritis[39], meaning "scribe" in Greek, is an open-source initiative designed to uniformly audit and govern the SSC by integrating with existing DevOps tools and workflows. It is a central repository for metadata about all software artifacts (container images, virtual machines, binaries, or packages) and their storage location.The primary strength of Grafeas lies in its ability to store, query, and derive valuable insights from the metadata associated with these artifacts. For instance, it can be leveraged to identify all artifacts derived from a specific git commit known to introduce

a security vulnerability and detect artifacts built using a compromised version of a build tool, helping mitigate risks associated with toolchain attacks. It can generate SBoMs for artifacts, facilitating transparency and adherence to regulatory requirements.

Grafeas supports both horizontal and vertical querying of metadata. In contrast, horizontal querying involves querying across all artifacts based on a specific property. Vertical querying focuses on gathering detailed metadata across the software development lifecycle for a specific artifact, such as tracing all source, build, test, and vulnerabilities metadata for a container image.

In-toto provides a comprehensive framework designed to secure the integrity of the software supply chain by meticulously verifying each step from development through deployment[3]. This framework ensures that all processes involved in creating and distributing software are transparent and secure, thereby preventing supply chain attacks that could compromise software before it reaches end users. The essence of in-toto lies in its layout specification, which acts as a blueprint for the SSC. This layout outlines the steps required in the software development process, identifies the authorized functionaries to perform these steps, and specifies the expected outcomes of each step. The layout itself is signed by a trusted party, which establishes a foundation of trust and integrity before any software development begins.

For example, imagine a recipe for baking a cake. The recipe (layout) specifies every step: mixing ingredients, baking, and decorating, along with who is responsible for each step (the functionaries). As you follow the recipe, you take pictures or notes at each step (link metadata) to show that the process is being followed correctly. Later, if someone altered the cake without your knowledge, you could compare the notes to the recipe and detect the deviation.

In-toto works in a similar way: as each step in the software development lifecycle is executed, cryptographically signed metadata (link metadata) is generated, recording the state of the software artifacts. This metadata creates an auditable chain that makes it easy to verify that every step was completed as prescribed and to identify any unauthorized changes. By capturing and verifying every action within the software supply chain, in-toto provides robust protection against various attack vectors. Requiring that all steps, from initial code generation to final deployment, are verifiable and traceable ensures that no unauthorized changes or malicious code insertions occur at any point in the supply chain [4].

This section on SSC challenges and existing solutions explored several technologies designed to enhance security throughout the software lifecycle. The following section looks at existing frameworks tailored explicitly for automotive OTA updates.

5. AUTOMOTIVE OTA UPDATES

This section discusses a prevalent OTA framework used in the automotive industry, Uptane[2], and its extension, Scudo[4], to add SSC security. The workflow for OTA vehicle updates involves securely transmitting the updates from the OEM's servers directly to the car. This process typically encompasses different stages:

Update Packaging: OEMs package the new software or firmware update, which may include new features, security patches, or system optimizations.

Secure Transmission: The update is transmitted over a cellular or Wi-Fi

connection to the vehicle's infotainment system or telematics control unit.

Verification: Upon receipt, the vehicle verifies the authenticity and integrity of the update using digital signatures.

Installation: The firmware on the ECU is updated, which can occur either in the background while the vehicle is operational or when parked, depending on the nature of the update.

5.1. The Uptane Framework

Uptane (**U**pdate **P**rotocol for **T**ransportation **a**nd **Ne**twork Security) [2] is a foundational framework designed to secure OTA software updates in the automotive sector. It enhances the integrity and authenticity of software updates through a blend of signing, verification, and redundancy mechanisms aimed at thwarting various threats, including those from malicious actors or compromised servers. Uptane's architecture is built on the principle of a two-tier structure [40], an image repository, and a director repository that collaborates to ensure the secure delivery of updates, as seen in Figure 3. The image repository stores the actual software update images (like firmware files) along with their signed metadata, while the director repository determines which specific updates should be sent to each ECU based on its current state, essentially acting as a guide for the update process by providing signed metadata that instructs which images to install from the image repository; both repositories work together to ensure secure software updates by verifying the authenticity and integrity of the updates throughout the process. The primary ECU acts as a gatekeeper, receiving updates, verifying their authenticity using extended principles from The Update Framework [41], and distributing them to secondary ECUs. Each secondary ECU independently verifies the update before installation, protecting against the propagation of potentially malicious or compromised updates within the vehicle.

Moreover, Uptane allows for customizable security policies tailored to different types of ECUs, enabling manufacturers to adjust security levels based on the criticality of functions performed by each ECU. It also includes mechanisms to accommodate vehicles that may not connect consistently to the Internet, facilitating secure offline updates and interactions between the primary and secondary ECUs. Its threat model systematically categorizes potential attacks into four primary groups, which are listed below:

Read Updates: Addresses the threat of intellectual property theft. In this scenario, attackers might perform an *eavesdropping attack*, intercepting unencrypted software updates while transmitting from the repository to the vehicle. Such unauthorized access could lead to the extraction of sensitive data or proprietary software, potentially resulting in significant intellectual property losses.

