# AVSDA: Autonomous Vehicle Security Decay Assessment

Lama Moukahal[1], Mohammad Zulkernine[1], and Martin Soukup[2]

[1] Queen's University, Kingston, Canada
{lama.moukahal, mz}@queensu.ca
[2] Irdeto, Ottawa, Canada
martin.soukup@irdeto.com

**Abstract.** Security practices become weaker over time as attackers' capabilities evolve. Security decay within vehicle software systems can have devastating consequences as it can pose a direct threat to people's lives. Thus, it is crucial to monitor the changing threat level on vehicles during their full lifespan. We present an Autonomous Vehicle Security Decay Assessment (AVSDA) framework that analyzes and predicts the system's security risk over vehicles' lifespan. The framework analyzes vulnerable software components periodically and estimates the security risk level to identify security decay. AVSDA employs several metrics specifically designed for autonomous vehicle systems to automatically identify potentially weak components and quantify security risk. We evaluate the framework on OpenPilot, an autonomous driving system. The case study demonstrates the effectiveness of the AVSDA framework in identifying security decay over time. The results show an accuracy rate of 94% and a recall rate of 78%, outperforming all other known metrics by at least 50%.

**Keywords:** Security Vulnerability · Autonomous Vehicle Systems Security · Decay Assessment · Risk Analysis.

## 1 Introduction

Attackers' capabilities evolve with time. What was once secure can become an easy target for skilled attackers to take advantage of systems' weaknesses to initiate attacks. Hence, any software system should be carefully monitored to identify possible security decay that can expose it to malicious behavior. Software integration and internet connectivity expose vehicles to cybersecurity challenges that, if not handled, can lead to destructive results. It is essential to identify security decay in automobile systems.

Automotive manufacturers are striving to diminish the chances of attacks. Currently, many developed automotive standards provide software guidelines to enhance vehicle security [1,10,11,17]. Following security standards during Vehicle Software Engineering (VSE) helps create more resilient Connected Autonomous Vehicles (CAVs) that defend against current attacks [29]. Nevertheless, attackers'

techniques are advancing. The average age of cars and trucks is more than ten years, and future vehicles are expected to operate for even a longer period. With such a long lifespan, new software vulnerabilities will be discovered, new attacker tools will be developed, and adopted security practices will become weaker. Ensuring CAV security requires planning that does not bind security assurance and risk assessment to the development phase but spans to cover the vehicles' operation phase.

Security decay represents a drop in system resilience due to newly discovered vulnerabilities, more skilled attackers, or changes in the operating environment of vehicle software. This research aims to identify security decay across vehicle software systems' full lifespan. We achieve this by identifying vulnerabilities in the system and assessing the evolving risks. We propose an Autonomous Vehicle Security Decay Assessment (AVSDA) framework composed of two phases. The first phase, vulnerability analysis, automatically and efficiently identifies potentially weak or vulnerable components. The second phase, risk analysis, focuses on quantifying the risk of weak components by determining an attack's likelihood and assessing its impact.

Traditional threat and risk assessment methods (e.g., E-Safety Vehicle Intrusion Protected Applications (EVITA) threat and risk model [31]) determine security risks by identifying and classifying potential threats. In contrast, we identify security risks by targeting the source of issues. The AVSDA framework distinguishes the vulnerable components that are responsible for the vast majority of vehicle cyberattacks. Considering the operational environment of vehicles, quantifying various threat scenarios becomes a daunting task. Hence, vulnerability analysis is used to efficiently measure the weak components that make vehicle software systems unprotected against attacks (e.g., unauthorized access to data, acceptance of bogus information, and unauthorized control of vehicles).

Assessing autonomous system security decay at the software level can help prevent malicious behavior and maintain vehicle safety. The AVSDA framework offers security engineers the opportunity to strengthen vehicles' resilience against attacks. It also warns security specialists about severe security decay that might require immediate update or even vehicle recalls to prevent incidents. This framework is critical for the United Nations Economic Commission for Europe (UN-ECE) WP.29 cybersecurity compliance [9].

