information

MDPI

Article

Graph-Based Automation of Threat Analysis and Risk Assessment for Automotive Security

Mera Nizam-Edden Saulaiman [1,*], Miklos Kozlovszky [2,3] and Akos Csilling [4,†]

[1] BioTech Research Center, Doctoral School of Applied Informatics and Applied Mathematics, Óbuda University, 1034 Budapest, Hungary
[2] John Von Neumann Faculty of Informatics, Institute of Biomatics, Óbuda University, 1034 Budapest, Hungary; kozlovszky.miklos@nik.uni-obuda.hu
[3] Applied Cyber-Medical Systems Research Team, Laboratory of Parallel and Distributed Systems, Institute for Computer Science and Control (SZTAKI), Hungarian Research Network (HUN-REN), 1518 Budapest, Hungary
[4] Academic Relations, Robert Bosch Kft., 1103 Budapest, Hungary; akos.csilling@hu.bosch.com
* Correspondence: mera.abbassi@stud.uni-obuda.hu
† Current address: ARM Hungary Kft., 1082 Budapest, Hungary; akos.csilling@arm.com

Abstract: The proliferation of cyber–physical systems in modern vehicles, characterized by densely interconnected Electronic Control Units (ECUs) and heterogeneous communication networks, has significantly expanded the automotive attack surface. Traditional Threat Analysis and Risk Assessment (TARA) methodologies remain predominantly manual processes that exhibit limitations in scalability, and comprehensive threat identification. This research addresses these limitations by developing a formalized framework for automating attack path analysis within the automotive architecture. While attack graph methodologies have demonstrated efficacy in conventional information technology domains, their application within automotive cybersecurity contexts presents unique challenges stemming from domain-specific architectural constraints. We propose a novel Graph-based Attack Path Prioritization (GAPP) methodology that integrates Extended Finite State Machine (EFSM) modeling. Our implementation employs the Neo4j property graph database architecture to establish the mappings between architectural components, security states, and exploitation vectors. This research contributes a systematic approach to automotive security assessment, enhancing vulnerability identification capabilities while reducing analytical complexity.

Keywords: automotive cybersecurity; attack graph modeling; threat analysis and risk assessment (TARA); ISO/SAE 21434; graph database; Neo4j

check for updates

Academic Editors: Dongkyoo Shin and Dongil Shin

Received: 9 April 2025
Revised: 13 May 2025
Accepted: 19 May 2025
Published: 27 May 2025

Citation: Saulaiman, M.N.-E.; Kozlovszky, M.; Csilling, A. Graph-Based Automation of Threat Analysis and Risk Assessment for Automotive Security. *Information* **2025**, *16*, 449. https://doi.org/10.3390/info16060449

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Today's vehicles have transformed from purely mechanical machines into sophisticated cyber–physical systems that increasingly resemble computers on wheels [1]. Inside these modern vehicles, hundreds of Electronic Control Units (ECUs) communicate constantly over internal networks, orchestrating everything from essential safety functions like steering and braking to convenience features like parking assistance and autonomous driving. These complex internal networks use specialized communication protocols such as CAN, Automotive Ethernet, and MOST to enable split-second coordination between components [2].

The automotive landscape is evolving further with emerging vehicle-to-everything (V2X) technologies including the vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communication technologies. These systems enable cars to "talk" directly with

each other and with smart infrastructure elements like traffic signals, providing real-time awareness of road conditions and surrounding traffic [3].

While these innovations promise to enhance safety and efficiency, particularly for autonomous driving systems, they also add new dimensions to vehicle security considerations that manufacturers and regulators must address. When security breaches occur in these interconnected systems, they are not merely technical problems, they represent genuine threats to human safety. Malicious actors who compromise vehicle systems could potentially manipulate critical functions like steering, acceleration, or braking, putting the lives of passengers, pedestrians, and other drivers at immediate risk [4]. A comprehensive approach to security engineering is necessary throughout the vehicle development process, including design, production, operation, and decommissioning, which is addressed in standards such as the ISO/SAE 21434 [5].

In prior work, we introduced the concept of the Graph-based Attack Path Prioritization Tool (GAPP) to automate the generation of attack paths [6]. The current work extends this foundation in several aspects that differentiate our approach from existing methodologies: (1) our approach specifically addresses the unique constraints of automotive E/E architectures through ECU-specific privilege models and communication-aware state transitions; (2) we implement formal Extended Finite State Machine (EFSM) modeling to capture hierarchical states within automotive components, providing significantly greater modeling fidelity than existing approaches; and (3) we develop a Neo4j database model that enables automated TARA processes fully integrated with ISO/SAE 21434 workflows, addressing the implementation gap in existing automotive cybersecurity tools.

The concept and the methods are designed to integrate directly into the TARA process as defined in the ISO/SAE 21434 standard, aligning with industry standards and practical security analysis requirements. The rest of the paper is structured as follows:

Section 2 establishes the foundational cybersecurity context by examining regulatory frameworks within the automotive domain, specifically the ISO/SAE 21434 standard and UNECE R155 regulation. Section 3 synthesizes current research contributions in automotive cybersecurity modeling, analyzing attack graph methodologies and vulnerability assessment approaches. Section 4 introduces GAPP methodology, delineating formal specification models, state–transition systems, and algorithmic approaches for integrating attack graph construction within standardized TARA workflows. Section 5.1 presents the reference example system from the ISO/SEA 21434 [5] standard and guides the reader through the application of the GAPP methodology to this system. The limited complexity of this example allows a clear demonstration of the process. Section 5.2 presents the technical implementation architecture based on the Neo4j graph database platform, describing data model instantiation, relationship schema formalization, and traversal algorithm performance through a reference implementation applied to the sample system. Section 6 provides an evaluation and a discussion about the proposed methodology. Section 7 presents conclusions and summarizes the contribution of this work, and presents the future research areas.

2. Background

The automotive industry has a well-established history of addressing safety risks through mandatory regulations and technical standards, such as ISO 26262 [7]. The same approach is now being applied to the emerging challenges of cybersecurity incidents in modern vehicles. The increasing reliance on software and connectivity in vehicles has blurred the lines between safety and security, making robust cybersecurity an indispensable part of modern vehicle engineering.

The UNECE R155 [8] requires vehicle manufacturers to consider cybersecurity during the development of new vehicles as part of the approval process.

2.1. UN Regulation No. 155—Cybersecurity and Cybersecurity Management System

UN R155 is a regulation established by the United Nations Economic Commission for Europe (UNECE) focusing on cybersecurity and Cybersecurity Management Systems (CSMSs) for vehicles. Its primary goal is to establish a standardized approach for securing connected vehicles throughout their lifecycle, thereby increasing their resilience against cyberattacks. A core requirement of UN R155 is that vehicle manufacturers must implement a robust CSMS that includes the risk-based consideration of cybersecurity along the product lifecycle.

Manufacturers are required to conduct comprehensive risk assessments to identify potential vulnerabilities and attack vectors by analyzing the vehicle software, hardware, and network architecture. Based on these assessments, they must implement appropriate security measures, such as secure coding practices, encryption, access control, and vulnerability management. Furthermore, a plan for handling cybersecurity incidents must be in place, including procedures for timely detections, investigations, and responses to cyberattacks. UN R155 includes a list of 32 common threat categories and 23 mitigations in its annex to be considered during the risk assessment [8].

This regulation is supported by the ISO/SAE 21434 standard as a mean to demonstrate compliance [5]. The ISO/SAE 21434 standard, published in 2021, provides a framework for cybersecurity engineering in the automotive industry. It aims to ensure that electrical and electronic (E/E) systems within road vehicles are developed with appropriate consideration for cybersecurity, keeping pace with evolving technology and attack methods.

A core element of the ISO/SAE 21434 standard is the Threat Analysis and Risk Assessment (TARA), used as a means to evaluate the risk of future security incidents and make risk-based decisions during the development and approval process.

2.2. Threat Analysis and Risk Assessment

The ISO/SAE 21434 standard [5] provides a standardized framework for conducting TARA in the automotive industry [1]. It outlines a systematic process for identifying and assessing potential threats, vulnerabilities, and their associated risks. Traditionally, TARA is performed manually by security experts: experts conduct the analysis and define threat scenarios based on their domain-specific knowledge. We will provide an overview of the TARA process as specified in ISO/SAE 21434 and shown in Figure 1.

TARA starts with defining the item under evaluation and the boundaries of its relevant operational environment. It provides a precise description of the scope of the analysis and the technical aspects of the Target of Evaluation (TOE). The item definition serves as the foundation for subsequent steps.

1. Asset identification: Assets are valuable components within a vehicle system that require protection from cybersecurity threats. Each asset has associated cybersecurity properties and potential damage scenarios. As shown in Figure 1, this phase corresponds to the “item definition and cybersecurity properties” box, which produces the assets and damage scenarios as outputs. The cybersecurity properties that must be protected according to ISO/SAE 21434 are as follows:
    - Confidentiality: ensuring that the information is accessible only to authorized entities.
    - Integrity: guaranteeing the accuracy and trustworthiness of information and systems.

- Availability: ensuring that authorized users have timely and reliable access to information and systems.

Damage scenarios describe potential negative outcomes that could occur if a threat compromises an asset's cybersecurity properties. These scenarios may include physical harm to road users in an accident, loss of intellectual property in a data breach, or operational disruptions affecting vehicle functionality.
In our methodology, we formalize this output using the asset model shown in Figure 2, which captures the main attributes of each asset within the defined TOE, including its name, associated security properties, and impact categories.

2. Threat scenario identification: Following the asset identification, the TARA process requires identification of the threat scenarios that could compromise the security properties of identified assets as specified in the ISO/SAE 21434 Section 15.4 [5]. As illustrated in Figure 1, this step builds directly upon the defined assets and their cybersecurity properties. The standard notes that additional information may be included, such as damage scenarios, technical interdependencies between assets, and attack surfaces. It also recognizes that a single damage scenario may correspond to multiple threat scenarios, and conversely, a threat scenario may lead to multiple damage scenarios.
In our methodology, we formalize these threats using the threat model defined in in Section 4.3. The UML class diagram represents threats aligning with the ISO requirements: the targeted asset (via the type attribute), the compromised security property (SP), and the cause of the compromise (represented by the STRIDE category and the threat identifier).
3. Impact rating: The TARA process requires an assessment of the potential impacts of damage scenarios as specified in ISO/SAE 21434 Section 15.5 [5]. As shown in Figure 1, this step evaluates the severity of consequences that could result from the successful exploitation of identified threats.
According to the standard's requirements [RQ-15-04], four impact categories are defined:

- Safety (S): Potential harm to persons.
- Financial (F): Monetary losses for stakeholders.
- Operational (O): Disruption to vehicle functions or operations.
- Privacy (P): Unauthorized access to or disclosure of personal data.

For each impact category, the standard [RQ-15-05] requires that the impact rating be determined as one of four levels:

- Severe: Catastrophic consequences requiring immediate response.
- Major: Significant impact on system operation or business.
- Moderate: Limited impact with containable consequences.
- Negligible: Minimal or no measurable impact.

In our methodology, we formalize these impact ratings in the asset model shown in Figure 3, where the impact attribute is represented as a composite structure containing both category and rating. We map these qualitative ratings to numerical values (severe: 2; major: 1.5; moderate: 1; negligible: 0) to facilitate quantitative risk assessment calculations later in the process.

4. Attack path analysis: Following impact rating, the TARA process requires an analysis of attack paths as specified in ISO/SAE 21434 Section 15.6 [5]. As shown in Figure 1, this step takes input from the previous steps and it analyzes how attackers can exploit vulnerabilities to realize the identified threat scenarios. According to the standard, an attack path is the combination of various attack actions that, when applied successively, lead to the threat scenario. The standard acknowledges that in

early development stages, attack paths may be incomplete or imprecise as specific implementation details may not yet be available. Much of the difficulty of the TARA process lies in the identification of the most relevant attack paths. Traditional manual approaches often struggle with the combinatorial complexity of potential attack sequences. Our GAPP methodology directly addresses this challenge by employing a graph-based approach to systematically identify and analyze attack paths. The formal model described in Section 4.4 provides the mathematical foundation for this analysis, while the implementation detailed in Section 5.2 leverages graph database technology to execute these algorithms.

5. Attack feasibility rating: The TARA process requires an assessment of attack feasibility as specified in ISO/SAE 21434 Section 15.7 [5]. As shown in Figure 1, this step takes the attack paths as input, and evaluates the difficulty an attacker would face when attempting to execute each identified attack path. According to the standard's requirements [RQ-15-10], each attack path must be assigned one of four feasibility ratings:
   - *High feasibility*: Indicates attacks requiring minimal effort.
   - *Medium feasibility*: Indicates attacks requiring moderate effort.
   - *Low feasibility*: Indicates attacks requiring substantial effort.
   - *Very Low feasibility*: Indicates attacks requiring extensive effort.

   The standard recommends three potential approaches for determining attack feasibility:
   - *Attack potential-based approach*: As recommended in [RC-15-12], this evaluates feasibility based on five core factors:
     - Elapsed time: Period required to identify and exploit the vulnerability.
     - Specialist expertise: Level of knowledge required.
     - Knowledge of the item or component: Specific information needed about the target.
     - Window of opportunity: Accessibility constraints.
     - Equipment: Tools required for the attack.
   - *CVSS-based approach*: As recommended in [RC-15-13], this uses the Common Vulnerability Scoring System (CVSS) [9] to determine attack feasibility based on exploitability metrics: attack vectors, attack complexity, privileges required, and user interaction.
   - *Attack vector-based approach*: As recommended in [RC-15-14], this evaluates feasibility based on the predominant attack vector used in the attack path.

   In our GAPP methodology, we implement the attack potential-based approach as described in Annex G.2 of the ISO/SAE 21434 standard. As described in Section 4.7, each threat is characterized using a 5-tuple of feasibility factors.
   For multi-step attack paths, we define the *affmax* aggregation function that determines the overall feasibility by selecting the maximum difficulty value for each factor across all threats in the path. This approach, formally defined in Section 4.7, acknowledges that a chain of attacks is at least as difficult as its most challenging component.
6. Risk value determination: The TARA process requires determining the overall risk value for each threat scenario. As illustrated in Figure 1, this step synthesizes previous assessments to produce risk metrics that guide security decision-making. ISO/SAE 21434 Section 15.8 [5] specifies that the risk value for each threat scenario is calculated based on the impact rating of the damage and the feasibility of the attack. For scenarios where multiple damage scenarios or impact categories apply, separate risk values may be calculated for each combination. In our GAPP methodology, we implement a risk calculation formula that aligns with these requirements while providing a quantitative foundation for automated analysis:

$$Risk = Impact \times Feasibility \quad (1)$$

where *Impact* represents the impact rating (as defined in Section 5.2.2), and *Feasibility* is derived from the aggregated feasibility factors using the *affmax* function.

7. Risk treatment decision: The final step in the TARA process defines appropriate risk mitigation strategies based on the evaluated risk values. This phase translates analytical findings into concrete security actions.