Deny Updates: This focuses on preventing vehicles from accessing necessary software updates. This can manifest through several attack strategies: *Drop-request attacks* block network traffic to prevent ECUs from receiving updates. In contrast, *slow-retrieval attacks* deliberately delay the delivery of updates to exploit known vulnerabilities. *Freeze attacks* repeatedly send outdated software to prevent newer updates from being installed, and *partial-bundle installation attacks* disrupt update integrity by allowing only parts of the update to be installed.

Deny Functionality: Attackers aim to cause vehicle malfunction. *Rollback attacks* trick an ECU into reinstalling outdated software with known vulnerabilities, while *endless data attacks* overwhelm an ECU with excessive data, leading to storage

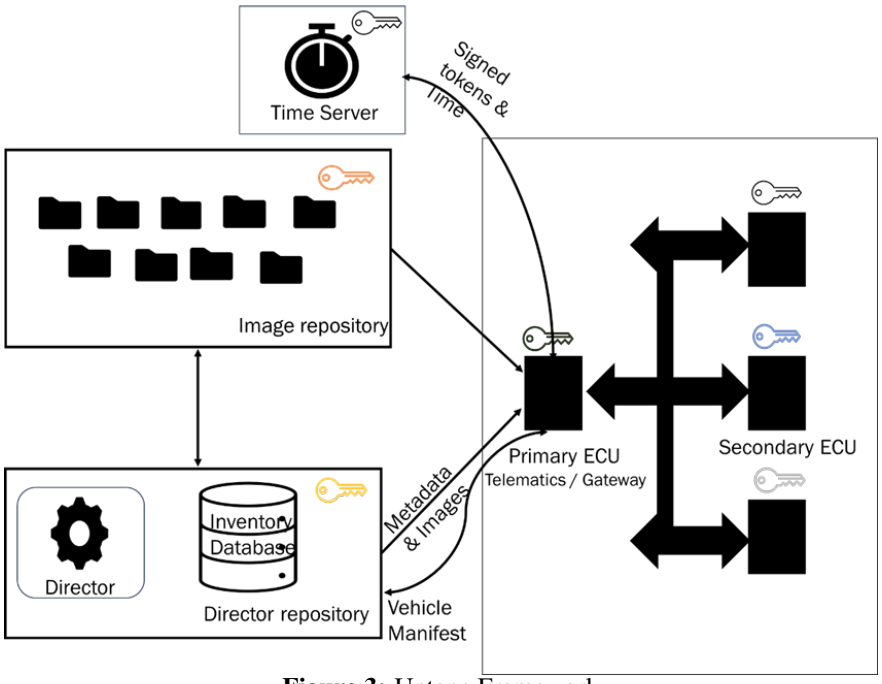

Figure 3: Uptane Framework

overflows and system crashes. *Mixed-bundle attacks* introduce system incompatibilities by installing conflicting software versions simultaneously, and *mix-and-match attacks* use compromised keys to release harmful software combinations.

Control: This involves scenarios where attackers install their software on an ECU, gaining complete control over its functionality. This could allow attackers to manipulate vehicle behaviors such as altering speed controls, turning off safety features, or engaging vehicle systems at unauthorized times, presenting a significant threat to passenger safety and vehicle integrity. Uptane effectively addresses these threats through a security architecture that includes multiple layers of protection. Techniques such as digital signatures for verification, requiring numerous trusted sources for update approval, and isolating the duties of different ECUs enhance the security posture of automotive OTA systems.

Despite its comprehensive security measures, Uptane does not inherently protect against attacks on the SSC or build systems. This limitation has led to the development of Scudo, an integration of in-toto and the Uptane framework[42].

5.2. Scudo

Scudo is a framework that integrates Uptane with the in-toto SSC security framework, achieving end-to-end security guarantees that span from the initial development of automotive software to its final delivery[4]. By merging Uptane's robust mechanisms for secure software artifact delivery with in-toto's comprehensive supply chain verification during image development, SCUDO creates a unified threat model that addresses vulnerabilities at both ends of the update process.

To implement SCUDO effectively, the Uptane image repository is modified to store in-toto metadata alongside traditional software images, as shown in Figure 4[2]. When an image is uploaded, it must be accompanied by its corresponding in-toto metadata, which is recorded in the custom