The rest of the paper is organized as follows: Section 2 reviews related work. Section 3 outlines the AVSDA framework. Section 4 presents results from applying the AVSDA framework. Finally, Section 5 concludes the paper.

## 2 Related Work

Evaluating security decay in software systems is a recent topic in the literature and standards. As we assess security decay based on risk analysis, we review the existing security risk estimation efforts.

SAE J3601 [17] recommends assessing security threats in the automotive industry to identify possible threats. However, it does not identify a specific Threat

Analysis and Risk Assessment (TARA) method that can best identify the automotive industry's security risks [23]. The EVITA threat and risk model [31] is considered one of the potent risk assessment models in the automotive industry. The model focuses on identifying all possible attacks against a specific target. However, the sets of attacks and targets within autonomous vehicle systems are practically large, making risk assessment a time-consuming and challenging job. ISO/SAE 21434 [4] proposes a generic risk assessment process that involves vulnerability analysis, which is estimated based on previously identified vulnerabilities. However, historical data is not sufficient to identify the evolving vulnerabilities of vehicles.

Burton et al. [15] stress the importance of identifying intentional third party hazards to enhance vehicle safety. The researchers suggest enhancing safety standards to include the categorization of malicious hazards that can affect safety. Similarly, Macher et al. [24] highlight the need for threat and risk assessment techniques for the automotive domain. The researchers propose an approach to classify cybersecurity threats and merge it with ISO 26262 safety HARA framework. However, it is not enough to address vehicle security from a safety perspective only. Security risks have several impacts other than safety, including operational and financial impacts.

Islam et al. [21] introduce a risk assessment framework that aims to identify security requirements for automotive systems. Though the researchers propose a solid framework, their approach operates based on the system's data flows that can be difficult to obtain in the automotive industry. Othmane et al. [14] propose including attacker's capabilities in threat likelihood estimation and follow a manual vulnerability identification approach. However, considering the size and complexity of automotive systems, manual validation may not always be feasible.

This paper offers a security decay assessment framework that quantitatively evaluates the system without any additional overhead. The framework is not bound to the development phase; it assesses security during the operation phase too.

## 3 Framework Design

This section introduces the Autonomous Vehicle Security Decay Assessment (AVSDA) framework and discusses its phases. We begin by providing an overview of AVSDA and then dive deeper into the framework phases.

### 3.1 Overview

The AVSDA framework aims to identify security decay of autonomous vehicle software systems. The proposed framework analyzes the security decay at the Software Component (SWC) level. We define SWC as *a structural element that provides an interface. It can utilize different automotive communication means and is connected to other parts to fulfill a function.* This includes all types of

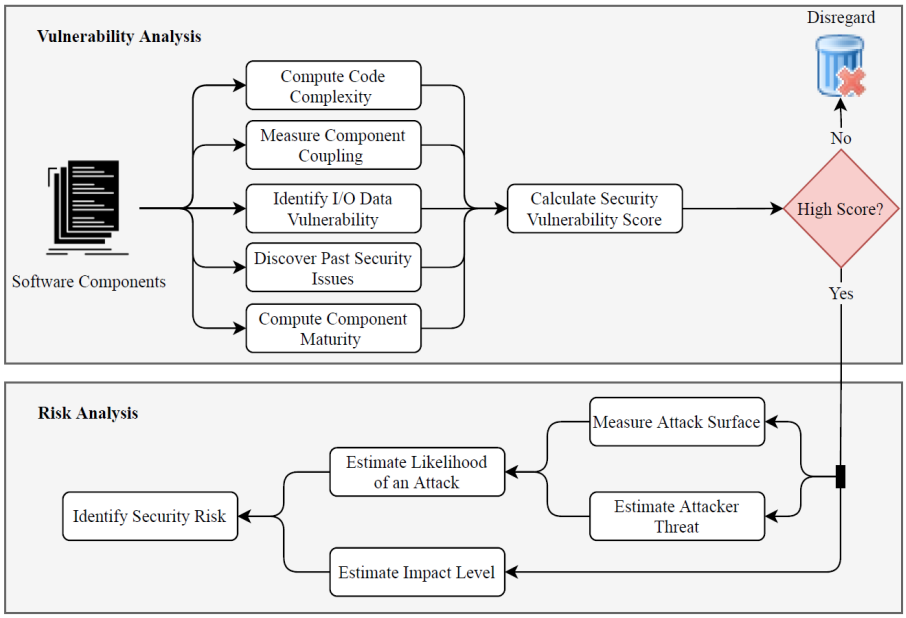


Fig. 1: Autonomous Vehicle Security Decay Assessment (AVSDA) framework.