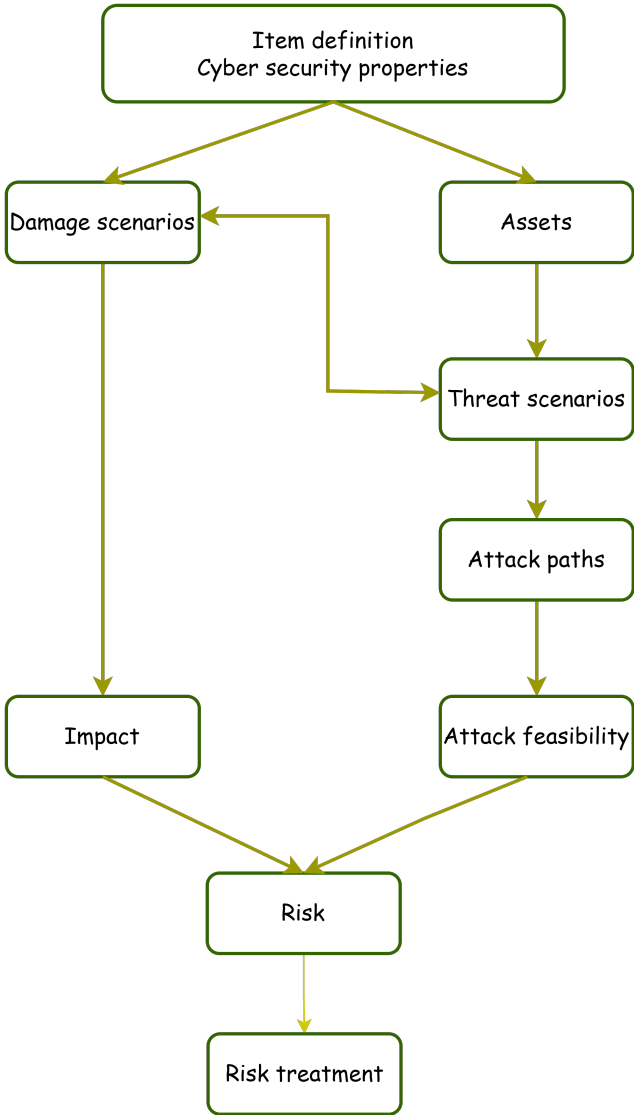

Figure 1. TARA process steps according to the ISO/SAE 21434 standard.

The TARA evaluation itself would be repeated multiple times iteratively during the design process until the residual risk after the implementation of all the defined security measures is accepted by the stakeholders and falls within acceptable thresholds.

The following section identifies the key challenges that arise when performing traditional manual TARA evaluations and demonstrates the need for automated approaches.

Challenges of Manual TARA

For complex systems, performing this process manually poses a number of challenges:

Time consuming and resource intensive:Traditional TARA processes often rely heavily on manual efforts, which can be time consuming and require significant expertise.

Prone to human error: The manual nature of these processes makes them susceptible to human error, potentially leading to incomplete or inaccurate evaluations.

Limited consideration of compound effects: Manual approaches often analyze vulnerabilities in isolation, overlooking the potential for multiple vulnerabilities to interact and create more severe security risks.

Subjective results: Any manual evaluation will naturally depend on the expertise of the people contributing. Therefore, the result of the evaluation will depend on subjective factors. These challenges provide a strong need for automation to reduce and eliminate as many manual steps as possible without compromising the validity of the results.

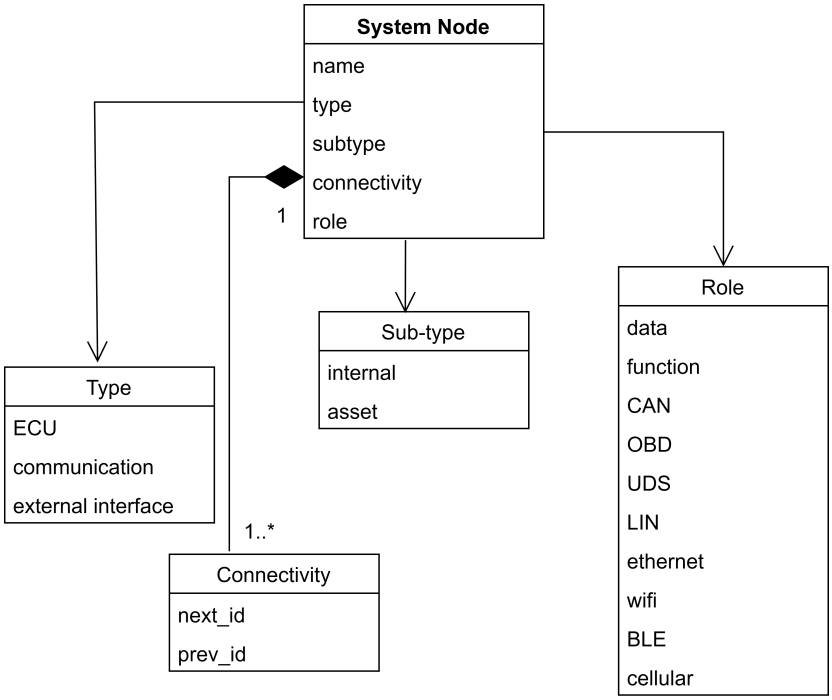

Figure 2. UML class diagram of the system model in GAPP.

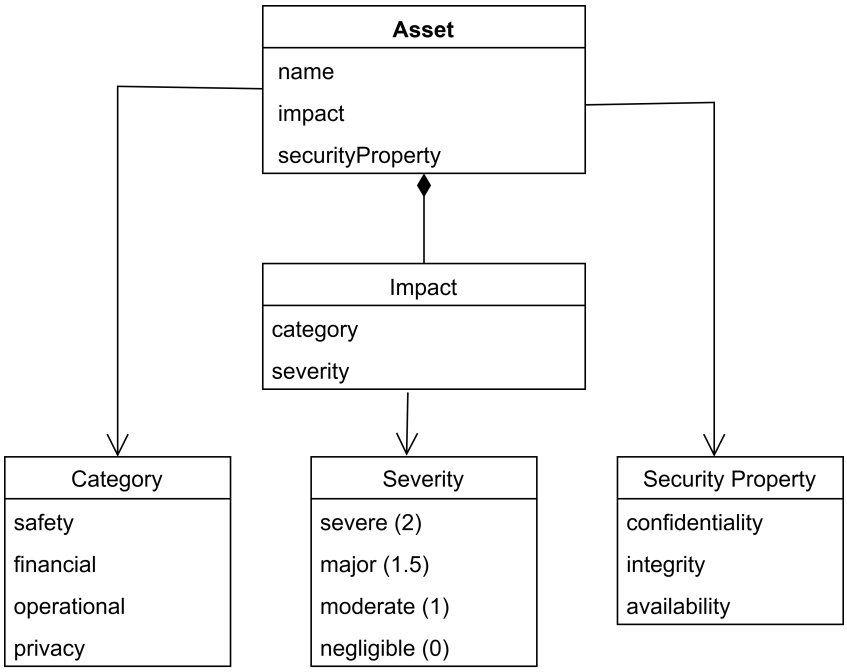

Figure 3. UML class diagram of the asset model in GAPP.

2.3. Automotive Network Security

Vehicles have evolved from purely mechanical machines into complex digital systems distributed over a heterogeneous network of ECUs. ECUs are designed to fulfill a certain set of functions, using input from and providing output to other ECUs. Most functions require low-latency real-time execution, with very short startup times, and the recurrent cost of the hardware limits the available processing power. The communication between ECUs still relies mostly on the CAN standard [10,11], introduced decades ago, with no security considerations at the time. The newer Automotive Ethernet standards provide more bandwidth and other benefits but still do not address security specifically [12].

Security considerations appeared along with the introduction of network segmentation by secure gateways. While the main functionality of the gateway is to bridge messages between the various CAN and Ethernet busses, adopting a white-list approach makes them an effective firewall. This is especially important with the external connections, including the onboard diagnostic (OBD) port and the telematics unit, providing broadband connectivity for cloud-based applications. Another important attack surface is presented by the infotainment unit, as it accepts connections from potentially compromised user devices [13].

One special consideration for vehicle security is the limited possibility of physical security. Even if we disregard the case when the owner is a willing participant in the attack, the fact that vehicles are often parked in the open allows physical attacks. A class of attacks observed in the wild starts by breaking the integrity of the physical protection to gain access to the internal network [14].

In the next chapter, we describe a new methodology to support the automation of the TARA by constructing an attack graph, as introduced in our previous paper [6]. Before that, however, we introduce some of the mathematical concepts used.

2.4. Graph Theory

Graph theory provides the mathematical foundation for modeling complex relationships between entities. At its core, a graph consists of a collection of elements, referred to as vertices or nodes, linked by connections called edges. Mathematically, we represent a graph G as the ordered pair (V, E) where V denotes the vertex set and E represents the collection of edges that establish connections between vertex pairs [15,16]. In its most basic form, an undirected graph represents symmetric relationships between objects, while a directed graph (digraph) uses arrows to indicate asymmetric relationships or flows. Edges may have associated weights representing costs, distances, or capacities [17,18].

Several fundamental properties characterize graph structures: vertex degree quantifies the number of edges connected to it; paths comprise sequential edge–vertex progressions connecting distant nodes; cycles constitute closed paths where initial and terminal vertices are the same; and connectedness describes whether navigational pathways exist between arbitrary node pairs [19]. Graph classification further distinguishes between a simple graph, where there is at most one edge joining each pair of distinct vertices, and no loops (edges joining a vertex to itself) are allowed, and more generalized structures that allow multiple parallel edges between vertex pairs and self-referential connections [20].

When implementing graph structures in computational systems, two primary data representation approaches predominate: the matrix-based adjacency representation, which utilizes two-dimensional arrays to indicate connection patterns between vertices, and the list-based adjacency structure, which maintains collections of neighboring vertices for each node in the graph. In our implementation, we will use the adjacency list. Each vertex in the graph is associated with a list of its neighboring vertices [19].

Several traversal algorithms can be employed to explore the connections within a graph [21]:

Depth-First Search (DFS) explores as far as possible along each branch before backtracking. DFS typically uses a stack data structure (or recursion) to keep track of the vertices to be visited [20].

Breadth-First Search (BFS) is a graph traversal algorithm that explores a graph level by level, starting from a given source vertex. It first visits all the neighbors of the source vertex, then all the neighbors of those neighbors, and so on. BFS uses a queue data structure to maintain the order of vertices to be visited [22].

Edges can have associated weights [23], and the weight represents a type of cost for traversing an edge. The shortest path in this case is the path with the minimum weight. BFS and DFS find the shortest paths in unweighted graphs, whereas for weighted algorithms, Dijkstra's algorithm and A* are used.

Dijkstra's algorithm is a greedy algorithm that finds the shortest paths from a single-source vertex to all other reachable vertices in a weighted graph where all edge weights are non-negative [24].

The A* algorithm is similar to Dijkstra's, but in addition it employs a heuristic function to make an informed guess for the next branch to explore [24]. At each step, it chooses the node that appears the best to reach the goal based on a guided decision using an evaluation function, usually referenced as $f(n)$ for a node n. The function is defined as the sum of two components [23]:

- g(n): The actual cost of the path discovered so far from the start node to node n.
- h(n): A heuristic estimate of the cost required to get from node n to a goal state.

Attack graphs build upon these fundamentals by representing potential paths an attacker might take through a system, with nodes typically representing system states or attacker privileges, and edges representing transitions between these states through exploitation of vulnerabilities.

2.5. Attack Graph

Attack graphs are graph-based security models (GrSMs) used to represent and analyze potential attack scenarios using a graph structure.

Attack graphs show various attack paths to a target by combining vulnerabilities, network connections, configurations, system states, exploits, and the relationships between all these elements [25,26]. Many variations in attack graphs exist. The initial technique in attack graph generation uses model checking, verifying logical statements against a given rule set or model. It uses state machines to test if a logical statement is true or false, and the counter examples of the statements are generated using the model checker to create the attack graph. The most common attack graphs are summarized in [25] as follows:

- State-based Attack Graphs (AGs): These structures provide formal representations of potential attack progressions, where vertices correspond to distinct system states within the security domain, and directed edges represent state transitions triggered by adversarial actions. Each vertex captures the security configuration at a specific point during attack execution, while edges point to the exploitation events that transform the system from one state to another [27].
- Attack Trees (ATs): Characterized by a hierarchical, tree-structured representation with a singular root node representing the adversary's ultimate objective. These models employ recursive decomposition to represent attack methodology, where sub-goals are organized through logical connectives, specifically conjunction (AND), disjunction (OR), and sequential conjunction (SAND), to encode the prerequisite relationships between attack components [28].

- Multiple Prerequisite Attack Graphs (MPAGs): These heterogeneous graphs implement a tripartite structure comprising distinct node classifications: state nodes (encoding attacker privilege levels within specific system components), prerequisite nodes (representing contextual conditions such as network reachability or credential possession), and vulnerability nodes (corresponding to exploitable security weaknesses within specific system interfaces) [29].
- Bayesian Attack Graphs (BAGs): These models extend traditional attack graphs by incorporating probabilistic inference mechanisms. Each vertex is augmented with conditional probability distributions that quantify exploitation likelihood based on multiple factors, including vulnerability characteristics, attacker capability profiles, and defensive control efficacy, enabling quantitative security risk assessment.

3. Related Work

The usage of attack graphs can be considered common in the IT domain, and there have been many approaches for utilizing attack graphs to visually represent the events that might facilitate an attack. One of the known tools in the research domain is the Multi-host Multi-stage Vulnerability Analysis (MULVAL) framework. MULVAL is a logical attack graph based on the Datalog modeling language, which is a subset of the Prolog logic programming language [30,31]. MULVAL requires input from multiple resources, and it requires security domain knowledge (typically using the CVE database), environment configuration data, security policies, and the reasoning rules [32]. MULVAL faces limitations when applied to an automotive security context as it primarily relies on network scanners such as Nessus to gather system configuration data. These scanner tools, designed for traditional IT infrastructure, are ineffective for automotive network architecture [24]. In addition MULVAL lacks inherent risk quantification [33], while researchers have tried to integrate quantitative assessment through Bayesian networks or CVSS-based metrics; these approaches encounter computational challenges as system complexity increases. Boudermine et al. demonstrated that Bayesian inference algorithms applied to complex attack graphs can exhibit exponential time complexity, making them impractical for comprehensive automotive systems [34]. Topological Vulnerability Analysis (TVA) presents an alternative approach, employing exploit dependency graphs with distinct node types representing attacks and security conditions. TVA's methodology simplifies computational complexity through the monotonicity assumption. However, TVA requires an extensive knowledge base encompassing detailed vulnerability information, attack techniques, precise precondition–postcondition relationships, and comprehensive network configuration details. Assembling such comprehensive knowledge for automotive systems presents significant challenges given the proprietary nature of vehicle technologies, architectural heterogeneity, and limited public documentation of vulnerabilities [31,35].

One main challenge for IT domain tools when applied to automotive systems is their dependence on vulnerability databases and network scanners. Salfer identifies that automotive ECUs lack adequate vulnerability database entries with CVSS ratings, and standard network scanning tools like nmap prove ineffective for automotive systems [24]. Yekta further observes that these databases predominantly contain IT-specific vulnerabilities irrelevant to vehicles while failing to adequately represent automotive-specific attack vectors, particularly those targeting specialized protocols such as CAN, FlexRay, and Automotive Ethernet, as well as the unique security challenges presented by converging IT and OT systems in modern vehicles [36]. This database deficiency constitutes a critical obstacle for successfully applying these attack graph methodologies to automotive cybersecurity.