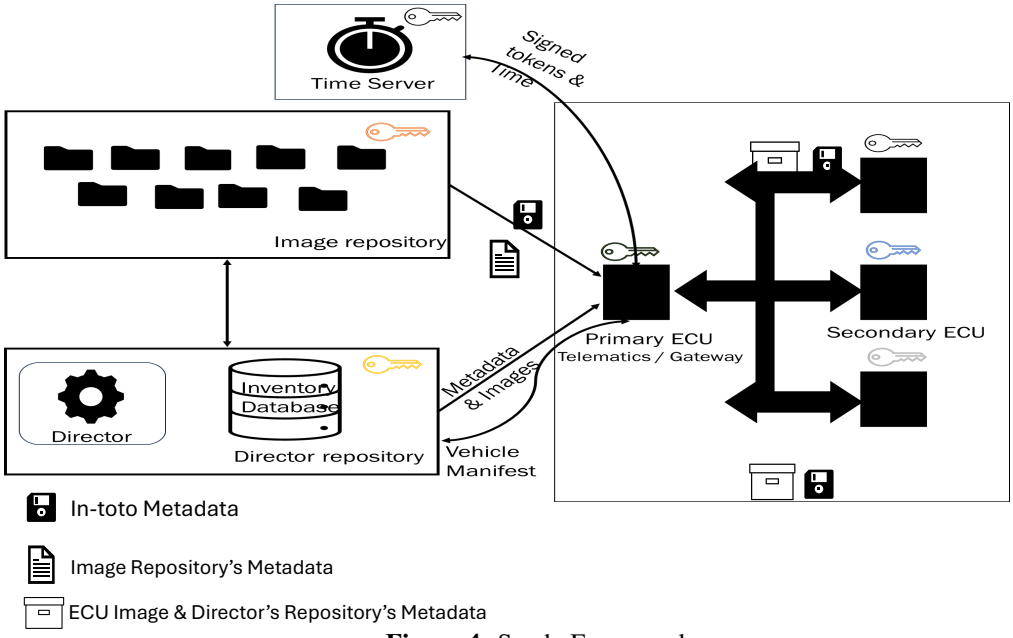

Figure 4: Scudo Framework

field of the image's entry in the target's metadata. This ensures no ambiguity regarding which in-toto metadata should be used during the retrieval and verification.

Given the practical challenge of issuing new metadata for every update, Uptane's delegation model is employed to enhance scalability. This model assigns delegated roles to sign the software image and its associated in-toto link metadata. Delegated roles help distribute the signing workload, reducing the burden on the primary signing authority, which ideally signs the in-toto layouts and key artifacts using offline keys. Once the Director repository selects and signs the necessary target metadata, the vehicle retrieves the images and their accompanying metadata through the prescribed interfaces of both repositories. The vehicle receives the Uptane metadata from the Image repository, the software image, and its in-toto metadata. The Primary ECU then disseminates these to secondary ECUs, which perform independent in-toto verifications to ensure that the update has not been tampered with, even if the Primary ECU is compromised.

For each target image, the client must install the corresponding in-toto layout, link metadata, and keys, which are identified via the custom field in the Uptane metadata. These in-toto artifacts are downloaded and verified by comparing their hashes with those specified in the Uptane metadata, which has been signed by either the Targets or delegated roles. This dual verification process extends Uptane's secure delivery properties to include the in-toto metadata, ensuring that the software image and its associated supply chain information are authentic and untampered. Moore *et al.* [42] implemented and evaluated Scudo in collaboration with Toradex. The latter is a manufacturer of embedded devices and provided a real-world platform conducive to testing the effectiveness of Scudo. The implementation's performance was gauged using metrics from the Toradex deployment. The verification process on the vehicle side through Scudo adds a manageable overhead of approximately 0.21 seconds per image, validating Scudo's feasibility for real-world applications.

Table 1: Metadata overhead on a vehicle of Scudo and Uptane [42].

Setup	Uptane Only	Scudo
Uptane	297.65 KiB	958.76 KiB
in-toto	-	6 KiB
Total	297.65 KiB	964.76 KiB
Overhead Percentage	0.155%	0.504%

Table 1 compares the metadata overhead before and after implementing SCUDO. In the original Uptane setup, the compressed Uptane metadata deployed by Toradex had a size of 297.65 KiB, accounting for approximately 0.155% of the total transmitted data (including both metadata and images). After integrating SCUDO, which incorporates in-toto metadata alongside Uptane's metadata, the total metadata size increased to 964.76 KiB of which 958.76 KiB is attributable to Uptane and 6 KiB to in-toto resulting in an overall overhead percentage of about 0.504%. Although this represents a roughly threefold increase in metadata size compared to the pre-SCUDO setup, the absolute increase remains minimal relative to the size of the transmitted software images. Despite these promising results, the evaluation faced limitations due to the singular use of the Toradex Colibri iMX7D v1.1B board, which may only partially reflect the diverse array of automotive ECUs that differ significantly in hardware capabilities and configurations. Additionally, ECUs sometimes have limited storage and processing capabilities, which can hinder the deployment of Scudo since they may demand additional storage and runtime resources that are not practical for many systems.

This limitation shows that Scudo is very effective in its tested environment. However, its applicability across diverse automobile circumstances may vary, which requires further testing and adaptation to meet broader industry needs.

Our proposed solution, **GuixChain**, replaces the in-toto component within the Scudo framework and builds on the existing capabilities of the Uptane framework, covering stages 2 – 4 (i.e., secure transmission, verification, and installation) of the update process. In addition, it introduces an alternative approach to address the enduring challenges in the software supply chain.

6. GUIXCHAIN

GuixChain offers a combination of technologies to enhance security and traceability throughout development to deployment. It integrates version control systems, blockchain technology, reproducible builds and software bill of materials to fortify the SSC:

Git Integration: By tracking changes in source code during software development, Git ensures that every modification is logged, providing an essential layer of documentation and control. This setup secures the code when integrated into the distributed ledger provided by blockchain technology. It enhances traceability and accountability, ensuring that each transaction or change can be independently verified and audited.

Blockchain Technology: As data is recorded in a ledger, a blockchain [43] records transactions between multiple nodes, ensuring that each transaction can be independently verified. It serves as an immutable audit trail, ensuring that every code change is permanently recorded and verifiable, crucial for tracing the origins of malicious components. This technology fosters a transparent and immutable data record maintained through consensus among network participants, providing a robust defense against tampering and malicious alterations.