SWCs defined by Automotive Open System Architecture (AUTOSAR) [1], covering all kinds of embedded hardware and firmware in a vehicle system.

Unauthorized access, acceptance of falsified information, interruption of service, and unauthorized control are all types of threats that an attacker can initiate to jeopardize vehicle software systems. Such threats violate vehicles' security and can threaten individuals' safety. The AVSDA framework utilizes a vulnerability analysis approach to estimate the vehicle software system's weak components that facilitate the existence of security threats.

As depicted in Fig. 1, AVSDA comprises two phases: vulnerability analysis and risk analysis. The first phase identifies potentially vulnerable components that can cause security failures. Attackers take advantage of existing software defects to initiate malicious behavior. Hence, to detect security decay, we first quantitatively identify the weak components based on the security metrics designed to target autonomous systems' vulnerabilities. The second phase thoroughly analyzes the system's vulnerable components to determine an attack's likelihood and impact. It also identifies security risk level for vulnerable components. Inspired by our previously proposed metrics [28], we define new and enhanced security metrics for this framework.

Estimating security risks before a product release is essential to prevent catastrophic results. AVSDA should be applied before moving a vehicle model to production and periodically during the operation phase to help security engineers measure vulnerabilities, estimate changing risk levels, and avoid unwanted security breaches. Comparing the results of subsequent runs can help security

engineers in identifying system security decay. An increase in the security risk level can alert security engineers to apply the proper mitigation measures.

### 3.2 Vulnerability Analysis Phase

Vulnerabilities in autonomous systems are common. Hence, it is essential to identify weaknesses before they are exposed and lead to successful attacks. The first phase of AVSDA, vulnerability analysis, measures the security vulnerability score of every component. Components with a high security vulnerability score are further analyzed in the risk analysis phase. Measuring security vulnerability involves six steps: (1) Compute code complexity, (2) Measure component coupling, (3) Identify input and output data vulnerability, (4) Discover past security issues, (5) Compute component maturity, and (6) Calculate security vulnerability score. Steps 1 through 5 can run simultaneously, and the results of these steps are utilized in the final assessment (Step 6).

In this phase, we consider the unique architecture of vehicles and the specific development challenges of vehicle software systems. Since vehicle software systems have vulnerability factors similar to other software systems, some of the steps of this phase can be applied to other systems (Steps 1, 4, and 5). Steps 2 and 3 are specific to autonomous vehicles. Step 2 measures how reliant a component is on other subsystems by defining the set of reachable Electronic Control Units (ECUs) from a component ECU. Step 3 identifies input and output data risks. We consider the different communication means that transmit inputs and outputs within vehicle software systems and the various threat levels that each poses on a vehicle.

**Compute Code Complexity.** Autonomous systems are by far one of the largest pieces of software in terms of size. A modern vehicle features around 100 million code lines, and this number is expected to grow to 300 million shortly as we move toward code-driven vehicles [27]. Integrating millions of code lines in vehicles enabled them to become more aware of their environment. However, the code complexity of autonomous vehicle systems can increase the number of defects. Many researchers associated code complexity with the existence of vulnerabilities [16,18,33]. Complex code is challenging to understand, test, validate, and maintain. Attackers look for defects in the system that can be exploited. Hence, complex code increases attackers' chances and is a good indicator of a high number of vulnerabilities. Researchers propose different attributes to calculate code complexity, including Source Line of Code (SLOC), Nesting Count, Nesting Depth (ND), McCabe's Cyclomatic, and Number of Children (NOC).