The EVITA (E-safety Vehicle Intrusion Protected Applications) project focused on securing in-vehicle networks and communication interfaces between vehicles and infras-

tructure [37]. The project objective was to design, verify, and prototype a security architecture for automotive on-board networks. One of its contributions was the development of specialized risk metrics adapted from ISO 15408 [38], which provided a framework for automotive-specific threat assessment, EVITA also utilized attack trees to document and analyze potential threats to automotive systems [39]. Although EVITA can identify multi-stage attacks, it requires security analysts to manually create these attack paths [40]. Furthermore, as noted by Abouelnaga et al. [41], the assessment values used in EVITA's risk metrics do not comply with the current ISO/SAE 21434 standard, limiting their direct applicability in contemporary automotive cybersecurity engineering.

Salfer's research [24] highlights the need for automation in security analysis for vehicles' on-board networks, with particular emphasis on remote code execution scenarios. His approach derives the attack graph from a combination of a system model, an exploit model, and an attacker profile. Salfer's framework employs a stochastic risk quantification methodology incorporating probabilistic reasoning to evaluate attack feasibility, accounting for variables such as exploitation likelihood and required technical sophistication. A notable strength of Salfer's implementation is its performance efficiency compared to Bayesian networks. However, focusing primarily on the design phase, their approach does not fully address protection-related system artifacts like security controls that are components of the complete TARA process. Additionally, their methodology does not align with the standardized impact categories required by ISO/SAE 21434, limiting its direct applicability in regulatory compliance contexts.

Sommer et al. [42] investigate the topic of classification of automotive threats and vulnerabilities. They developed the Automotive Attack Database (AAD), which proposes a structured approach to categorizing attack vectors for the automotive domain. An important aspect of their approach is the development of a taxonomy that incorporates established categories like STRIDE within an automotive-specific context. We found their concept of "attack privilege" particularly useful, as it formalizes the levels of system access that attackers progressively achieve during multi-stage exploitations. Their Automotive Model-Based Security Testing (AMBST) methodology focuses on model-based security testing, enabling the automatic generation of attack paths to identify vulnerabilities, thereby supporting testing activities and vulnerability identification. GAPP leverages similar concepts such as the privilege model, however, GAPP's distinct contribution lies in its explicit design for formalized integration into the TARA process [5]. While Sommer et al.'s work primarily supports security testing and vulnerability discovery through model-based simulation and analysis of potential attack paths, GAPP focuses on automating the risk assessment aspects of TARA itself.

Complementary to graph-based vulnerability modeling, structured threat identification frameworks such as OCTAVE and STRIDE provide methodological approaches for systematizing security risk assessment [43,44]. While these frameworks do not inherently implement graph-based representations, they establish formal taxonomies and evaluation protocols that can be integrated with attack graph methodologies to enhance security analysis. STRIDE offers a categorical classification of security threats (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege), and it is often used in conjunction with other threat modeling techniques, like attack trees, to provide a more comprehensive view of potential attack scenarios [44]. STRIDE is recommended by the ISO/SAE 21434 standard as a potential threat modeling method. Karahasanovic et al. [44] presented a methodological adaptation of the STRIDE threat modeling framework tailored to automotive architectures. The authors discussed using the STRIDE approach to generate a list of threats based on Data Flow Diagrams (DFDs). The

STRIDE analysis was performed on the AUTOSAR platform using a template designed for the automotive systems.

GAPP incorporates STRIDE categorization and leverages structured threat information representation within its graph-based methodology. GAPP's distinct contribution lies in its design for automated attack path analysis and prioritization explicitly integrated into the TARA process, providing a systematic and repeatable approach aiming for compliance with automotive standards like ISO/SAE 21434.

4. GAPP Methodology

The Graph-based Attack Path Prioritization (GAPP) methodology automates the identification and analysis of attack paths by leveraging formal modeling techniques integrated with established automotive security frameworks. This approach addresses the growing complexity of vehicle systems where manual analysis becomes increasingly impractical [45]. GAPP enhances security assessment by systematically generating and prioritizing attack paths based on quantified risk values, providing security engineers with actionable insights for defense planning. GAPP directly implements the attack path analysis phase of the TARA process as defined in ISO/SAE 21434 [5]. Unlike general-purpose security analysis tools, GAPP is specifically designed to accommodate the unique constraints and requirements of automotive architectures. The methodology employs Extended Finite State Machines (EFSMs) to model system behavior under attack conditions. EFSMs extend traditional finite state machines with variables, guards, and actions [46], providing the expressiveness needed to represent complex automotive systems with hierarchical states. States in this model represent attacker privilege levels on specific system components, while transitions correspond to the exploitation of vulnerabilities that enable attackers to move between states. This formal representation allows for systematic exploration of the attack surface through graph traversal algorithms, identifying all viable paths from external interfaces to critical assets. In this article, in order to limit the complexity of the analysis, we consider the case when the TOE is an ECU, represented by one state machine, without consideration of its internal complexity.

4.1. Steps of the GAPP Methodology

This framework integrates directly with the established TARA process, as defined in ISO/SAE 21434 [5], with particular emphasis on enhancing the attack path analysis phase. The methodology aligns with security modeling approaches recommended by security researchers [47], while addressing automotive-specific requirements [48]. The methodology encompasses five sequential, interdependent analytical phases:

1. The process begins with the creation of a detailed system model representing the architecture of the TOE. This phase involves the instantiation of nodes representing ECUs, communication networks, and external interfaces, with the definition of their interconnections through a reachability list. The system model establishes the topological foundation for subsequent security analysis phases.
2. State model creation: Building upon the architectural foundation, this phase implements an EFSM framework to define the state space of the system. Each node's possible states are formally characterized based on attacker privilege levels, with corresponding transition functions.
3. Vulnerability mapping: Following the creation of the state model, this phase establishes the relationships between the states of the system and the applicable vulnerabilities. The mapping process utilizes a multi-dimensional classification schema incorporating STRIDE categories, component types, and privilege levels to systematically associate relevant threat vectors with specific system states.

4. Attack path traversal analysis: A core part of the methodology is the automated graph traversal and search algorithms that systematically identify all viable exploitation sequences from attack surfaces to critical assets.
5. Risk-based path prioritization: The final analytical phase implements a quantitative risk assessment for each identified attack path. Using the affmax aggregation function, the methodology calculates composite feasibility metrics across multi-stage attack sequences, combining these with the asset impact ratings to derive the risk scores in accordance with ISO/SAE 21434 criteria.

Each phase of the GAPP methodology implements formal security modeling techniques while maintaining alignment with established automotive cybersecurity engineering frameworks.

4.2. System Model

The system model is created from the TOE description, including internal components like ECUs, communication networks, and external components that interact with the system. The first step of the TARA process describes the system and the corresponding assets. As explained in Section 1, the asset node will have a corresponding security property that is required to be protected from attacks, in addition to defining the impact of the damage for this asset.

A system node $\mathcal{N}$ is formally defined as a 5-tuple:

$$N = (Name, Type, Subtype, Conn, Role) \tag{2}$$

where

Name is the name of the component within the TOE system model.

Type is an enumerated categorical value from the set {*ECU, Communication, External Interface*}, specifying the fundamental nature of the node.

$Subtype \in \{internal, asset\}$ differentiates nodes that represent critical system assets requiring protection from standard internal components.

$Role \in \{data, function, CAN, OBD, UDS, LIN, Ethernet, BLE, wifi, cellular\}$ specifies the functional classification of the node within the system architecture.

Connectivity defines a bidirectional adjacency relationship represented as

$$Connectivity = (Prev_id, Next_id), \tag{3}$$

where

$Prev_id \subseteq \mathcal{N}$ is the set of nodes that can initiate communication to this node.

$Next_id \subseteq \mathcal{N}$ is the set of nodes that can receive communication from this node.

The directionality constraints enforce that communication flows only between physically connected components and follows the established network topology. For ECUs, communication is permitted only through connected network interfaces. For communication networks, message flow follows the network protocol specifications.

Figure 2 presents the system model as a class diagram using standard UML notation. The diagram illustrates the central node class with its specialized types (ECU, Communication, External Interface). Attributes within each class correspond to the formal properties defined in our mathematical model. The UML notation elements used include: the asterisk (*) indicating multiplicity relationships representing 'zero or more' connections, the notation '1..*' specifying 'one or more' mandatory connections between nodes, and the solid diamond (rhombus) symbols representing composition relationships, indicating that the connected components are integral parts of the system node and cannot exist independently.

If the subtype of the system node is an asset, then it will inherit extra properties that are defined in the UML Figure 3. An asset A within the system is characterized by the 3-tuple:

$$A = (Name, Impact, SP), \tag{4}$$

where

Name represents the name of the asset.

Impact is the measure of the damage caused by the loss of the security properties of the asset. It is itself a composite structure, defined as

$$Impact = (Category, Rating), \tag{5}$$

where

$Category \in \{safety, privacy, operational, financial\}$ represents the domain affected by the damage.

$Rating \in \{severe, major, moderate, negligible\}$ quantifies the severity using the ISO/SAE 21434 rating scale [5].

SP stands for *Security Property* and defines the security goal to be protected.

Security property, $SP \in \{confidentiality, integrity, availability\}$.

In this article, we adopt a simplified security model where the compromise of a security property is considered to affect the entire component. For example, violating the integrity of the Headlight Unit means that all data on the Headlight Unit can be manipulated. This is a simplification, taken for ease of understanding. The separation into different assets, such as firmware or functions, is a natural extension, and indeed, a major benefit of the approach.

4.3. Threat Model

We associate threats with system model nodes and state nodes, using the type and privilege attributes of the given node, as well as its STRIDE category. As a result, each threat will be linked to the nodes in the system state model where the threat is applicable. The threat model TM is formally defined as a set of threats defined by the 6-tuple:

$$TM = (Threat, STRIDE, Type, Feasibility, Privilege, SP) \tag{6}$$

where

Threat refers to the identifier of the given threat. In our current implementation, we use the threats defined in the UN R155 regulation [8].

In practical applications, these can be replaced by threats coming from a specific database defined by the project or mapped from some common database, for example, the NVD [49].

STRIDE categorizes the threat according to the STRIDE model. The classification is shown in Table 2.

Type specifies the general type of the system node that the threat applies to.

Feasibility represents the ease with which an attacker could exploit the threat as discussed in Section 4.7.

Privilege is the privilege acquired by exploiting a vulnerability. A vulnerability can lead to more than one gained privilege; for example, full control also implies all the lower-privilege states. The attacker privilege model is described below in the section titled Attacker Privilege Model for Automotive Systems.

Security Property (SP) defines the security property that the threat breaks.

The UML class diagram in Figure 4 provides a visual counterpart to the formal mathematical model. It is well suited for our threat model representation as it captures the hierarchical classification of threats.

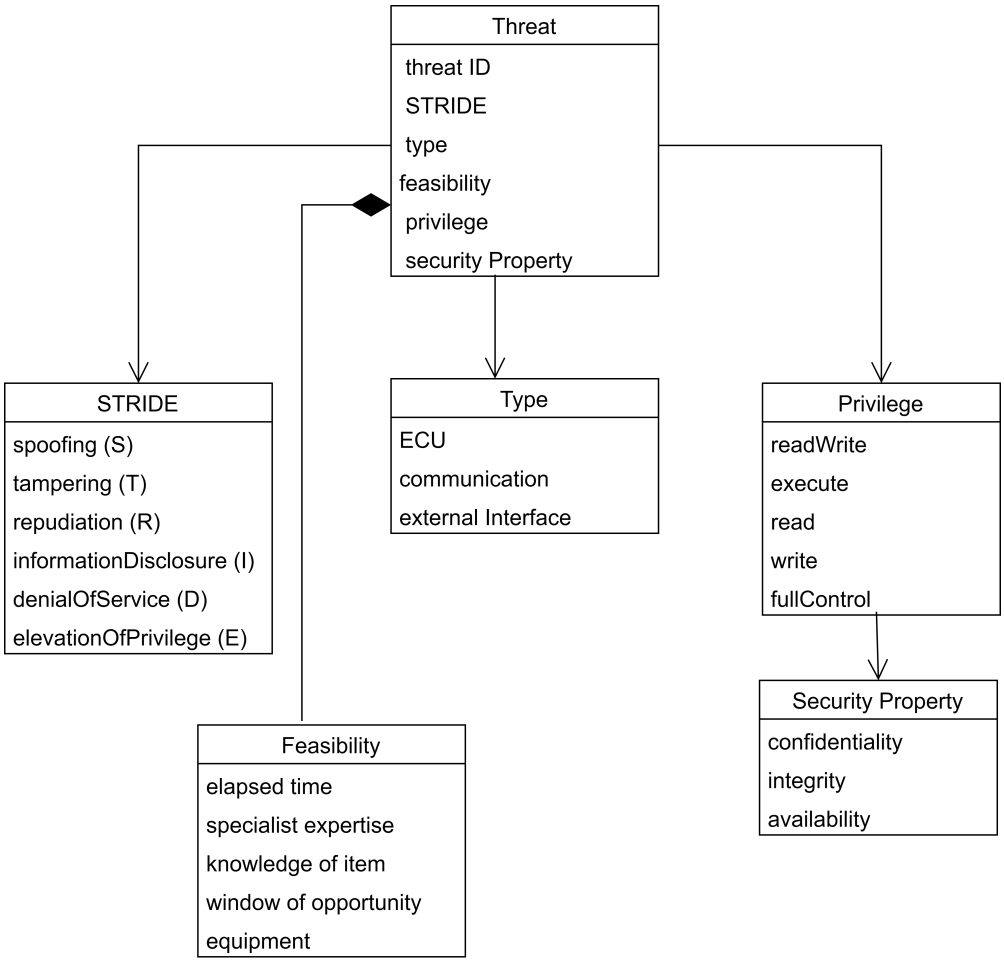

Figure 4. UML class diagram of the threat model in GAPP.

Attacker Privilege Model for Automotive Systems

The conventional access control paradigms utilized in information technology domains prove insufficient for modeling attacker capabilities within automotive architectures. While IT security models typically differentiate between "root access" and "user access" privilege levels, automotive systems require a more nuanced approach due to their heterogeneous ECU architecture, diverse communication protocols, and domain-specific functional constraints [50].

To address these domain-specific requirements, we adopt and extend the attacker privilege model introduced by Dürrwang et al. [40]. This model defines abstract privilege categories that characterize an attacker's operational capabilities within automotive system components, aligning with privilege classifications established in related work by Jürgen [51] and Sommer [42]. We have tailored this model to accommodate our system representation framework.

The privilege categories represent distinct attacker capabilities with corresponding security implications. This categorization aligns with observed real-world automotive attack methodologies documented in security research [14,52] and provides a more accurate representation of attacker progression than binary IT privilege models. These privileges depend on the type of the target node, reflecting the inherent differences between the exploitation of communication networks versus embedded controllers.

1. Read/Write (Communication Systems): Enables an attacker to intercept, monitor, and manipulate message traffic on communication channels, facilitating activities such as message replay, injection, or denial of service attacks.

2. Execute (ECU Systems): Provides capability to invoke specific functions implemented within an ECU, enabling operational control without necessarily exposing data or enabling arbitrary code execution.
3. Read (ECU Systems): Grants access to internal data structures, configuration parameters, or program memory without the ability to modify them. This supports reconnaissance activities and intellectual property extraction.
4. Write (ECU Systems): Allows modification of internal data structures, configuration parameters, or program memory, facilitating tampering and potentially enabling persistent modifications to ECU behavior.
5. Full Control (ECU Systems): Establishes comprehensive administrative access to the ECU, analogous to root-level access in conventional IT systems, enabling arbitrary code execution and complete system manipulation.