Guix for Reproducible Builds: Guix

emphasizes the importance of reproducible builds [44], whereby the same source code and build environment consistently yield identical binary outputs. This process is facilitated through strict control over the build environment, including aspects such as compilers and libraries. Guix's approach supports cross-compilation, generating software for various target platforms, and maintains transparency by defining all build processes in plain Scheme code [45]. Reproducible builds are crucial for security because any deviation in the final output indicates a potential compromise or unauthorized modification in the build process, such as malware injection. By verifying that all builds produce the same object code, we can confidently detect anomalies and safeguard against tampering.

Software Bill of Materials (SBoM): In response to growing concerns about software security, US Executive Order 14028 [23], issued in spring 2021, mandates significant improvements in software development, testing and distribution processes. This order requires all software producers to provide an up-to-date SBoM that includes details on commercial, open-source, and off-the-shelf software components. In addition to emphasizing the importance of SBoM, Doug Bush, the Army's chief acquisition official, issued a memo mandating the incorporation of SBoM into new software-related contracts by February 2025 [46]. This policy shift highlights the critical role of SBoMs in improving transparency and accountability in SSCs, enabling better risk management and response strategies against vulnerabilities.

The operational framework of GuixChain is further enhanced by the development of smart contracts that automate and secure the build processes. These smart contracts ensure that build operations are triggered only by verified actions, such as initiating the source code compilation, which creates reproducible builds in a controlled environment. This setup, as seen in Figure 5, aims to mitigate the risk of attacks, which would now require simultaneous corruption across multiple systems, a significantly more challenging feat than exploiting a single vulnerability in an isolated system.

The objectives of GuixChain are as follows:

Integrate Git with blockchain: Enhance the security of code repositories by integrating Git with blockchain to ensure traceability and immutability in the software development process.

Develop smart contracts for automated code compilation: Create smart contracts that automate the code compilation process, ensuring that only verified actions can trigger these operations.

Implement reproducible and deterministic software builds: Establish a reproducible build environment to produce deterministic software builds, minimizing variations and ensuring consistency across different systems.

Create an SBoM: Implement a comprehensive SBoM during compilation to enhance transparency and security. GuixChain implementation strategy is grouped into 3 stages, they are :

6.1. Stage 1: Integrating Git with Blockchain

This integration improves the security of software supply chains by eliminating a single point of failure. By reducing reliance on centralized servers, the system mitigates risks such as data loss or downtime from threats including DDoS attacks, ransomware, and other malware. The distributed ledger acts as a redundant version control system, complementing Git by addressing its inherent limitations. While Git provides

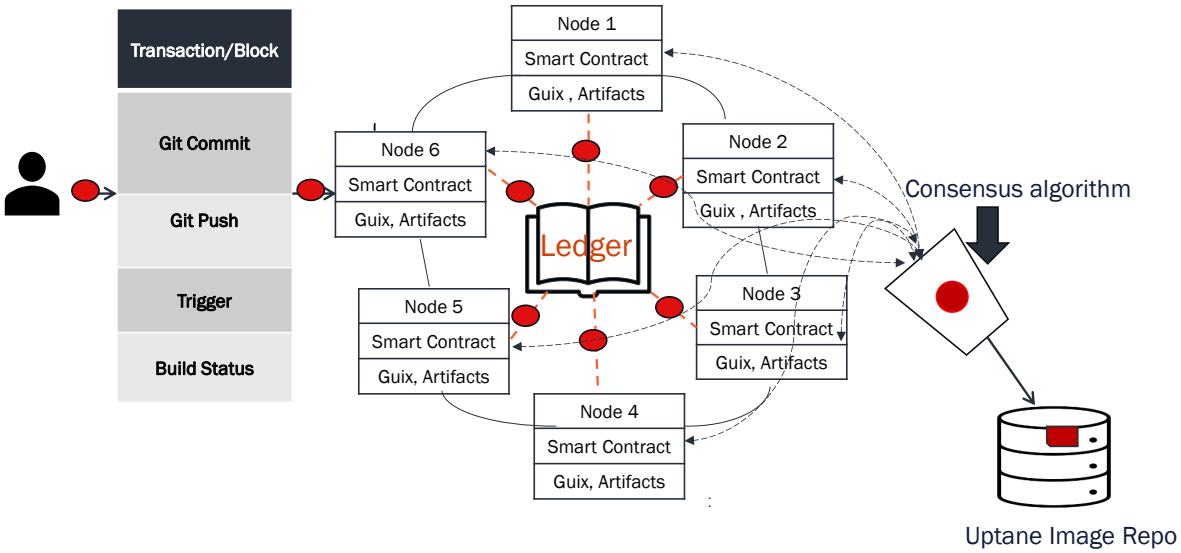

Figure 5: GuixChain Framework

comprehensive versioning for source code with features like branches, merges, and history tracking, it lacks the immutability, transparency, and distributed trust essential in multi-stakeholder environments. The immutable nature of the blockchain ledger ensures that every change to the software is transparent and verifiable, which is essential for tracing the origins of any malicious components. With this merger, we achieve a system in which changes are tracked and permanently recorded on a decentralized ledger, enhancing auditability and security throughout the supply chain [43].

After evaluating various blockchain technologies, we selected Hyperledger Fabric (HLF) due to its permission-based architecture, modular design, and high configurability [47], making it ideal for secure SSC management. HLF is well-suited for enterprise applications that require a trusted network among known participants. Its ability to facilitate private and confidential transactions ensures that only the involved parties can access transaction details, thus maintaining privacy and confidentiality. Additionally, HLF introduces the concept of channels ,private sub-networks within the larger network which enable specific groups of participants to create separate transaction ledgers accessible only to them.