Durisic et al. [18] in collaboration with Volvo Car Corporation show that code complexity and coupling can efficiently be used in the automotive industry. In general, developers consider Nesting Count, Nesting Depth (ND), and lack of structure to be the attributes that most reflect complexity in a system [13]. Moreover, ND and Number of Children (NOC) correlate to vulnerabilities the most [16,33]. Accordingly, we define Code Complexity (CX) as a combination of

Source Line of Code (SLOC), ND, and NOC. The CX of component $C$ can be calculated using Equation (1). We use different weights $\omega_1$, $\omega_2$, and $\omega_3$ to give security experts a chance to assign different importance to different attributes. The weight values should be defined at the beginning of the assessment to apply them consistently to all components.

$$CX(C) = \omega_1 SLOC + \omega_2 ND + \omega_3 NOC \tag{1}$$

**Measure Component Coupling.** Coupling between objects is the concept that two or more entities rely on each other to fulfill functionality. In the automotive industry, code coupling can support engineers in finding complex components that might require more attention and testing than other parts in the system [18]. Code coupling is not only used to recognize complexity but is also extensively used to identify vulnerabilities [16, 26]. The dependability of entities in a system can help a malicious message propagate from one component to another, making an attack impact more severe.

In autonomous vehicle systems, coupling can be at two levels: components and functions. We covered function coupling in the code complexity. Component Coupling (CC) aims to measure how reliant a component is on other subsystems. Communication between autonomous systems components is needed to offer customers various functionalities. For example, the safety system in modern vehicles can communicate with the central locking system to lock the doors when the vehicle reaches a certain speed and unlock them when it stops. Such communication between components is essential to ensure the safety of passengers. However, components coupling permits the propagation of malicious messages [36].

We define CC as the set of ECUs reachable from a component's ECU, calculated using transitive closure. The transitive closure determines direct and indirect coupling. For example, consider component $A$ which runs on $ECU_A$ and connects through the gateway ECU to components $B$ and $C$ through $ECU_B$ and $ECU_C$, respectively. Component $A$ does not rely on component $D$, so no communication between $ECU_A$ and $ECU_D$ occurs. However, component $B$ communicates with component $D$. Consequently, a malicious message can propagate indirectly from $ECU_A$ to $ECU_D$ through $ECU_B$. The CC of component $C$ is calculated using Equation (2), where $R$ is the set of relationships between the ECUs of Component $C$.

$$CC(C) = \bigcup_{i=1}^{\infty} R^i \tag{2}$$

**Identify Input and Output Data Vulnerability.** Components of autonomous systems operate based on the collected data from sensors, radars, cameras, vehicles, infrastructure, users' mobile devices, and other sources. According to the inputs received and the embedded functionality, a component will transmit signals that control the vehicle's behavior. Inputs and outputs (I/O) offer an exceptional opportunity for attackers. Vehicles' diverse operating conditions make data validation a challenging job. Vehicles are always moving and sensing their surrounding environment. Hence, it is impossible to quantify all possible I/O.

Inputs and outputs are transmitted through different communication means. For example, autonomous systems depend on data received by the Global Positioning System (GPS) receiver to identify a vehicle position and navigate drivers to their destinations. Vehicle to Vehicle (V2V) communication is used to distribute traffic information. GPS and V2V channels each pose different risks on CAVs . GPS is vulnerable to jamming and spoofing [30], while V2V communication exposes the vehicle to external attacks like eavesdropping, spoofing, Denial of Service (DoS), and spamming [35]. While both of these communication means put the vehicle at risk, the level of threat between one communication means and the other is different. V2V communication exposes the autonomous system to a broader range of attacks [35].