We formally define the attacker privilege model AP as the 2-tuple:

$$AP = (P, T_A), \tag{7}$$

where

$P \in \{readwrite, execute, read, write, fullcontrol\}$ represents the privilege level required to exploit the threat, as defined above.

$T_A : P \to \mathcal{P}(P)$ defines the transition function mapping a privilege to its potential subsequent privileges, where

$$T_A(readwrite) = \{execute, read, write, fullcontrol\} \tag{8}$$
$$T_A(execute) = \{readwrite, read, write, fullcontrol\} \tag{9}$$
$$T_A(read) = \{readwrite, write, fullcontrol\} \tag{10}$$
$$T_A(write) = \{readwrite, fullcontrol\} \tag{11}$$
$$T_A(fullcontrol) = \{readwrite\} \tag{12}$$

This privilege model enables systematic analysis of potential attack paths based on privilege escalation sequences. The interpretation is as follows: in order to exploit a given threat an attacker needs privilege level P on the node. After a successful attack, the attacker has access to the privilege levels listed in $T_A(P)$.

Note that the STRIDE categories and the privilege levels are not independent. The attacker privilege level determines which attack categories are possible; Table 1 maps these privilege levels to corresponding STRIDE threat categories.

Table 1. Automotive attacker privilege classification with corresponding STRIDE categories.

Privilege Level	STRIDE Categories
Read/Write (Communication Systems)	Spoofing, Tampering, Information Disclosure, Denial of Service (S, T, I, D)
Execute (ECU Systems)	Spoofing, Tampering, Repudiation, Denial of Service, Elevation of Privilege (S, T, R, D, E)
Read (ECU Systems)	Spoofing, Tampering, Repudiation, Information Disclosure, Elevation of Privilege (S, T, R, I, E)
Write (ECU Systems)	Spoofing, Tampering, Repudiation, Elevation of Privilege (S, T, R, E)
Full Control (ECU Systems)	Spoofing, Tampering, Repudiation, Elevation of Privilege (S, T, R, E)

4.4. State Security Model

Building upon the system model and attacker privilege model, we formulate a state security model that provides a formal foundation for attack path analysis. This model integrates both static architectural elements and dynamic behavioral aspects of vehicular systems through an EFSM formalism, enabling the systematic exploration of potential attack sequences.

4.4.1. Extended Finite State Machines for Automotive Security Modeling

The implementation of GAPP employs an EFSM formalism to model the dynamic behavior of automotive systems under attack conditions. EFSMs extend traditional finite state machines by incorporating variables, guards, and actions, providing enhanced expressiveness particularly suited to modeling complex automotive systems with hierarchical states and conditional transitions [46].

4.4.2. Formal Model Definition

The state security model M is formally defined as the 6-tuple:

$$M = (S, \Sigma_e, \Sigma_o, S_{\text{start}}, V, T), \tag{13}$$

where

S represents a finite set of states. Each state $s \in S$ is defined as a tuple

$$S = (Node, Privilege), \tag{14}$$

where

Node identifies the system component node from the system model, and

Privilege specifies the attacker's access level.

Σ_e constitutes a finite set of events representing threat vectors; in our implementation, these events are derived from the UN R155 catalog. Each event $e \in \Sigma_e$ corresponds to a specific exploitation technique.

Σ_o defines a finite set of outcomes manifested when transitions occur, representing security property modifications, impact, and feasibility metrics.

$S_{\text{start}} \subset S$ subset of states representing initial attack entry points, typically corresponding to external interfaces in their default privilege level.

V is a finite set of state variables that are available in every state $s \in S$. The state variables include the security property violated by the transition, the feasibility of the attack, and the impact on the security of the system. In the context of our security model, this set is defined as follows:

$$V = (Violated\ Security\ Property, Feasibility, Impact), \tag{15}$$

where

Violated Security Property $\in SP$ as defined in Equation (4).

Feasibility is a variable that represents the feasibility of executing the attack described by the transition.

Impact is a variable that represents the impact of the vulnerability being exploited on the security of the system. The impact may be empty, if the attack causes no damage to any of the assets in the system.

T in Equation (13) represents the set of state transitions. Each transition $t \in T$ is defined as

$$t \in T : s_1 \xrightarrow{e[g]/a} s_2, \tag{16}$$

where

$s_1, s_2 \in S$ are the source and target states.

$e \in \Sigma_e$ is the event triggering the transition.

g is the guard condition that must be satisfied.

$a \in \Sigma_o$ is the resulting outcome. The value for the impact may be empty if it is not an asset state.

In the model, transitions are represented as 3-tuples:

$$T = (Source, Destination, Threat), \tag{17}$$

where

$Source \in S$ is the origin state in the transition.

$Destination \in S$ is the target state after the transition.

$Threat \in \Sigma_e$ is the specific threat enabling the transition.

For a transition $t \in T : s_1 \xrightarrow{e[g]/a} s_2$ to be valid, the following guard conditions and constraints must be satisfied:

1. Type Compatibility: The *Type* of the *Destination Node* must be compatible with the *Type* of the *Threat*.
2. STRIDE Compatibility: The *STRIDE* category of the *Threat* must align with the *STRIDE* category of the *Privilege* level of the *Destination*, as defined in Table 1.
3. Connectivity Constraint: There must be a direct connection between the *Source* and *Destination System Node*s according to the *Connectivity* list.
4. Privilege Monotonicity: For transitions between nodes of the same type, the privilege level cannot decrease: $privilege(source) \leq privilege(destination)$.

The output action a produces state variable modifications reflecting the security impact of an asset:

- $ViolatedSecurityProperty := threat.SP$;
- $Feasibility := threat.Feasibility$;
- $Impact := destination.node.subtype == asset?destination.node.Impact : \emptyset$.

4.5. STRIDE Mapping Methodology and Security Property Correlation

The assignment of STRIDE categories to UN R155 threats in Table 2 was performed based on threat characteristics and potential attack mechanisms. The mapping process evaluated each threat against the following criteria:

1. Attack Method: Examination of the technical mechanisms typically employed in exploiting each threat.
2. Target Component: Identification of the primary system components affected.
3. Attack Outcome: Determination of the potential security violations resulting from a successful exploitation.

For example, UN-T6 (“Messages/data are modified in transit”) is classified as Spoofing (S) rather than Tampering (T) because the primary attack mechanism involves impersonating legitimate message sources rather than modifying existing communications. Similarly, UN-T9 (“Connection to unauthorized communication partners”) is mapped to multiple security properties (C, I, A) because unauthorized connections can simultaneously compromise confidentiality through data leakage, integrity through unauthorized modifications, and availability through service disruption.

The formal relationship between STRIDE categories and security properties follows a systematic mapping:

- Spoofing (S) → Confidentiality (C): Spoofing attacks primarily compromise authentication mechanisms, leading to unauthorized access and potential confidentiality breaches.
- Tampering (T) → Integrity (I): Tampering attacks directly target data or code integrity by unauthorized modification.
- Repudiation (R) → Integrity (I): Repudiation threats compromise accountability records, affecting system integrity. If actions cannot be reliably attributed, the trustworthiness of the system's logs and records is undermined.
- Information Disclosure (I) → Confidentiality (C): Information disclosure directly compromises confidentiality by exposing protected data.
- Denial of Service (D) → Availability (A): DoS attacks specifically target system availability.
- Elevation of Privilege (E) → Multiple (C,I,A): Privilege escalation can lead to multiple security property violations depending on the specific privileges gained.

It is important to note that the relationship between STRIDE and security properties is not always one-to-one. As noted with Elevation of Privilege, a single threat category can impact multiple security properties. Understanding this mapping helps define security requirements and identify appropriate countermeasures. If a system is vulnerable to spoofing, the security goal of authentication and the property of confidentiality need to be strengthened.

4.6. Threats Classification

To establish a foundation for our vulnerability database, we leverage the threats enumerated in Annex 5 of the UN R155 regulation. The regulation document contains 32 distinct threats, each accompanied by examples of potential attack methodologies [8]. This annex serves as a standardized baseline for automotive cybersecurity risk assessment and mitigation strategies. Our classification approach, presented in Table 2, integrates these threats within a structured taxonomy framework.

We analyzed these examples along with supporting clarifications from the regulation to systematically categorize each threat according to the STRIDE model.

The assignment of STRIDE categories (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege) reflects the fundamental attack methodology inherent to each threat. For example, attacks that primarily involve identity impersonation are classified under Spoofing, while those that focus on altering system data or behavior are categorized as Tampering.

Security properties (Confidentiality, Integrity, Availability) are assigned based on the direct impact of threat exploitation on the affected component.

The component type indicates the primary architectural element where the threat manifests:

- ECU: Electronic Control Unit components.
- C: Communication networks and interfaces.
- External Interface: Connection points to external systems or devices.

This taxonomy creates a framework for describing threats. By integrating these threat classifications within our graph database implementation, we enable querying and analysis of attack paths and potential vulnerabilities. Moreover, this approach facilitates the automated generation of security models and supports the development of appropriately targeted mitigation strategies.

Table 2 presents our mapping of UN R155 threats to their corresponding STRIDE categories, affected component types, and compromised security properties.

Table 2. Mapping of threats listed in [8] to STRIDE categories, node types, and the violated security properties. See the text for explanations of the abbreviations.

UN Threat	STRIDE	Type	SP
UN-T1: "Back-end servers used as a means to attack a vehicle or extract data"	T, I, E	ECU	C, I, A
UN-T2: "Services from back-end server being disrupted, affecting the operation of a vehicle"	D	ECU	A
UN-T3: "Vehicle-related data held on back-end servers being lost or compromised ('data breach')"	I, T	ECU	C, I
UN-T4: "Spoofing of messages or data received by the vehicle"	S	C	C
UN-T5: "Communication channels used to conduct unauthorized actions"	T	C	C, I
UN-T6: "Messages/data are modified in transit"	S	C	C, I
UN-T7: "Eavesdropping on vehicle communication channels"	I	C	C
UN-T8: "Denial of service attacks via communication channels to disrupt vehicle functions"	D	C	A
UN-T9: "Connection to unauthorized communication partners"	S	C	C, I, A
UN-T10: "Viruses embedded in communication media are able to infect vehicle system	T	C	C, I
UN-T11: "Messages received by the vehicle (for example X2V or diagnostic messages), or transmitted within it, contain malicious content"	T	C	C, I
UN-T12: "Misuse or compromise of update procedures"	T, E	ECU	C, I
UN-T13: "It is possible to deny legitimate updates"	D	ECU, C	A
UN-T15: "Legitimate actors are able to take actions that would unwittingly facilitate a cyber-attack"	T, I, D	ECU	C, I, A
UN-T16: "Manipulation of the connectivity of vehicle functions"	S, T, D	ECU, External interface	C, I, A
UN-T17: "Hosted 3rd party software, e.g., entertainment applications, used as a means to attack vehicle systems"	S, T, I, D	ECU	C, I, A
UN-T18: "Devices connected to external interfaces, e.g., USB ports, OBD port, used as a means to attack vehicle systems"	S, T, E	External interface	C, I
UN-T19: "Extraction of vehicle data/code"	I	ECU	C
UN-T20: "Manipulation of vehicle data/code"	T, R, I	ECU	C, I
UN-T21: "Erasure of data/code"	T, R	ECU	C, I
UN-T22: "Introduction of malware"	T	ECU	C, I
UN-T23: "Introduction of new software or overwrite existing software"	T, E	ECU	C, I
UN-T24: "Disruption of systems or operations"	D	ECU	A
UN-T25: "Manipulation of vehicle parameters"	T	ECU	A, I
UN-T26: "Cryptographic technologies can be compromised or are insufficiently applied"	S, D, I, E	ECU, C	C, I, A
UN-T27: "Parts or supplies could be compromised to permit vehicles to be attacked"	T	ECU	
UN-T28: "Software or hardware development permits vulnerabilities"	T, E	ECU	C, I
UN-T29: "Network design introduces vulnerabilities"	S, I, D, T	C	C,I,A
UN-T30: "Physical loss of data"	T, D	ECU	C, I, A
UN-T31: "Unintended transfer of data can occur"	I	ECU, C	C, I
UN-T32: "Physical manipulation of systems can enable an attack"	T	ECU	C, I

4.7. Feasibility

GAPP uses a structured approach to assess the feasibility of attack paths in automotive networks as described in our previous work in [6]. This method involves assigning feasibility factors to each threat, collecting threats along the attack path, comparing and selecting the maximum factor for each category, and calculating the overall feasibility. The *Feasibility* associated with a *Threat* mentioned in Section 4.3 is defined as a 5-tuple of the feasibility factors defined in ISO/SAE 21434 [5]:

$$Feasibility = (elapsed\ time, expertise, knowledge, opportunity, equipment) \tag{18}$$

To evaluate the overall feasibility of multi-step attack paths, we define the affmax function, which aggregates individual attack feasibility factors across a sequence of exploits. This function enables the identification of the most critical constraints in a composite attack sequence. Each of these factors is evaluated on a numerical scale according to Annex G.2 in [5].

Let $R = r_1, \ldots, r_k$ denote the list of all *Feasibility* tuples along an attack path consisting of an arbitrary count $k \in \mathbb{N}$ steps. Using the notation $r_i = (v_{i,1}, ..., v_{i,5})$ to represent the individual attack feasibility factors within the feasibility tuples, we can define the affmax function as

$$\text{aff}_{\max}[r_1, ..., r_k] := (max[v_{1,1}, ..., v_{k,1}], ..., max[v_{1,5}, ..., v_{k,5}]) \quad (19)$$

The affmax function operates on attack feasibility factor tuples, calculating component-wise maximums that reflect the minimum capabilities an attacker must possess across all dimensions [53].

Our selection of component-wise maximum aggregation, rather than alternative approaches like weighted averages or probabilistic methods, is motivated by the practical realities of automotive security. In vehicle systems, attacks typically require specific threshold capabilities in specialized domains; a single capability gap can render an entire attack sequence infeasible [50,52]. While future research might explore more complex aggregation techniques, including Bayesian probability models or context-sensitive weighting schemes, our understanding of the automotive security regulations [5,8] indicates that the component-wise maximum introduces a good balance between analytical precision and computational efficiency.

In Neo4j, we incorporated this affmax function to select the maximum value for each attack feasibility factor across all threats in a given attack path.

5. Results

5.1. Sample System

To demonstrate the GAPP methodology in practice, we utilized a reference example derived from the ISO/SAE 21434 standard Annex H.2 [5] as shown in Figure 5. This allowed us to ground our approach in a standardized use case while facilitating comparison with established security assessment practices. The example defines the item boundaries, and the operational environment as shown in Figure 5. The system comprises an On-Board Diagnostics (OBD) interface connected via a CAN bus to a gateway (GW), which is further connected via a second CAN bus to a Headlight Unit (HL). The system contains two external communication interfaces, a Bluetooth interface and a cellular interface, and both are connected to a navigation ECU, further connected to the gateway. The Headlight Unit represents the asset, containing data, communication functionality, and ECU firmware with security properties encompassing confidentiality, integrity, and availability. The operating context significantly influences the relevance and priority of threats in automotive security analysis. We can accommodate these contextual variations through adjustable impact ratings for different deployment scenarios.