HLF employs chaincode to define business logic and automate processes. Chaincode is comparable to smart contracts on other blockchain platforms and can be written in various programming languages such as Go, JavaScript, TypeScript, and Java, offering flexibility and customization. A smart contract is encapsulated within a chaincode and outlines the rules governing interactions between different organizations in executable code. These smart contracts define and execute rules/logic on the blockchain in a deterministic and decentralized manner. For example, consider a smart contract function `recordCommit` that accepts a Git commit hash, timestamp, and author information, and writes these details to the blockchain ledger. Another function, `verifyCommit`, can be used to compare a stored commit hash with a newly submitted one to ensure the integrity of the commit before triggering further processes, such as automated builds.

Security features within the smart contract

prevent unauthorized access and ensure that it can only be triggered by verified actions, maintaining consistent input, output, and ledger state. Each chaincode is associated with an endorsement policy, which specifies the organizations in the blockchain network that must endorse a transaction generated by a given smart contract to be considered valid. The endorsement policy is crucial, as it ensures that transactions meet the necessary approval criteria before execution.

To validate our approach further, we designed an experimental framework as part of the overall GuixChain system to test the complexity and performance of integrating Git with HLF. This framework is an integral component of GuixChain, used specifically to evaluate performance metrics such as synchronization time, consistency across nodes, security robustness, and resource utilization. We plan to explore three scenarios:

First, the **Independent Final Clone and Commit Hash Storage** scenario evaluates the ability of each repository clone to operate independently by securely storing commit hashes on the blockchain. This ensures that even if one node fails, the integrity of the commit history is maintained.

Second, the **Synchronized Repositories with Differential Commit** scenario tests the synchronization of repositories that have differing commit histories. This scenario focuses on merging only the differential commits, which is crucial for ensuring consistency across nodes without unnecessary redundancy or data transfer.

Third, the **Automated Synchronization with Direct Push** scenario assesses the automation of the synchronization process by enabling direct push mechanisms. This approach reduces manual intervention, streamlines updates, and verifies that the synchronization process is both secure and efficient.

These experiments will comprehensively evaluate the integrated system's performance and reliability, ensuring that the combined solution of Git and Hyperledger Fabric provides decentralized security and redundancy for the automotive software supply chain.

6.2. Stage 2: Enhancing reproducibility with blockchain and SBoM generation

Integrating blockchain with a reproducible build environment such as Guix ensures precise control over software environments, supporting cross-compilation and producing byte-for-byte identical outputs at each network node. Smart contracts trigger parallel builds across multiple nodes, ensuring that outputs are bit-wise identical and enhancing the build process's reproducibility and reliability. This setup guarantees deterministic builds and provides a transparent and auditable trail of all software versions and their corresponding builds [48].

An SBoM is generated in the compilation phase along with the firmware. This SBoM is parsed to catalog all materials used in the firmware, enhancing security during the consensus process. Here, images, metadata, and SBoMs, collectively referred to as artifacts in Figure 5, are hashed and compared across all nodes. Transmission to the Uptane image repository occurs only if the hashes match across the nodes, ensuring uniformity. Discrepancies in the output hashes prevent the dispatch of the artifacts, thereby safeguarding against unauthorized modifications. This verification process ensures that validated and consistent builds are advanced to the deployment stage.

6.3. Stage 3: Integrating Guix with the Uptane image repository

After the byte-wise comparison, the artifact is transferred to the Uptane image repository for storage. By requiring consensus among multiple nodes prior to transmission, the integrity of the artifacts stored in the Uptane repository is better protected. This stage provides a secure location for end users to access and download the build, which is then distributed to the appropriate ECU using the Uptane framework.

7. GUIXCHAIN THREAT MODEL

This section presents the GuixChain threat model, which outlines some cybersecurity risks that can compromise the software development and deployment processes in the automotive sector. These risks include ransomware attacks, insider threats, and build server compromises, each posing significant threats to operational continuity.

Ransomware Attacks: Ransomware can severely disrupt operations by encrypting key software assets on build servers, thus halting production and updates unless a ransom is paid. Integrating blockchain with Git in our GuixChain system ensures that every software version and subsequent updates are stored on an immutable ledger. This redundancy allows for a swift reversion to secure, pre-attack states, thus neutralizing the impact of ransomware without necessitating ransom payments.

Arbitrary Edits Attacks: Unauthorized modifications, whether by insiders or through compromised credentials, can introduce vulnerabilities such as backdoors into the software. GuixChain utilizes blockchain to meticulously track and validate every change made in the git repository. Smart contracts ensure that changes are only merged after passing security checks, making unauthorized edits detectable and reversible.

Build Server Compromise: If attackers gain control of the build server itself, they could inject malicious code that would propagate to every vehicle updated with the compromised software. Our approach uses deterministic builds triggered by smart contracts, ensuring that the same source code produces identical binary output across different nodes. This consistency allows for the detection of tampered binaries, as any unauthorized modifications on the build server would result in mismatched outputs across nodes. Every build process is also logged on the blockchain for enhanced auditing and verification.