The Input and Output Data Vulnerability (DV) observes two elements: the type and the mean of communication used. Fixed I/O data types (e.g., data types with constant values) are easier to validate and considered less risky compared to fluctuating I/O data types (e.g., an integer value that has an extensive range) that are challenging to validate. Moreover, different communication technologies are subject to various security issues. Thus, each communication mean is assigned a weight according to its criticality. The DV of component $C$ can be calculated using Equation (3). $K$ represents the total number of communication means, $FI$ and $FO$ represent fixed inputs and outputs, respectively, $LI$ and $LO$ represent fluctuating inputs and outputs respectively, $\omega_k$ is the weight of a specific communication mean, and $\alpha$ and $\beta$ are weights of fluctuating I/O.

$$DV(C) = \sum_{k=1}^{K} \omega_k |FI(C)| + \alpha\omega_k |LI(C)| + \omega_k |FO(C)| + \beta w_k |LO(C)| \quad (3)$$

**Discover Past Security Issues.** There have been many successful attacks against CAVs [7]. News of a security breach often gets the attention of malicious users who take advantage of an exposed vulnerability to conduct similar events. Hence, any bug, vulnerability, or attack on the vehicle software system must be carefully examined to prevent future malicious actions. Past Security Issue (PSI) gives higher importance to components that were subject to attacks.

PSI examines the frequency and age of an incident. A security incident that occurs regularly indicates a weakness in the system. Thus, attacks that happen many times are given higher importance. Attacks that arose from a long time ago and did not recur are more likely resolved. Hence, PSI introduces the forgetting factor to give more importance to recently discovered vulnerabilities. Equation (4) illustrates how the PSI of component $C$ can be calculated. $Y$ represents the total number of years since the first vehicle attack, $\alpha_y$ represents the number of attacks that occurred in year $y$, and $\lambda$ is the forgetting factor.

$$PSI(C) = \sum_{y=1}^{Y} \alpha_y \lambda^{Y-y} \mid 0 \leq \lambda \leq 1 \quad (4)$$

**Compute Component Maturity.** Component Maturity (CM) is essential for identifying vulnerabilities and security decay during vehicle operation. A component can witness many changes due to requirements changes, enhancements, security updates, and bug fixes. Researchers observe that continuous updates

and code changes can weaken code robustness and make it more prone to vulnerabilities [19, 33]. Code Churn (CCH) calculates the modifications made to a component over time and quantifies the changes' extent. We evaluate CCH by identifying the ratio of changes in a component, including deleted, added, and modified SLOC, as presented in Equation (5).

As we are interested in evaluating a component's security decay, it is vital to exclude the changes that are meant to enhance the security of an element while calculating CCH. Reviewing the security practices developed within a component can enhance the security measures and improve the component's defense mechanism. We consider components that witness security improvements as more resilient against cyberattacks. We calculate the Security and Maintenance Intensity (SMI) by counting the security enhancement activities since a product release. The reverse percentage is used to determine a low risk for proper security-maintained components, as shown in Equation (6). Therefore, CM covers extensively changed and low security-maintained code. The CM of component $C$ can be calculated following Equation (7).

$$CCH(C) = \frac{|Changed\ SLOC|}{|SLOC|} \tag{5}$$

$$SMI(C) = 1 - \frac{|Security\ Maintenance\ Activity|}{Age} \tag{6}$$

$$CM(C) = CCH(C) + SMI(C) \tag{7}$$

**Calculate Security Vulnerability Score.** The final assessment of a component's Security Vulnerability (SV) is calculated based on the values obtained from the previous five steps. As presented in Equation (8), to have proportional values, the results obtained from each step for a component $C$ are divided by the maximum ($MAX$) value that can be acquired by the corresponding step covering all components of the system. maDifferent weights can be assigned to each step.

$$\begin{aligned} SV(C) = \alpha\left(\frac{CX(C)}{MAX(CX)}\right) + \beta\left(\frac{CC(C)}{MAX(CC)}\right) + \gamma\left(\frac{PSI(C)}{MAX(PSI)}\right) \\ +\delta\left(\frac{DV(C)}{MAX(DV)}\right) + \theta\left(\frac{CM(C)}{MAX(CM)}\right) \end{aligned} \tag{8}$$

### 3.3 Risk Analysis Phase

The vulnerability analysis phase and risk analysis complement each other in identifying system security decay. The second phase of the decay model examines the potentially weak entities closely and quantifies the system's overall risk level. The risk analysis phase involves five steps: (1) Measure attack surface, (2) Estimate attacker threat, (3) Estimate likelihood of an attack, (4) Estimate impact level, (5) and Identify security risk.