5.1.1. Damage Scenarios

We identify three primary damage scenarios for the Headlight Unit:

- Loss of Headlight Functionality: Vehicle cannot be safely operated during night conditions due to compromised headlamp function.
- Unauthorized Control: Malicious deactivation of headlamps during nighttime driving at moderate speed, potentially resulting in a collision with stationary objects.

- Data Breach: Exfiltration of sensitive intellectual property related to Headlight Unit design and implementation. Modern automotive lighting systems contain substantial proprietary technology including adaptive lighting algorithms, matrix LED control systems, and communication protocols for integration with vehicle dynamics and perception systems [13,52]. These systems often incorporate proprietary calibration data, lighting patterns, and control algorithms that represent significant research investments.

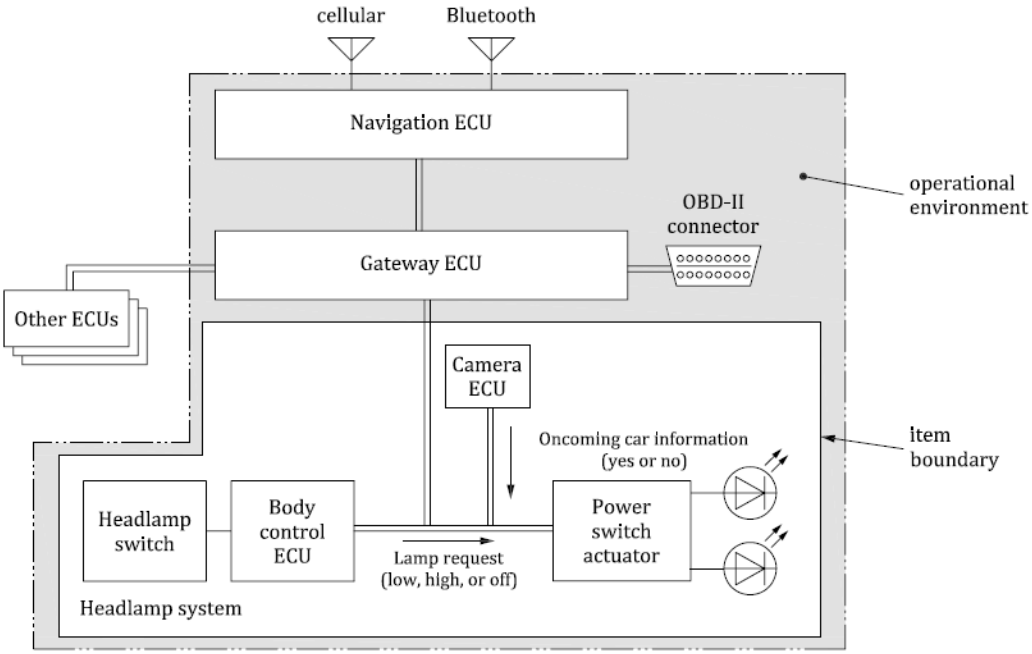

Figure 5. Example headlamp system from the ISO 21434 [5].

5.1.2. Threat Scenarios

From the UN R155 threat catalog [8], we selected a subset of applicable threats that could potentially realize the identified damage scenarios. Table 3 presents the threats according to the formal model described in Section 4.3.

The feasibility values assigned to each threat are derived through expert analysis following the methodology detailed in Section 5.1.8. This approach to threat modeling supports the creation of reusable automotive vulnerability databases that can be leveraged across different projects by formalizing threat characteristics according to standardized taxonomies and metrics.

Table 3. Threat model for the sample system.

Threat ID	STRIDE	Type	Feasibility	Privilege	Security Property
UN-T5	T	C	{4, 4, 5, 4, 4}	r/w	C, I
UN-T6	S	C	{4, 3, 5, 4, 4}	r/w	C, I
UN-T16	S, T, D	ECU	{2, 3, 3, 1, 4}	ex, r, w, fc	C, I, A
UN-T18	S, T, E	External	{1, 3, 3, 1, 4}	r/w, ex, r, w, fc	C, I
UN-T19	I	ECU	{4, 6, 5, 4, 4}	r	C
UN-T24	D	ECU	{4, 6, 5, 4, 7}	ex	A
UN-T28	T, E	ECU	{4, 6, 5, 4, 7}	ex, r, w, fc	C, I

5.1.3. System Model

The system architecture can be formally represented according to our definition framework:

- $\mathcal{N}_{\text{OBD}} = (\text{OBD}, \text{ExternalInterface}, \text{internal}, \{\emptyset, \text{CAN1}\}, \text{OBD})$;
- $\mathcal{N}_{\text{CAN1}} = (\text{CAN1}, \text{Communication}, \text{internal}, \{\text{OBD}, \text{GW}\}, \text{CAN})$;
- $\mathcal{N}_{\text{GW}} = (\text{GW}, \text{ECU}, \text{internal}, \{\text{CAN1}, \text{CAN2}\}, \{\text{data}, \text{function}\})$;
- $\mathcal{N}_{\text{CAN2}} = (\text{CAN2}, \text{Communication}, \text{internal}, \{\text{GW}, \text{HL}\}, \text{CAN})$;
- $\mathcal{N}_{\text{HL}} = (\text{HL}, \text{ECU}, \text{asset}, \{\text{CAN2}, \emptyset\}, \{\text{data}, \text{function}\})$;
- $\mathcal{N}_{\text{Cellular}} = (\text{Cellular}, \text{ExternalInterface}, \text{internal}, \{\emptyset, \text{NAV}\}, \text{Cellular})$;
- $\mathcal{N}_{\text{BLE}} = (\text{BLE}, \text{ExternalInterface}, \text{internal}, \{\emptyset, \text{NAV}\}, \text{BLE})$;
- $\mathcal{N}_{\text{NAV}} = (\text{NAV}, \text{ECU}, \text{internal}, \{\text{Cellular}, \text{BLE}, \text{CAN0}\}, \{\text{data}, \text{function}\})$;
- $\mathcal{N}_{\text{CAN0}} = (\text{CAN0}, \text{Communication}, \text{internal}, \{\text{NAV}, \text{GW}\}, \text{CAN})$;
- $\mathcal{A}_{\text{HL}} = (\text{HL}, (\{\text{safety}, \text{operational}\}, \{\text{severe}\}), \{\text{C}, \text{I}, \text{A}\})$.

5.1.4. State Model and Transition Function

We will use the following abbreviated notation for privilege levels: read/write (rw), read (r), write (w), execute (ex), and full control (fc). To manage computational complexity, we exclude self-transitions, resulting in three distinct transition types:

1. External interface to communication transitions:
   - rw $\rightarrow$ rw
2. ECU internal state transitions:
   - ex $\rightarrow$ r, ex $\rightarrow$ w, ex $\rightarrow$ fc
   - r $\rightarrow$ ex, r $\rightarrow$ w, r $\rightarrow$ fc
   - w $\rightarrow$ ex, w $\rightarrow$ r, w $\rightarrow$ fc
3. Communication to ECU transitions:
   - rw $\rightarrow$ ex, rw $\rightarrow$ r, rw $\rightarrow$ w, rw $\rightarrow$ fc
4. ECU to communication transition
   - ex $\rightarrow$ rw, r $\rightarrow$ rw, w $\rightarrow$ rw, fc $\rightarrow$ rw

5.1.5. Event Space Definition

The event space Σ_e for our model is defined as follows:

$$\Sigma_e = \{\text{Threat}_1, \text{Threat}_2, \text{Threat}_3, \text{Threat}_4, \text{Threat}_5, \text{Threat}_6, \text{Threat}_7\} \tag{20}$$

where:

$$\text{Threat}_1 = (\text{UN-T5}, \text{T}, \text{C}, \{4, 4, 5, 4, 4\}, \text{r/w}, \{\text{C}, \text{I}\}) \tag{21}$$

$$\text{Threat}_2 = (\text{UN-T6}, \text{S}, \text{C}, \{4, 3, 5, 4, 4\}, \text{r/w}, \{\text{C}, \text{I}\}) \tag{22}$$

$$\text{Threat}_3 = (\text{UN-T16}, \{\text{S}, \text{T}, \text{D}\}, \text{ECU}, \{2, 3, 3, 1, 4\}, \{\text{ex}, \text{r}, \text{w}, \text{fc}\}, \{\text{C}, \text{I}, \text{A}\}) \tag{23}$$

$$\text{Threat}_4 = (\text{UN-T18}, \{\text{S}, \text{T}, \text{E}\}, \text{External}, \{1, 3, 3, 1, 4\}, \{\text{ex}, \text{r}, \text{w}, \text{fc}\}, \{\text{C}, \text{I}\}) \tag{24}$$

$$\text{Threat}_5 = (\text{UN-T19}, \text{I}, \text{ECU}, \{4, 6, 5, 4, 4\}, \text{r}, \text{C}) \tag{25}$$

$$\text{Threat}_6 = (\text{UN-T24}, \text{D}, \text{ECU}, \{4, 6, 5, 4, 7\}, \text{ex}, \text{A}) \tag{26}$$

$$\text{Threat}_7 = (\text{UN-T28}, \{\text{T}, \text{E}\}, \text{ECU}, \{4, 6, 5, 4, 7\}, \{\text{ex}, \text{r}, \text{w}, \text{fc}\}, \{\text{C}, \text{I}\}) \tag{27}$$

5.1.6. Security Model Instantiation

Following the threat model mapping to the state transition system, we instantiate the security model with

$$\text{state} = (\text{node}, \text{privilege}) \tag{28}$$

$$S_{\text{start}} = \{\text{OBD}, \text{rw}\} \tag{29}$$

$$V = (\text{SP}, \text{feasibility}, \text{impact}) \tag{30}$$

The transition function T is defined as

$$T = \{t_0 = \{\text{start point}, \text{OBD}_{\text{rw}}, \text{UN-T18}\}, \tag{31}$$

$$t_1 = \{\text{OBD}_{\text{rw}}, \text{CAN1}_{\text{rw}}, \text{UN-T6}\}, \tag{32}$$

$$t_2 = \{\text{CAN1}_{\text{rw}}, \text{GW}_{\text{ex}}, \text{UN-T28}\}, \tag{33}$$

$$t_3 = \{\text{GW}_{\text{ex}}, \text{CAN2}_{\text{rw}}, \text{UN-T5}\}, \tag{34}$$

$$t_4 = \{\text{CAN2}_{\text{rw}}, \text{HL}_{\text{r}}, \text{UN-T19}\}, \tag{35}$$

$$t_5 = \{\text{CAN2}_{\text{rw}}, \text{HL}_{\text{fc}}, \text{UN-T24}\}, \tag{36}$$

$$t_6 = \{\text{start point}, \text{Cellular}_{\text{rw}}, \text{UN-T16}\}, \tag{37}$$

$$t_7 = \{\text{Cellular}_{\text{rw}}, \text{NAV}_{\text{ex}}, \text{UN-T16}\}, \tag{38}$$

$$t_8 = \{\text{start point}, \text{BLE}_{\text{rw}}, \text{UN-T16}\}, \tag{39}$$

$$t_9 = \{\text{BLE}_{\text{rw}}, \text{NAV}_{\text{ex}}, \text{UN-T16}\}, \tag{40}$$

$$t_{10} = \{\text{NAV}_{\text{ex}}, \text{CAN0}_{\text{rw}}, \text{UN-T5}\}, \tag{41}$$

$$t_{11} = \{\text{CAN0}_{\text{rw}}, \text{GW}_{\text{w}}, \text{UN-T25}\}, \tag{42}$$

$$t_{12} = \{\text{GW}_{\text{w}}, \text{CAN2}_{\text{rw}}, \text{UN-T6}\}, ...\} \tag{43}$$

5.1.7. Attack Path Analysis

To illustrate the methodology, we analyze a specific attack path: UN-T18 → UN-T6 → UN-T28 → UN-T5 → UN-T19.

This path represents a multi-stage attack sequence:

1. Initial Access (UN-T18): An attacker employs diagnostic tools (e.g., OBD dongles) to establish initial system access, enabling both direct and indirect vehicle parameter manipulation.
2. Message Manipulation (UN-T6): The attacker intercepts and modifies communications between the OBD port and internal systems, potentially hijacking legitimate sessions.
3. Privilege Escalation (UN-T28): Software vulnerabilities are exploited to elevate privileges within the Gateway ECU, enabling message manipulation and forwarding capabilities.
4. Channel Exploitation (UN-T5): Communication channels are leveraged to manipulate vehicle data/code, potentially through parameter alteration or command injection.
5. Extraction of vehicle data (UN-T19): Proprietary software or sensitive information is extracted from vehicle systems, compromising confidentiality and potentially exposing intellectual property.

5.1.8. Threat Feasibility Assessment

The feasibility assessment for each threat vector follows the attack potential-based approach specified in ISO/SAE 21434, with evaluations across five key dimensions: elapsed time (ET), specialist expertise (SE), knowledge of item/component (KoIC), window of opportunity (WoO), and equipment (EQ). To illustrate this assessment approach, we provide detailed analysis for three threats in our model:

UN-T5: Communication channels used to conduct unauthorized actions

- Elapsed Time: $\leq$1 month (Value: 4)
  Vehicle function manipulation requires a comprehensive analysis of control message structures and timing constraints within the CAN protocol implementation. This requires time for reverse engineering proprietary message formats and timing sequences.
- Specialist Expertise: Expert (Value: 4)
  Successful exploitation requires Expert-level understanding of vehicle functional architecture and cross-domain security mechanisms.
- Knowledge of Target: Restricted (Value: 5)
  Implementation-specific knowledge of function activation mechanisms and associated authentication protocols is needed.
- Window of Opportunity: Moderate (Value: 4)
  Moderate accessibility, dependent on vehicle operational state.
- Equipment: Specialist (Value: 4)
  Specialized diagnostic equipment and protocol analyzers are required.

UN-T18: External interface compromise

- Elapsed Time: $\leq$1 week (Value: 1)
  Due to readily accessible interfaces with documented vulnerabilities and established exploitation techniques in the public domain.
- Specialist Expertise: Proficient (Value: 3)
  Knowledge of automotive network protocols and interface specifications is needed.
- Knowledge of Target: Restricted (Value: 3)
  Restricted but partially accessible through public documentation.
- Window of Opportunity: Unlimited (Value: 1)
  External interfaces are typically continuously accessible and designed for connectivity.
- Equipment: Standard (Value: 4)
  Commercial interface adapters and open-source software tools are typically sufficient for exploitation.

UN-T19: Extraction of vehicle data

- Elapsed Time: $\leq$1 month (Value: 4)
  Extracting proprietary code or sensitive data requires significant time investment for identifying storage locations and developing extraction methodologies.
- Specialist Expertise: Expert (Value: 6)
  Specialist knowledge of automotive firmware structures and memory protection is needed.
- Knowledge of Target: Restricted (Value: 5)
  Internal data organization and storage mechanisms are restricted information.
- Window of Opportunity: Moderate (Value: 4)
  Moderate, limited by operational states and security mechanisms designed to prevent unauthorized data access.
- Equipment: Specialist (Value: 4)
  Specialized diagnostic and memory analysis tools are required.