Malware Injection in the Build Process: Beyond compromising the build server, malicious code can also be introduced at other points in the build process, such as through infected dependencies or a developer's compromised environment. The reproducible build environment guaranteed by Guix ensures that any anomalies in build outputs are immediately apparent across multiple independent builds, allowing for quick intervention before compromised software is distributed. While a build server compromise is one way malware can be introduced, this category addresses a broader range of injection vectors that might not require direct control of the build server.

7.1. Differentiating GuixChain from Existing Frameworks

GuixChain distinguishes itself from existing solutions such as FISHY, SBOMs, Sigstore, Grafeas, and in-toto (discussed in Section 4) by integrating blockchain technology with reproducible builds. While each framework offers valuable security features, GuixChain's unique combination addresses SSC threats.

As shown in Table 2, existing solutions offer valuable security features tailored

Solution	Continuous Monitoring	Artifact Metadata Management	Secure Signing	Provenance & Integrity	Build enviroment Security
GuixChain	✓	✓	✓	✓	✓
FISHY	✓	✗	✗	✓	✗
SBOM	✗	✓	✗	✓	✗
Sigstore	✗	✓	✓	✓	✗
Grafeas	✗	✓	✓	✓	✗
In-toto	✗	✓	✓	✓	✗

Table 2: Comparison of Software Supply Chain Frameworks

to specific aspects of the SSC. However, GuixChain distinguishes itself by delivering a comprehensive suite of capabilities, especially the protection against build server compromise. It ensures the immutability and traceability of code changes, guarantees that identical source code produces the same binaries in parallel build environments and ensures the correct image is sent to the uptane repository.

FISHY focuses on securing Internet of Things (IoT) components and ECUs within vehicles. While FISHY provides robust lifecycle protection, GuixChain offers immutable, transparent, and verifiable records of software changes and updates, fostering a more decentralized trust model not central to FISHY.

SBoMs provide detailed inventories of software components. GuixChain enhances traditional SBOM usage by embedding them within a blockchain, creating an immutable record. This adds an additional layer of security, enabling package validation before image deployment to the Uptane repository.

Sigstore and Grafeas offer solutions for software signing and metadata management, respectively. Sigstore simplifies software artifact signing and verification, while Grafeas provides granular control over software artifact metadata. GuixChain goes beyond artifact management by securing the build environment itself and ensuring reproducibility. This approach, which encompasses both artifact and environment security, surpasses the capabilities of Sigstore and Grafeas by securing the entire software development and deployment pipeline.

In-toto, similar to GuixChain, focuses on securing the software supply chain but focuses primarily on the integrity of development steps. GuixChain integrates these steps into a blockchain environment, verifying products and processes. GuixChain's use of smart contracts to automate and secure these processes offers a proactive approach to preventing unauthorized changes, a capability beyond the current scope of the in-toto.

Unlike traditional frameworks that often rely on the integrity of a single build server, GuixChain ensures identical binary outputs from the same source inputs across multiple build servers. This reproducibility is crucial. Any unauthorized changes or malicious code injections at the build server level are automatically detectable. A compromised build server that produces a binary different from a secure environment immediately reveals the discrepancy.

Guix's reproducible builds, integrated into GuixChain, extend verification beyond simple artifact checking. By leveraging blockchain, each build's details are immutably recorded, enabling traceability and auditing of any deviation from the expected binary output. This transparency and accountability establish a robust foundation of trust in the software delivery process.

Furthermore, the deterministic nature of these builds, combined with the blockchain's decentralized ledger, eliminates reliance on individual build server security. This mitigates the impact of single points of failure and distributes trust across multiple nodes, enhancing overall system resilience against build infrastructure attacks.

7.2. Future Work

The potential of GuixChain, as outlined, requires further empirical validation to fully assess its viability and effectiveness. Future work will focus on fully implementing this proposed solution within a controlled environment to evaluate its practical implications and performance metrics.

8. CONCLUSION

This paper examined the challenges confronting the software supply chain in the automotive industry and evaluated existing solutions such as the Uptane and Scudo frameworks. Our analysis highlighted the strengths and limitations of these approaches, particularly in managing OTA updates and ensuring the integrity of the build environment.

We introduced GuixChain, a framework that integrates blockchain, smart contracts, SBoMs, and reproducible builds to address these shortcomings in existing solutions. As illustrated in Table 2, GuixChain uniquely combines continuous monitoring, artifact metadata management, secure signing, provenance, and build environment security, ensuring that every component within the supply chain is transparently documented, rigorously verified, and each software is consistently reproduced across all nodes.

Our preliminary study underscores the promising potential of GuixChain. However, further empirical validation is necessary to fully ascertain its performance, scalability, and effectiveness against emerging threats. Future work will focus on full-scale implementation and comprehensive framework evaluation. We are optimistic that the continued development and refinement of GuixChain will enhance the security of automotive software systems, paving the way for more resilient and trustworthy supply chain practices.

References

[1] M. S. Melara and M. Bowman, "What is software supply chain security?" *arXiv preprint arXiv:2209.04006*, 2022.

[2] T. Karthik, Kuppusamy, and D. McCoy, "Uptane: Securing software updates for automobiles," 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:26794761.

[3] 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)*, 2019, pp. 1393–1410.

[4] M. Moore, A. S. A. Yelgundhalli, T. K. Kuppusamy, S. Torres-Arias, L. A. DeLong, and J. Cappos, *Scudo: A proposal for resolving software supply chain insecurities in vehicles*, 2022.

[5] D. Spinellis, "Git," *IEEE software*, vol. 29, no. 3, pp. 100–101, 2012.

[6] B. Xia, T. Bi, Z. Xing, Q. Lu, and L. Zhu, "An empirical study on software bill of materials: Where we stand and the road ahead," *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pp. 2630–2642, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:255825791.

[7] D. Arthur, C. Becker, A. Epstein, B. Uhl, S. Ranville, *et al.*, "Foundations of automotive software," United States. Department of Transportation. National Highway Traffic Safety …, Tech. Rep., 2022.

[8] A. Chawan, W. Sun, A. Javaid, and U. Gurav, "Security enhancement of over-the-air update for connected vehicles," in *Wireless Algorithms, Systems, and Applications: 13th International Conference, WASA 2018, Tianjin, China, June 20-22, 2018, Proceedings 13*, Springer, 2018, pp. 853–864.

[9] S. S. Kute and S. D. Thorat, "A review on various software development life cycle (sdlc) models," *International Journal of Research in Computer and Communication Technology*, vol. 3, no. 7, pp. 778–779, 2014.

[10] Siemens, *Oems, suppliers, and how to use a program management system*, https://blogs.sw.siemens.com/thought-leadership/2018/12/20/oems-suppliers-and-how-to-use-a-program-management-system/, Accessed: 2024-04-27, Dec. 2018.

[11] CMSTAT, *Configuration management across multi-site aerospace & defense suppliers – part 1*, https://cmstat.com/cmsights-news-posts/configuration-management-across-multi-site-aerospace-defense-suppliers-part-1, Accessed: 2024-04-27, Dec. 2020.

[12] J. J. Martinez and J. M. Durán, "Software supply chain attacks, a threat to global cybersecurity: Solarwinds' case study," *International Journal of Safety and Security Engineering*, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:244056951.

[13] P. Ferreira, F. Caldeira, P. Martins, and M. Abbasi, "Log4j vulnerability," in *International Conference on Information Technology & Systems*, Springer, 2023, pp. 375–385.

[14] S. Snider, "Massive okta breach: What cisos should know," *InformationWeek*, Dec. 2023, Accessed: February 18, 2025. [Online]. Available: https://www.informationweek.com/cyber-resilience/massive-okta-breach-what-cisos-should-know.

[15] D. Bradbury, "Tracking unauthorized access to okta's support system," *Okta Security Blog*, Oct. 2023, Accessed: February 18, 2025. [Online]. Available: https://sec.okta.com/articles/2023/10/tracking-unauthorized-access-oktas-support-system/.

[16] D. Bradbury, "Okta october 2023 security incident investigation closure," *Okta Security Blog*, Feb. 2024, Accessed: February 18, 2025. [Online]. Available: https://sec.okta.com/articles/harfiles/.

[17] Sonatype, *9th annual software supply chain*, , [Online; accessed 04-October-2024], 2023.

[18] K. Grimm, "Software technology in an automotive company - major challenges," *25th International Conference on Software Engineering, 2003. Proceedings.*, pp. 498–503, 2003. DOI: 10.1109/ICSE.2003.1201228.

[19] M. Broy, "Challenges in automotive software engineering," *Proceedings of the 28th international conference on Software engineering*, 2006.

[Online]. Available: https://api.semanticscholar.org/CorpusID:207159040.

[20] J. Soriano, G. Jiménez, E. Correa, and N. Ruiz, "Challenges in the automotive software supply chain, connected car : Benefits from an intent policy framework," *2021 IEEE 22nd International Conference on High Performance Switching and Routing (HPSR)*, pp. 1–5, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:236151472.

[21] W. Enck and L. Williams, "Top five challenges in software supply chain security: Observations from 30 industry and government organizations," *IEEE Security & Privacy*, vol. 20, no. 2, pp. 96–100, 2022.

[22] M. Broy, I. Krüger, A. Pretschner, and C. Salzmann, "Engineering automotive software," *Proceedings of the IEEE*, vol. 95, pp. 356–373, 2007. DOI: 10.1109/JPROC.2006.888386.

[23] The White House, *Executive order 14028 on improving the nation's cybersecurity*, https://www.whitehouse.gov/briefingroom/presidential-actions/2021/05/12/executive-order-on-improving-thenations-cybersecurity/, [Online; published 12-May-2021, accessed 24-Mar-2024], May 2021.

[24] J. Soriano, G. Jiménez, E. Correa, and N. Ruiz, "Challenges in the automotive software supply chain, connected car : Benefits from an intent policy framework," *2021 IEEE 22nd International Conference on High Performance Switching and Routing (HPSR)*, pp. 1–5, 2021. DOI: 10.1109/HPSR52026.2021.9481853.

[25] OpenSSF, *Open source security foundation*, https://openssf.org/, Accessed: 2024-04-27, 2024.

[26] P. Buxmann, A. V. Ahsen, L. Díaz, and K. Wolf, "Usage and evaluation of supply chain management software – results of an empirical study in the european automotive industry," *Information Systems Journal*, vol. 14, 2004. DOI: 10.1111/j.1365-2575.2004.00172.x.

[27] M. Hossfeld, C. Ackermann, and C. Griffy-Brown, "A cyberphysical vehicle platform for the mobility of the future—creating new value networks and business models," *IEEE Engineering Management Review*, vol. 49, pp. 99–107, 2021. DOI: 10.1109/emr.2021.3117149.