The feasibility of the attack path is calculated in Table 4 using the affmax function, which identifies the maximum value for each feasibility factor across all threats in the path. The resulting feasibility matrix $[4, 6, 5, 4, 7]$ yields an aggregated feasibility value of 26. According to the ISO/SAE 21434 standard's normalization rules, this value translates to a feasibility rating of "Very Low" (0), as values ≥ 25 indicate attacks requiring extensive expertise and resources. When combined with the "Severe" impact rating (2) associated with compromising the Headlight Unit's confidentiality, the risk is calculated using the standard formula:

$$R = I \times f = 2 \times 0 = 0 \tag{44}$$

This risk value indicates that although the potential impact is severe, the extremely low feasibility of successfully executing this complex attack path results in an effectively negligible risk level.

Table 4. Feasibility analysis of attack path components.

Threat	ET	SE	KoIC	WoO	EQ
UN-T18	1	3	3	1	4
UN-T6	4	3	5	4	4
UN-T28	4	6	5	4	7
UN-T5	4	4	5	4	4
UN-T19	4	6	5	4	4
affmax	4	6	5	4	7

5.2. Implementation

Having established the theoretical foundations of the GAPP methodology, with a practical illustration, we now present its implementation using the Neo4j graph database platform. This section demonstrates how the formal models and algorithms previously defined are translated into a practical cybersecurity analysis tool. The implementation leverages Neo4j's native graph processing capabilities to efficiently represent complex automotive systems and generate attack paths in accordance with the formal models.

Graph databases are a class of database management systems that utilize a graph data model to store and manage data effectively [54]. These databases provide a simpler yet more expressive model compared to traditional relational databases and other NoSQL stores. This simplicity and expressiveness enable a more efficient representation of complex relationships and data structures, offering a powerful approach to handling highly interconnected data.

5.2.1. Neo4j

Neo4j is a leading graph database, known for its flexibility in modeling complex real-world data structures [55]. It employs a property graph model, which consists of three fundamental components: nodes, relationships, and properties.

- Nodes: These represent entities or objects within the data model. Nodes can be assigned labels, which categorize them and facilitate filtering during algorithm execution or querying. Notably, a node can possess multiple labels, allowing for nuanced categorization.
- Relationships: These denote connections or interactions between nodes. Each relationship has a type and direction, specifying the subject and object of the interaction, thereby providing context to the connections.
- Properties: These are key–value pairs that provide additional information about both nodes and relationships, enhancing the richness of the data model.

Neo4j utilizes Cypher as its query language, a declarative language specifically designed for querying graph data. This makes it well suited for complex data analysis tasks [56].

In the realm of cybersecurity, Neo4j is increasingly utilized for various applications, including network visualization, data flow analysis, and attack path identification [57]. It

also plays a role in cyber threat intelligence, constructing knowledge graphs for cybersecurity [58], and analyzing correlations between cybersecurity and safety [59]. The ability of Neo4j to represent complex relationships and integrate data from multiple sources makes it particularly suitable for our model.

5.2.2. System Model

The system model is implemented using the Neo4j graph database as shown in the pseudocode in Listing 1, where each element is represented as either a node or a relationship, effectively translating formal definitions into a concrete model.

- Nodes are uniquely identified by an ID. They are categorized using labels to specify their roles, such as "ECU", "Communication", and "External interface". Each node contains properties corresponding to the formal model, such as type, subtype, and role.
- Edges are derived from a "connectivity list" as defined in the formal model.

The results for the system model implementation for our sample system in Neo4j is presented in Figure 6.

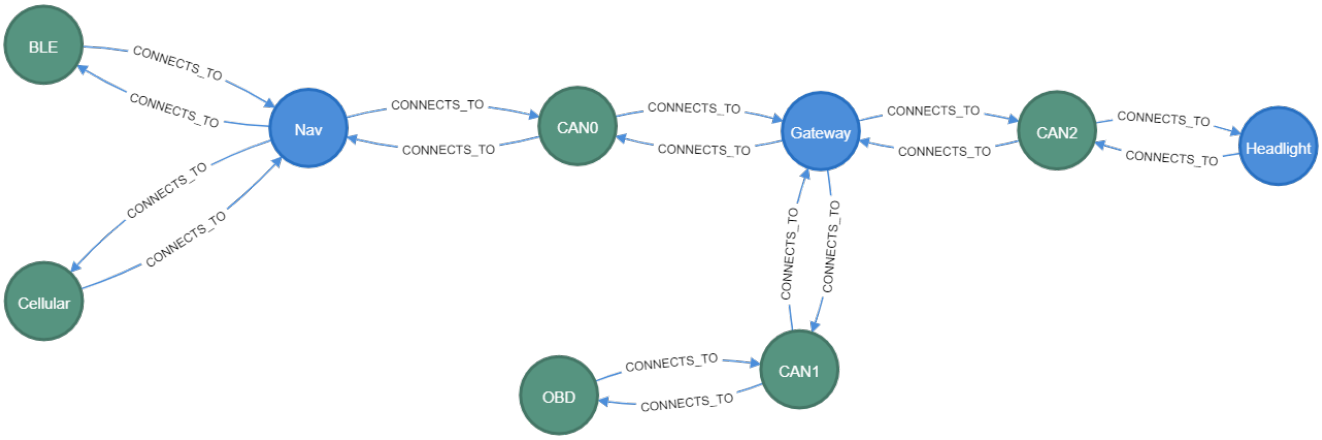

Figure 6. Implementation of the GAPP system model in Neo4j graph database.

5.2.3. State Model Implementation

The state space generation implements a deterministic algorithm that traverses the system model and generates corresponding state representations according to the formal privilege model. For each component $n \in N$, the algorithm instantiates state nodes $s \in S$ based on component type:

- For components where $n.Type = ExternalInterface$ or where $n.Type = Communication$, a single state node is generated with $privilege = readwrite$.
- For components where $n.Type = ECU$, multiple state nodes are generated corresponding to the hierarchical privilege levels defined in the attacker privilege model: $\{execute, read, write, fullcontrol\}$.

Figure 7 illustrates the resulting state model implementation for the reference automotive system described in Section 5.1. The figure illustrates the Neo4j implementation where different node colors represent distinct component types: navy blue nodes represent ECU components, green nodes indicate communication networks and external interfaces, and turquoise nodes represent the privilege states associated with each component. Due to the complexity of the graph structure and space constraints in the visualization, some state node labels appear truncated with ellipsis notation ("..."), representing longer identifiers abbreviated for display purposes. Each ECU node (navy blue) branches into multiple

privilege-level states (turquoise) such as execute, read, write, and full control as defined in the formal model, creating the hierarchical structure visible in the implementation.

Listing 1. System model creation algorithm.

```
function CreateSystemModel(TOE_Description):
    FOR EACH Component in TOE_Description:
        // Determine node label based on Component.Type
        IF Component.Type == "ECU" OR
           Component.Type == "Communication" OR
           Component.Type == "ExternalInterface":

            // Create node with appropriate type label
            CREATE (:{Component.Type} {
                name: Component.Name,
                type: Component.Type,
                subtype: Component.Subtype,
                conn: Component.Conn,
                role: Component.Role,
            })

            // Handle Asset nodes
            IF Component.Subtype == "asset":
                MATCH (c) WHERE c.name = Component.Name
                SET c.sub_type = "Asset",
                    c.security_property = Component.SecurityProperty,
                    c.impact = Component.Impact,
                    c.Impact_rating = Component.ImpactRating
// Create connections between components
FOR EACH Conn IN TOE_Description.ConnectivityList:
    MATCH (src), (tgt)
    WHERE src.name = Conn.Source AND tgt.name = Conn.Target
    CREATE (src)-[:CONNECTS_TO]->(tgt)
```

5.2.4. Relationships Creation

The implementation establishes four primary relationship categories within the graph topology:

- HAS_STATE Relationship: This edge type establishes a mapping between system components and their corresponding state representations. Each component node ($n \in N$) is associated with its potential state configurations ($s \in S$) through directed edges that facilitate hierarchical traversal between the architectural and behavioral model layers.
- AFFECTS Relationship: These directed edges implement the vulnerability mapping between threats ($e \in \Sigma_e$) and the system states they can potentially compromise.
- LEADS_TO Relationship: These edges constitute the graph-theoretical implementation of the state transition function (T). Each relationship instance represents a potential state transition $t \in T : s_1 \rightarrow s_2$ that satisfies the guard conditions (g) defined in the formal model.
- CONNECTS_TO Relationship: These edges implement the connectivity constraints defined in the system model's reachability list. They represent physical or logical communication channels between architectural components.

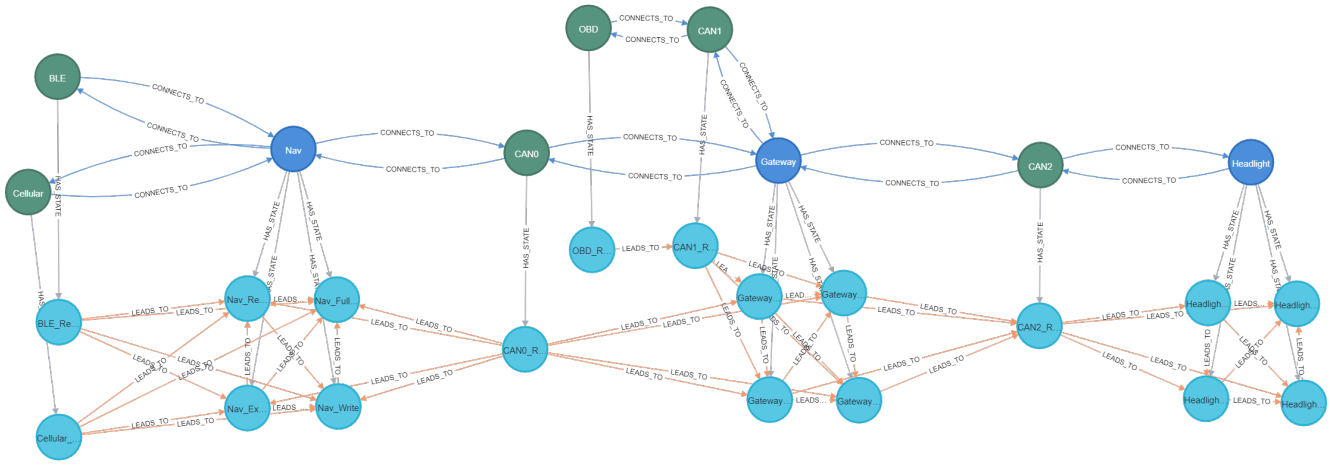

Figure 7. Implementation of the GAPP state model with privilege levels in Neo4j.

5.2.5. Vulnerability Association and Mapping Methodology

This mapping process demonstrated with the pseuocode in Listing 2 enforces the formal constraints defined in the threat model Section 4.3 through three primary classification:

- STRIDE Classification: The algorithm verifies that each vulnerability's STRIDE categorization aligns with the corresponding threat STRIDE categories for the target state's privilege level as defined in Table 1.
- Component Type Compatibility: The mapping enforces architectural compatibility by validating that each vulnerability's target component type (ECU, Communication, External Interface) matches the architectural classification of the corresponding state node's parent component.
- Privilege Level: Each vulnerability is associated with state nodes that match the privilege levels as defined in the attacker privilege model.

Figure 8 illustrates the security model integration, depicting the complete graph topology with system model components, derived state representations, and mapped vulnerabilities. This comprehensive view demonstrates the complete GAPP security model integration within Neo4j, where the color coding distinguishes between different node types, pink nodes represent the threats from the UN R155 catalog. The AFFECTS relationships are visible as connections between the pink vulnerability nodes and the turquoise state nodes, demonstrating how specific threats can compromise particular privilege levels within the system components.

Listing 2. Vulnerability mapping algorithm.

```
function MapVulnerabilities():
    // Map vulnerabilities to states with matching attributes
    MATCH (v:Vulnerability), (s:State), (c)-[:HAS_STATE]->(s)
    WHERE v.type CONTAINS c.type
      AND ANY(cat IN v.STRIDE WHERE cat IN s.stride)
      AND (v.priv == s.level OR v.priv_max >= s.level)
    CREATE (v)-[:AFFECTS]->(s)
```

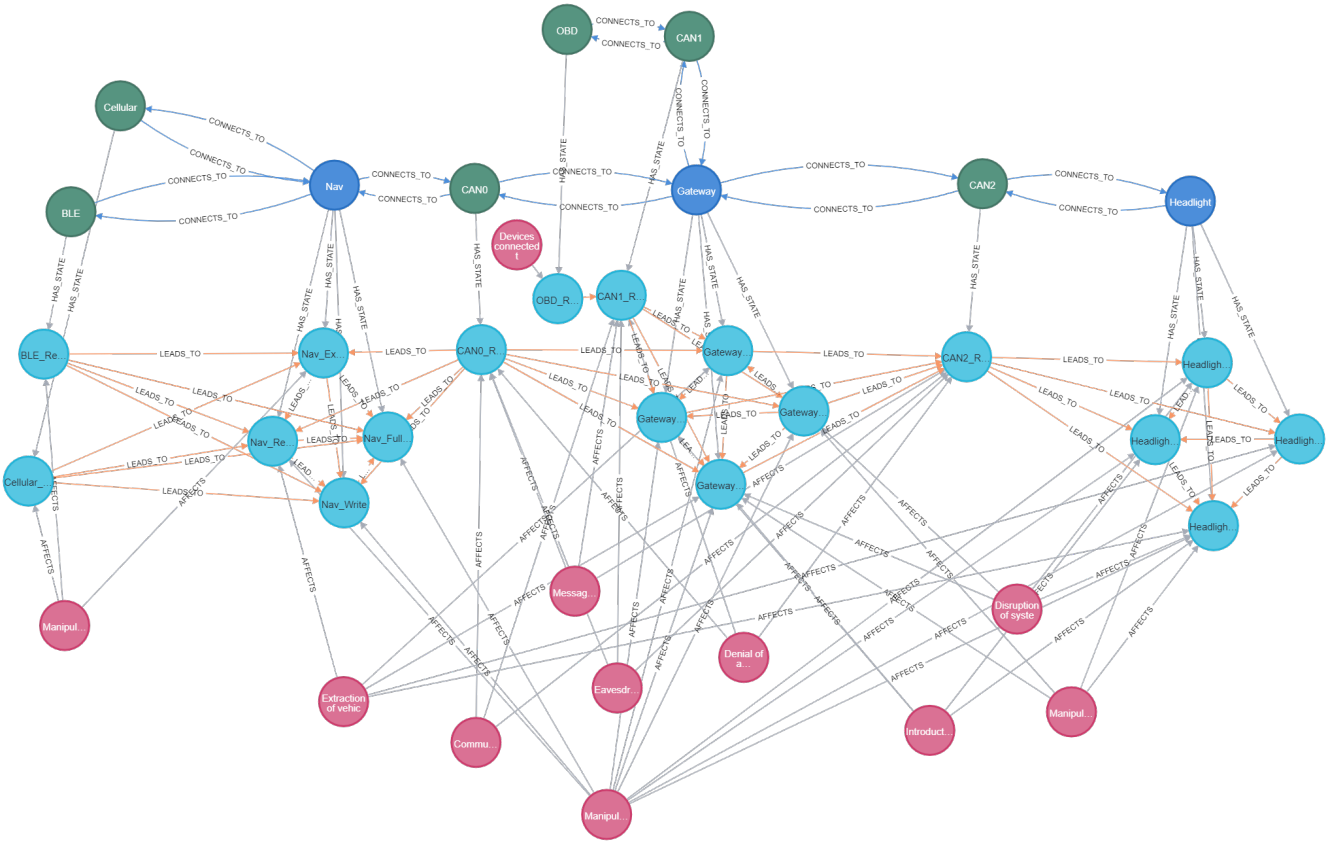

Figure 8. The full graph implementation of the example system in Section 5.1. Vulnerabilities are mapped to state nodes, which are mapped to architecture elements.