[28] M. Fourné, D. Wermke, W. Enck, S. Fahl, and Y. G. Acar, "It's like flossing your teeth: On the importance and challenges of reproducible builds for software supply chain security," *2023 IEEE Symposium on Security and Privacy (SP)*, pp. 1527–1544, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:259371795.

[29] W. Scacchi and T. A. Alspaugh, "Securing software ecosystem architectures: Challenges and opportunities," *IEEE Software*, vol. 36, pp. 33–38, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:131776365.

[30] C. Lamb and S. Zacchiroli, "Reproducible builds: Increasing the integrity of software supply chains," *IEEE Software*, vol. 39, no. 2, pp. 62–70, 2021.

[31] Z. Newman, J. S. Meyers, and S. Torres-Arias, "Sigstore: Software signing for everybody," *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:253371237.

[32] *Sap integrated business planning (ibp)*, https://www.sap.com/products/integrated-business-planning.html, Accessed: 2025-02-27, 2023.

[33] *Oracle scm cloud*, https://www.oracle.com/scm/, Accessed: 2025-02-27, 2023.

[34] *Siemens teamcenter*, https://www.plm.automation.siemens.com/global/en/products/teamcenter/, Accessed: 2025-02-27, 2023.

[35] *Ptc windchill*, https://www.ptc.com/en/products/windchill, Accessed: 2025-02-27, 2023.

[36] T. Bi, B. Xia, Z. Xing, Q. Lu, and L. Zhu, "On the way to sboms: Investigating design issues and solutions in practice," *ACM Transactions on Software Engineering and Methodology*, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:258332158.

[37] M. Mirakhorli, D. Garcia, S. Dillon, *et al.*, "A landscape study of open source and proprietary tools for software bill of materials (sbom)," *ArXiv*, vol. abs/2402.11151, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:267750339.

[38] Y.-C. Chang, "How to use sigstore without sigstore," *IACR Cryptol. ePrint Arch.*, vol. 2023, p. 3, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:256361482.

[39] Grafeas, *Grafeas: The artifact metadata api*, https://grafeas.io/, Accessed: 2024-04-27, 2024.

[40] Uptane, *Uptane: Secure over-the-air software updates for automotive*, https://uptane.org/, Accessed: 2024-04-27, 2024.

[41] 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*, 2010, pp. 61–72.

[42] M. Moore, A. S. A. Yelgundhalli, and J. Cappos, "Securing automotive software supply chains," in *Proceedings of the Network and Distributed Systems Security Symposium (NDSS) 2024*, San Diego, CA, USA, 2024. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/vehiclesec2024-15-paper.pdf.

[43] K. R. K. Reddy, A. Gunasekaran, P. Kalpana, V. R. Sreedharan, and S. A. Kumar, "Developing a blockchain framework for the automotive supply chain: A systematic review," *Computers & Industrial*

Engineering, vol. 157, p. 107 334, 2021.

[44] *Reproducible builds*, https://reproducible-builds.org/, [Online; accessed August 2024].

[45] G. Guix, *Gnu guix*, https://guix.gnu.org/, Accessed: 2024-08-27, 2024.

[46] Department of the Army, Office of the Assistant Secretary of the Army (Acquisition, Logistics and Technology), *Assistant secretary of the army (acquisition, logistics and technology) software bill of materials policy*, Memorandum, SAAL-ZE, 103 Army Pentagon, Washington, DC 20310-0103, Memorandum for See Distribution, n.d. [Online]. Available: https://federalnewsnetwork.com/wp-content/uploads/2024/09/081624_Army_SBOM_Memo.pdf.

[47] E. Androulaki, A. Barger, V. Bortnikov, *et al.*, "Hyperledger fabric: A distributed operating system for permissioned blockchains," in *Proceedings of the Thirteenth EuroSys Conference*, ser. EuroSys '18, Porto, Portugal: Association for Computing Machinery, 2018, ISBN: 9781450355841. DOI: 10.1145/3190508.3190538. [Online]. Available: https://doi.org/10.1145/3190508.3190538.

[48] L. Courtès, "Building a secure software supply chain with gnu guix," *arXiv preprint arXiv:2206.14606*, 2022.

9. CONTACT INFORMATION

Iwinosa Aideyan
Ph.D. Student, Computer Engineering
Holcombe Department of Electrical and Computer Engineering
Clemson University
Riggs Hall
Clemson, SC.
iaideya@clemson.edu

10. ACKNOWLEDGMENT

This work was supported by Clemson University's Virtual Prototyping of Autonomy Enabled Ground Systems (VIPR-GS), under Cooperative Agreement W56HZV-21-2-0001 with the US Army DEVCOM Ground Vehicle Systems Center (GVSC).

11. ACRONYMS

ECU	Electronic Control Unit
SSC	Software Supply Chain
OEM	Original Equipment Manufacturer
SBoM	Software Bill of Material
TCU	Telematic Control Unit
UDS	Unified Diagnostic Services
OBD	Onboard Diagnostic
SDLC	Software Development Lifecycle
SDV	Software Defined Vehicle
JNDI	Java Naming and Directory Interface
OIDC	OpenID Connect
CLI	Command-Line Interface
HLF	Hyperledger Fabric