5.2.6. Attack Path Generation

This process utilizes Neo4j's query capabilities while maintaining the formal transition constraints through a multi-phase approach as shown in the algorithm in Listing 3:

- Topological Path Identification: Parameterized queries identify paths connecting external interface nodes to asset nodes while enforcing architectural connectivity and privilege monotonicity constraints:

$$\forall n_i, n_{i+1} \in Path : (n_i.Type = n_{i+1}.Type = \text{ECU}) \implies (Privilege(n_i) \leq Privilege(n_{i+1})) \tag{45}$$

- Threat Vector Integration: Each state transition is evaluated against applicable threat vectors according to the transition function (T), verifying that each transition $t \in T : s_1 \rightarrow s_2$ satisfies all guard conditions (g) specified in Section 4.4.2.
- Security Property Validation: Path construction terminates upon reaching an asset state with a threat compromising a defined security property:

$$\exists e \in \Sigma_e : (target(t) = s_{asset}) \wedge (e.SP \cap asset.SP \neq \emptyset) \tag{46}$$

- Risk Quantification: Each path undergoes risk assessment using the affmax aggregation function:

$$Risk(Path) = f(\text{affmax}[r_1, r_2, ..., r_k]) \times I_{asset} \tag{47}$$

The implementation, as shown in Listing 3 assumes monotonicity in attacker progression, precluding backtracking behaviors during exploitation sequences. The output comprises prioritized attack paths ordered by quantified risk scores, enabling focused mitigation efforts on critical exploitation vectors in accordance with ISO/SAE 21434 methodologies.

Listing 3. Attack path generation and validation algorithm.

```
function FindAttackPaths(source_type, target_type, algorithm_type)
    if algorithm_type in {"BFS", "DFS"} then
        paths ← GraphTraversal(
            start: State(type=source_type),
            end: State(sub_type=target_type),
            relation: "LEADS_TO",
            mode: algorithm_type,
            constraint: PrivilegeMonotonicity
        )
    else if algorithm_type = "Dijkstra" then
        paths ← WeightedSearch(
            start: State(type=source_type),
            end: State(sub_type=target_type),
            relation: "LEADS_TO",
            weight: "weight",
            constraint: PrivilegeMonotonicity
        )
    else if algorithm_type = "A*" then
        PrepareHeuristics(target_type)
        paths ← WeightedSearch(
            start: State(type=source_type),
            end: State(sub_type=target_type),
            relation: "LEADS_TO",
            weight: "weight",
            constraint: PrivilegeMonotonicity
        )
    for each path in paths do
        feasibility ← affmax(ExtractFeasibilityMatrices(path))
        impact ← GetImpactRating(path.targetNode)
        path.risk ← feasibility × impact

    return SortByMetrics(paths, ["feasibility", "length", "risk"])
```

5.2.7. Search Algorithm

The attack path generation in the GAPP framework is influenced by the underlying traversal algorithm selection. This section presents a comparative analysis of four distinct search strategies implemented and evaluated within our Neo4j-based attack graph model: BFS, DFS, Dijkstra's algorithm, and A* search as shown in Table 5.

Table 5. Search algorithm performance comparison.

Algorithm	Paths Found	Execution Time (ms)	Time per Path (ms)	Computational Complexity
DFS	32,500	78,172	2.4	$O(b^d)$
BFS	32,500	108,627	3.3	$O(b^d)$
Dijkstra	30	472	15.7	$O(E + V \log V)$
A*	54	403	7.5	$O(E)$

where b is branching factor, d is path depth, E is number of edges, and V is number of vertices.

In analyzing algorithms within graph theory, it is essential to consider their computational complexity, which measures the time and memory resources required as a function of the input size. For DFS and BFS, the complexity for traversing a graph is typically $O(|V| + |E|)$, where $|V|$ is the number of vertices and $|E|$ is the number of edges [21]. How-

ever, when these algorithms are used to enumerate all possible attack paths in a graph with cycles or multiple paths to target nodes, their practical complexity approaches $O(b^d)$, where b is the branching factor and d is the maximum path depth, which can explain the increased resources needed for BFS as it has higher branching than DFS. The heuristic-based A* algorithm and Dijkstra's algorithm offer advantages in finding optimal paths according to defined metrics. Dijkstra's algorithm, with complexity $O(E + V \log V)$, identifies shortest paths based purely on edge weights. A* enhances this approach by incorporating a heuristic function $h(n)$ that estimates the cost from the current node to the goal:

$$f(n) = g(n) + h(n) \tag{48}$$

where $g(n)$ is the known cost from the start node to the current node, and $h(n)$ is the estimated cost from the current node to the goal. In our implementation, the heuristic function is based on node privilege levels:

$$h(n) = |level(n) - level(target)| \tag{49}$$

This privilege-level based heuristic estimates the "distance" in terms of privilege escalation steps needed to reach target assets, providing domain-specific guidance to the search process. The effectiveness of A* algorithm highly depends on the heuristic function in guiding the search process.

- DFS (Depth-First Search): Identified 32500 unique attack paths with an execution time of 78.344 ms.
- BFS (Breath-First Search): Exhibited similar coverage with 32,500 attack paths but with a longer execution time at 108,342 ms.
- Dijkstra's: Identified 30 optimized attack paths with an execution time of 380 ms.
- A*: Identified 54 optimized attack paths with an execution time of 394 ms.

While exhaustive search algorithms (BFS/DFS) provide extensive attack surface visibility, the volume of identified paths exceeds practical manual review capacity. For practical usability, an automated sorting or filtering is required to help experts identify the highest-risk attack paths.

Conversely, informed search algorithms (Dijkstra/A*) produce highly selective results that may miss other important attack vectors. Here the opposite is needed, and all other paths with a similar risk level must also be identified.

6. Discussion

The goal of the TARA process is to identify the highest risk attack paths in a product. The GAPP methodology automates part of the process, reducing the burden on the experts during the analysis. We now evaluate the results obtained to confirm the validity of the results and also the effectiveness of the automation.

6.1. Path Analysis

Our implementation generated 32,500 unique attack paths using both BFS and DFS search algorithms. This large number of paths represents different combinations and sequences of threats that could potentially compromise system assets. Even in our relatively simple example system, the number of possible paths clearly demonstrates the complexity challenges faced by security analysts when performing manual TARA analysis. To make these results more manageable for expert review, we implemented a multi-criteria sorting approach that considers the risk value, the path length, and the feasibility value. Table 6 presents the top sorted results, showing paths with different combinations of threats that result in varying risk levels.

Table 6. High-risk attack paths with corresponding feasibility and risk values.

Paths	Path Length	Feasibility Matrix	Normalized Feasibility	Risk
[UN-T18, UN-T7, UN-T24, UN-T7, UN-T24]	4	[3, 4, 4, 4, 4]	1.5	3
[UN-T18, UN-T7, UN-T24, UN-T8, UN-T24]	4	[3, 4, 4, 4, 4]	1.5	3
[UN-T16, UN-T16, UN-T5, UN-T24, UN-T6, UN-T24]	5	[3, 4, 4, 4, 4]	1.5	3
[UN-T18, UN-T8, UN-T25, UN-T7, UN-T25]	4	[3, 5, 5, 4, 4]	1	2
[UN-T18, UN-T6, UN-T25, UN-T8, UN-T24]	4	[3, 5, 5, 4, 4]	1	2
[UN-T18, UN-T6, UN-T25, UN-T7, UN-T24]	4	[3, 5, 5, 4, 4]	1	2
[UN-T18, UN-T5, UN-T24, UN-T5, UN-T25]	4	[3, 5, 5, 4, 4]	1	2
[UN-T18, UN-T7, UN-T25, UN-T7, UN-T25]	4	[3, 5, 5, 4, 4]	1	2
[UN-T18, UN-T8, UN-T24, UN-T6, UN-T25]	4	[3, 5, 5, 4, 4]	1	2

The paths with the highest risk (Risk = 3) have the same length as many others (4 steps) but with a higher normalized feasibility (1.5), resulting in a higher overall risk, and were the top results across all the algorithms.

The first two paths correspond to the same state sequence:

$$\begin{aligned}\text{OBD}(\text{ReadWrite}) \rightarrow \text{CAN1}(\text{ReadWrite}) \rightarrow \text{Gateway}(\text{Execute}) \rightarrow \\ \text{CAN2}(\text{ReadWrite}) \rightarrow \text{Headlight}(\text{Execute})\end{aligned}$$

For the threat sequence [UN-T18, UN-T7, UN-T24, UN-T7, UN-T24], we can establish direct correspondence with the reference sequence from the ISO/SAE 21434 [5], as follows:

- Initial access via OBD interface (UN-T18).
- Network traffic eavesdropping to gather system information (UN-T7).
- Our path uses system disruption techniques (UN-T24) rather than parameter manipulation (UN-T25) shown in the ISO example, but both achieve the same privilege level on the Gateway ECU.
- Additional eavesdropping for confirmation/feedback (UN-T7).
- Our path concludes with system disruption (UN-T24) directly targeting the Headlight functionality, which achieves the same end result as the parameter manipulation in the ISO example.

The other threat sequence with the same risk value [UN-T18, UN-T7, UN-T24, UN-T8, UN-T24] uses denial of service instead of eavesdropping, but captures the same essential attack progression: external access → network exploitation → gateway compromise → target disruption.

6.2. Validation

To validate the GAPP methodology, we compared our automated results against the ISO/SAE 21434 Annex H [5] reference examples along various criteria.

- Architectural Consistency: The fundamental attack progression through system components to target assets is aligned with the standard's reference architecture.
- Attack Vector Representation: Critical attack vectors identified in the standard (diagnostic interface exploitation, message manipulation, privilege escalation) were represented in GAPP-generated paths.
- State Transition Validity: All generated paths exhibited state transitions consistent with the formal transition rules defined in Section 4.
- Recall Assessment: The paths from the ISO/SAE 21434 example were successfully identified by GAPP, yielding a recall of 100% for documented critical paths.

6.3. Discussion and Limitation

The most striking observation is the very large number (32,500) of potential attack paths found for even such a simple system. We will discuss the reasons for this large number, and also solutions on how to bring this down to a manageable level for the expert review.

First of all, multiple attack paths contain identical threats but with different sequencing. While these are considered distinct paths in our model, the need for expert review can be reduced by clustering them together in the future.

Second, the current implementation does not consider security controls and component-specific assumptions, which significantly impact the feasibility of executing certain attack sequences. For instance, extracting code from one ECU might be considered impossible due to implemented security controls, while the same action could be feasible on another ECU with different protection mechanisms. Similarly, breaking cryptographic implementations on specific ECUs may be computationally infeasible due to hardware security modules, while other implementations might be vulnerable.

Component-specific security controls and assumptions can be easily implemented in the model to better reflect real-world attack scenarios and reduce false positives. This will require extending the model to include detailed security control specifications for each component and their effectiveness against specific threat types. Our expectation is that this will eliminate a large number of attack paths.

Third, the comparative analysis of search algorithms reveals significant differences in their effectiveness for attack path identification. While exhaustive algorithms (BFS, DFS) generated 32,500 potential paths, Dijkstra's algorithm and A* identified only 30 and 54 paths, respectively, focusing specifically on the highest-risk attack vectors. The performance advantage of informed search algorithms stems from their ability to prioritize paths based on risk metrics, with their effectiveness heavily dependent on the quality of edge cost assignments and heuristic functions. Our privilege-level based heuristic for A* proved effective, enabling it to identify more high-risk paths than Dijkstra's algorithm while maintaining comparable execution speed. Both BFS and DFS algorithms effectively enumerate all possible attack paths, enabling subsequent analysis through multi-criteria sorting based on risk factors, feasibility metrics, and path length.

In addition, the affmax function used to combine the attack feasibility factors along a path could be replaced by a more sophisticated and fine-grained approach that takes into account the relative difficulty of finding longer attack paths, even if they are composed of simple steps. Note that a more fine-grained risk estimation must not be the only tool. Concentrating only on the highest risk may lead to ignoring other important attack vectors.

Our current approach treats each asset as a monolithic entity with uniform impact values across all security properties. This simplification, while useful for proof-of-concept implementation, does not reflect the nuanced reality of automotive systems. In practice, assets should be modeled with greater granularity, differentiating between data components and functional components, each with specific security property requirements and impact assessments. Future work should extend the asset model to include subtypes (e.g., firmware, configuration data, functional code) with individualized impact assessments across different categories (safety, financial, operational, privacy). This would allow for more precise risk calculations and better prioritization of attack paths based on the specific asset components they compromise.

Our current implementation does not consider the temporal evolution of the system state during attack progression. The temporal dependency modeling would require our graph database to evolve dynamically as attack paths are executed, reflecting how earlier exploits influence the feasibility and impact of subsequent attack steps. Future development

could enhance the model with dependency specifications between nodes, allowing for more realistic simulation of multi-stage attacks where the system state changes progressively.

Finally, we point out several deliberate simplifications made in the current implementation to manage computational complexity:

- Self-loops were eliminated from the state model to reduce path explosion.
- Backward transitions (lowering privilege levels) were excluded based on the assumption that attackers maintain or escalate privileges rather than relinquishing them.
- Monotonicity was assumed in privilege progression during attacks.

We consider that these limitations do not affect the core of the methodology. While selectively removing them may be feasible, this would not considerably improve the usefulness of the results.

7. Conclusions

In this study, we have presented the GAPP methodology as a systematic approach to automotive cybersecurity engineering. Our research addresses the fundamental limitations of traditional TARA processes, which have historically relied on manual analysis techniques with inherent constraints in scalability and comprehensive threat identification. Through our investigation, we have formalized an approach that integrates EFSM modeling with standardized threat taxonomies derived from the UNECE R155 regulatory framework. Our implementation employing Neo4j's property graph database architecture has demonstrated considerable efficacy in establishing precise mappings between architectural components, security states, and exploitation vectors. The formal model we developed facilitates systematic exploration of the attack surface while maintaining alignment with ISO/SAE 21434 workflows.

Automatic identification of the highest-risk paths allows the targeted introduction of new security controls. Further analysis of the produced data also allows for the identification of systemic weaknesses, where security improvements could be most effective.

Future research directions should focus on three primary areas: (1) extending the asset model to incorporate finer-grained security property representations that reflect the heterogeneous nature of automotive systems; (2) implementing component-specific security controls to reduce false positives and improve model fidelity; and (3) developing methodologies to identify weak points in a system and propose optimal security controls.

Author Contributions: Conceptualization, M.N.-E.S. and A.C.; methodology, M.N.-E.S.; software, M.N.-E.S.; writing—original draft preparation, M.N.-E.S.; writing—review and editing, A.C.; visualization, M.N.-E.S.; supervision, M.K.; funding acquisition, M.K. All authors have read and agreed to the published version of the manuscript.

Funding: The research was supported by the 2024-2.1.1 University Research Scholarship Program of the Ministry for Culture and Innovation from the source of the National Research, Development, and Innovation Fund. The authors would like to thank the 2019-1.3.1-KK-2019-00007 "Innovációs szolgáltató bázis létrehozása diagnosztikai, terápiás és kutatási célú kiberorvosi rendszerek fejlesztésére" national project for the support.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Acknowledgments: The authors would like to thank the AIAM (Applied Informatics and Applied Mathematics) doctoral school of Obuda University, Budapest, Hungary, for their support in this research. The research was supported by the 2024-2.1.1 University Research Scholarship Program of the Ministry for Culture and Innovation from the source of the National Research, Development, and Innovation Fund. We would like to thank the Robert Bosch Ltd./Bosch Group, Hungary, for their professional support of the research.

Conflicts of Interest: The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

A*	A-star (search algorithm)
affmax	Aggregation Function for Feasibility Maximization
AG	State-based Attack Graphs
AT	Attack Tree
BAG	Bayesian Attack Graph
BFS	Breadth-First Search
BLE	Bluetooth Low Energy
CAN	Controller Area Network
CSMS	Cybersecurity Management System
CVSS	Common Vulnerability Scoring System
DFS	Depth-First Search
E/E	Electrical and Electronic
ECU	Electronic Control Unit
EFSM	Extended Finite State Machine
EVITA	E-safety Vehicle Intrusion Protected Applications
fc	Full Control (privilege level)
GAPP	Graph-based Attack Path Prioritization
GW	Gateway
HL	Headlight
MPAG	Multiple Prerequisite Attack Graph
MOST	Media Oriented Systems Transport
NAV	Navigation
OBD	On-Board Diagnostics
r	Read (privilege level)
r/w	Read/Write (privilege level)
STRIDE	Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, Elevation of privilege
TARA	Threat Analysis and Risk Assessment
TOE	Target of Evaluation
UDS	Unified Diagnostic Services
UML	Unified Modeling Language
UNECE	United Nations Economic Commission for Europe
V2I	Vehicle-to-Infrastructure
V2V	Vehicle-to-Vehicle
V2X	Vehicle-to-Everything
w	Write (privilege level)

References

1. Dobaj, J.; Ekert, D.; Stolfa, J.; Stolfa, S.; Macher, G.; Messnarz, R. Cybersecurity Threat Analysis, Risk Assessment and Design Patterns for Automotive Networked Embedded Systems: A Case Study. *J. Univers. Comput. Sci. (JUCS)* **2021**, *27*, 830–849. [CrossRef]
2. Mundhenk, P. *Security for Automotive Electrical/Electronic (E/E) Architectures*; Cuvillier Verlag: Göttingen, Germany, 2017.
3. Boban, M.; Kousaridas, A.; Manolakis, K.; Eichinger, J.; Xu, W. Use cases, requirements, and design considerations for 5G V2X. *arXiv* **2017**, arXiv:1712.01754.
4. Lu, R.; Zhang, L.; Ni, J.; Fang, Y. 5G vehicle-to-everything services: Gearing up for security and privacy. *Proc. IEEE* **2019**, *108*, 373–389. [CrossRef]
5. *ISO/SAE 21434:2021*; Road Vehicles—Cybersecurity Engineering. ISO: Geneva, Switzerland, 2021. Available online: https://www.iso.org/standard/70918.html (accessed on 24 May 2025).

6. Saulaiman, M.N.E.; Csilling, A.; Kozlovszky, M. Integrated Automation for Threat Analysis and Risk Assessment in Automotive Cybersecurity Through Attack Graphs. *Acta Polytech. Hung.* **2025**, *22*, 149–168. [CrossRef]
7. *ISO 26262-1:2018*; Road Vehicles—Functional Safety. ISO: Geneva, Switzerland, 2021. Available online: https://www.iso.org/standard/68383.html (accessed on 24 May 2025).
8. UN Regulation No. 155—Cyber Security and Cyber Security Management System | UNECE. Available online: https://unece.org/transport/documents/2021/03/standards/un-regulation-no-155-cyber-security-and-cyber-security (accessed on 24 May 2025).
9. Common Vulnerability Scoring System Version 4.0 Calculator. 2025. Available online: https://www.first.org/cvss/calculator/4-0 (accessed on 24 May 2025).
10. *ISO 11898-1:2024*; Road Vehicles—Controller Area Network (CAN), Part 1: Data Link Layer and Physical Coding Sublayer. ISO: Geneva, Switzerland, 2021. Available online: https://www.iso.org/standard/86384.html (accessed on 24 May 2025).
11. *ISO 11898-2:2024*; Road Vehicles—Controller Area Network (CAN), Part 2: High-Speed Physical Medium Attachment (PMA) Sublayer. ISO: Geneva, Switzerland, 2021. Available online: https://www.iso.org/standard/85120.html (accessed on 24 May 2025).
12. Toghuj, W.; Turab, N. Automotive Ethernet architecture and security: Challenges and technologies. *Int. J. Power Electron. Drive Syst./Int. J. Electr. Comput. Eng.* **2023**, *13*, 5209. [CrossRef]
13. ENISA. *Cyber Security and Resilience of Smart Cars*; ENISA: Athens, Greece, 2016.
14. Koscher, K.; Czeskis, A.; Roesner, F.; Patel, S.; Kohno, T.; Checkoway, S.; McCoy, D.; Kantor, B.; Anderson, D.; Shacham, H.; et al. Experimental Security Analysis of a Modern Automobile. In Proceedings of the 2010 IEEE Symposium on Security and Privacy, Oakland, CA, USA, 16–19 May 2010; pp. 447–462. [CrossRef]
15. Chartrand, G.; Zhang, P. *A First Course in Graph Theory*; Courier Corporation: North Chelmsford, MA, USA, 2013.
16. Gross, J.L.; Yellen, J.; Anderson, M. *Graph Theory and Its Applications*; Chapman and Hall/CRC: Boca Raton, FL, USA, 2018; pp. 1–115.
17. Mondal, B.; De, K. An overview applications of graph theory in real field. *Int. J. Sci. Res. Comput. Sci. Eng. Inf. Technol.* **2017**, *2*, 751–759.
18. Dickson, A. *Introduction to Graph Theory*; CRC Press: Boca Raton, FL, USA, 2006.
19. Grinberg, D. An introduction to graph theory. *arXiv* **2024**, arXiv:2308.04512.
20. Wilson, R.J. *Introduction to Graph Theory*; Pearson Education India: Noida, India, 1979.
21. Jungnickel, D.; Jungnickel, D. *Graphs, Networks and Algorithms*; Springer: Berlin/Heidelberg, Germany, 2005; Volume 3.
22. Baier, C.; Katoen, J.P. *Principles of Model Checking*; The MIT Press Ser; The MIT Press: Cambridge, MA, USA; London, UK, 2008.
23. Rich, E. *Automata, Computability and Complexity: Theory and Applications*; Pearson Prentice Hall: Upper Saddle River, NJ, USA, 2008.
24. Salfer, M. *Automotive Security Analyzer for Exploitability Risks: An Automated and Attack Graph-Based Evaluation of On-Board Networks*; Springer Fachmedien: Wiesbaden, Germany, 2024. [CrossRef]
25. Hong, J.B.; Kim, D.S.; Chung, C.J.; Huang, D. A survey on the usability and practical applications of graphical security models. *Comput. Sci. Rev.* **2017**, *26*, 1–16. [CrossRef]
26. Kordy, B.; Pièvre-Cambacédès, L.; Schweitzer, P. DAG-based attack and defense modeling: Don't miss the forest for the attack trees. *Comput. Sci. Rev.* **2014**, *13*, 1–38. [CrossRef]
27. Sheyner, O.; Haines, J.; Jha, S.; Lippmann, R.; Wing, J.M. Automated generation and analysis of attack graphs. In Proceedings of the 2002 IEEE Symposium on Security and Privacy, Berkeley, CA, USA, 12–15 May 2002; pp. 273–284.
28. Schneier, B. Attack trees. *Dr. Dobb's J.* **1999**, *24*, 21–29.
29. Ingols, K.; Lippmann, R.; Piwowarski, K. Practical attack graph generation for network defense. In Proceedings of the 2006 22nd Annual Computer Security Applications Conference (ACSAC'06), Miami Beach, FL, USA, 11–15 December 2006; pp. 121–130.
30. Ou, X.; Boyer, W.F.; McQueen, M.A. A scalable approach to attack graph generation. In Proceedings of the 13th ACM Conference on Computer and Communications Security, Alexandria, VA, USA, 30 October–3 November 2006; pp. 336–345.
31. Saulaiman, M.N.E.; Kozlovszky, M.; Csilling, Á.; Banati, A.; Benhamida, A. Overview of Attack Graph Generation For Automotive Systems. In Proceedings of the 2022 IEEE 10th Jubilee International Conference on Computational Cybernetics and Cyber-Medical Systems (ICCC), Reykjavík, Iceland, 6–9 July 2022; pp. 000135–000142.
32. Tayouri, D.; Baum, N.; Shabtai, A.; Puzis, R. A Survey of MulVAL Extensions and Their Attack Scenarios Coverage. *IEEE Access* **2023**, *11*, 27974–27991. [CrossRef]
33. Salfer, M.; Eckert, C. Attack graph-based assessment of exploitability risks in automotive on-board networks. In Proceedings of the 13th International Conference on Availability, Reliability and Security, Hamburg, Germany, 27–30 August 2018; pp. 1–10.
34. Boudermine, A.; Khatoun, R.; Choyer, J.H. Dynamic logic-based attack graph for risk assessment in complex computer systems. *Comput. Netw.* **2023**, *228*, 109730. [CrossRef]
35. Barik, M.S.; Sengupta, A.; Mazumdar, C. Attack graph generation and analysis techniques. *Def. Sci. J.* **2016**, *66*, 559. [CrossRef]
36. Yekta, A.R.; Spychalski, D.; Yekta, E.; Yekta, C.; Katzenbeisser, S. VATT&EK: Formalization of Cyber Attacks on Intelligent Transport Systems—A TTP based approach for Automotive and Rail. In Proceedings of the 7th ACM Computer Science in Cars Symposium, Darmstadt, Germany, 5 December 2023; pp. 1–17.

37. Henniger, O.; Ruddle, A.; Seudié, H.; Weyl, B.; Wolf, M.; Wollinger, T. Securing vehicular on-board it systems: The evita project. In Proceedings of the VDI/VW Automotive Security Conference, Ingolstadt, Germany, 27–28 November 2009; Volume 41.
38. *ISO/IEC 15408-1:2022*; Information Security, Cybersecurity and Privacy Protection—Evaluation Criteria for IT Security. ISO: Geneva, Switzerland, 2022. Available online: https://www.iso.org/standard/72891.html (accessed on 24 May 2025).
39. Ruddle, A.; Ward, D.; Weyl, B.; Idrees, S.; Roudier, Y.; Friedewald, M.; Leimbach, T.; Fuchs, A.; Gürgens, S.; Henniger, O.; et al. Deliverable D2. 3: Security requirements for automotive on-board networks based on dark-side scenarios. *EVITA Proj.* **2009**. Available online: https://evita-project.org/deliverables.html (accessed on 14 January 2025).
40. Dürrwang, J.; Sommer, F.; Kriesten, R. Automation in automotive security by using attacker privileges. In Proceedings of the 19th Escar Europe—The World's Leading Automotive Cyber Security Conference, Escar Europe, Frankfurt, Germany, 10–11 November 2021; pp. 137–152. [CrossRef]
41. Abouelnaga, M.; Jakobs, C. Security Risk Analysis Methodologies for Automotive Systems. *arXiv* **2023**, arXiv:2307.02261.
42. Sommer, F. Automatic Attack Path Generation in Automotive Model-Based Security Testing. Ph.D. Thesis, Institute of Distributed Systems Faculty of Engineering, Computer Science and Psychology, Ulm University, Ulm, Germany, 2024.
43. Alberts, C.J.; Behrens, S.G.; Pethia, R.D.; Wilson, W.R. *Operationally Critical Threat, Asset, and Vulnerability Evaluation (OCTAVE) Framework, Version 1.0*; Technical Report; Carnegie Mellon University: Pittsburgh, PA, USA, 1999.
44. Karahasanovic, A.; Kleberger, P.; Almgren, M. Adapting threat modeling methods for the automotive industry. In Proceedings of the 15th ESCAR Conference, Berlin, Germany, 7–8 November 2017; pp. 1–10.
45. Saulaiman, M.; Csilling, A.; Kozlovszky, M. Leveraging Attack Graphs in Automotive Threat Analysis and Risk Assessment. In Proceedings of the SECURWARE 2023: The Seventeenth International Conference on Emerging Security Information, Systems and Technologies, Porto, Portugal, 25–29 September 2023.
46. Alagar, V.; Periyasamy, K. *Specification of Software Systems*; Texts in Computer Science; Springer: London, UK, 2011. [CrossRef]
47. Sommer, F.; Kriesten, R.; Kargl, F. Survey of Model-Based Security Testing Approaches in the Automotive Domain. *IEEE Access* **2023**, *11*, 55474–55514. [CrossRef]
48. Luo, F.; Jiang, Y.; Wang, J.; Li, Z.; Zhang, X. A framework for cybersecurity requirements management in the automotive domain. *Sensors* **2023**, *23*, 4979. [CrossRef] [PubMed]
49. National Vulnerability Database (NVD). 2025. Available online: https://nvd.nist.gov/ (accessed on 14 January 2025).
50. Checkoway, S.; McCoy, D.; Kantor, B.; Anderson, D.; Shacham, H.; Savage, S.; Koscher, K.; Czeskis, A.; Roesner, F.; Kohno, T. Comprehensive experimental analyses of automotive attack surfaces. In Proceedings of the 20th USENIX Security Symposium (USENIX Security 11), San Francisco, CA, USA, 8–12 August 2011.
51. Dürrwang, J. Steigerung der Betriebssicherheit von Personenkraftwagen Durch Bedrohungsanalysen für die Informationssicherheit. Ph.D. Thesis, Technische Universität München, Münich, Germany, 2022.
52. Miller, C.; Valasek, C. A Survey of Remote Automotive Attack Surfaces. In Proceedings of the Black Hat USA, Las Vegas, NV, USA, 2–7 August 2014; pp. 1–94.
53. Angermeier, D.; Wester, H.; Beilke, K.; Hansch, G.; Eichler, J. Security risk assessments: Modeling and risk level propagation. *ACM Trans. Cyber-Phys. Syst.* **2023**, *7*, 1–25. [CrossRef]
54. Robinson, I.; Webber, J.; Eifrem, E. *Graph Databases: New Opportunities for Connected Data*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2015.
55. Anuyah, S.; Bolade, V.; Agbaakin, O. Understanding graph databases: A comprehensive tutorial and survey. *arXiv* **2024**, arXiv:2411.09999.
56. Vukotic, A.; Watt, N.; Abedrabbo, T.; Fox, D.; Partner, J. *Neo4j in Action*; Manning Publications Co.: Shelter Island, NY, USA, 2014.
57. Pelofske, E.; Liebrock, L.M.; Urias, V. Cybersecurity threat hunting and vulnerability analysis using a Neo4j graph database of open source intelligence. *arXiv* **2023**, arXiv:2301.12013.
58. Noel, S.; Harley, E.; Tam, K.H.; Gyor, G. Big-data architecture for cyber attack graphs: Representing security relationships in NoSQL graph databases. In Proceedings of the IEEE Symposium on Technologies for Homeland Security (HST), Waltham, MA, USA, 14–16 April 2015; pp. 1–6.
59. Wang, Y.; Ren, Y.; Cui, Z.; Yu, H. Proactive security defense: Cyber threat intelligence modeling for connected autonomous vehicles. *arXiv* **2024**, arXiv:2410.16016.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.