The risk analysis phase is tailored to accommodate the uniqueness of vehicle software systems. For example, Step 1 of the second phase starts by measuring vehicle software systems' attack surfaces. We consider all communication means used by vehicle software systems. Moreover, we describe the attacker threat and impact level parameters specifically for vehicle software systems.

**Measure Attack Surface.** The attack surface of a component is the subset of system resources that an attacker can use to initiate malicious behavior [32]. To conduct an attack, malicious users connect to one of the vehicle's networks and invoke certain functions to send or/and receive information. For example, in 2015, a simulated attack was initiated on Jeep Cherokee while operating on the highway. Using the telematics system's Wi-Fi connection, the attackers transmitted messages to disable the brakes and halt the engine functions [2]. Hence, an attacker usually connects to one of the system's channels, invokes some methods, and sends data items to establish an attack on the system.

The attack surface examines the sets of entry points, exit points, communication channels, and untrusted data of a system [25]. The entry point set holds the means through which data can enter into the autonomous software system from the vehicle's environment (e.g., user inputs, sensor inputs, and incoming signals). The exit point set carries the means that enable data to exit from the system (e.g., outgoing signals). There exist various communication channels that an attacker can use to connect to a vehicle. A remote attack in the vehicle software system may occur through long-distance communication mechanisms such as cellular and satellite radio. Access to the vehicle's on-board diagnostics (OBD) port permits physical attacks that enable attackers to connect to the internal vehicle network [34]. In between are close-range wireless communications such as Near Field Communication (NFC) and Bluetooth, which can be utilized to perform remote attacks with nearby relays and proxies. Finally, the untrusted data set contains persistent data items stored on the nonvolatile memory of ECUs to send or receive data indirectly.

The DV calculates the risk of I/O considering their type and the used mean of communication. Nevertheless, not all I/O can be used in an attack. The attack surface metric includes only the resources contributing to an attack. Hence, in this step, some manual validation is required. We closely look at the DV metric result to identify which elements can ease attacks.

The attack surface (AS) of each component is assigned one of the three levels[3]: large (value of 8), medium (value of 3), and small (value of 1). Large

[3] We define specific level values to rate the risk. These values are identified to reflect the level of risk and enable quantitative measurement. Different risk values have comparable ranges to reflect various risk levels accurately. Consistently, the highest risk level between different parameters has a value of 8, and the lowest has 1. In between these two levels, values are assigned depending on the number of medium levels (e.g., one medium level assigned value 3, two medium levels assigned values 4 and 2). Security engineers can assign other values but have to follow the same approach assuring proportional ranges in the risk values of different levels.

Table 1: Attacker threat parameters.[3]

| Parameter | Definition | Level | Value |
|---|---|---|---|
| Skill | Technical experience an agent should possess. | Non-specialists: No experience is required to conduct an attack. | 8 |
| | | Skilled: Some experience is expected in the fundamentals of technology to initiate a successful attack. | 3 |
| | | Specialist: Profound knowledge in attacking techniques is required to break the system. | 1 |
| Knowledge | Background knowledge an agent should acquire about the vehicle software system architecture. | Public: Information needed to attack a system is publicly available (e.g., standards and protocols) | 8 |
| | | Restricted: Data required to initiate an attack is shared with partners and protected by non-disclosure agreement. | 4 |
| | | Private: Sensitive data shared internally with specific members is needed to conduct an attack. | 2 |
| | | Critical: Information required to conduct an attack is strictly shared with few members (e.g., cryptography keys). | 1 |
| Equipment | Software tools and hardware tools needed to attack a vehicle. | Standard: Tools needed are cheap and broadly available (e.g., RTL-SDR). | 8 |
| | | Sophisticated: Obtaining the equipment is not easy and expensive. | 3 |
| | | Rare: Equipment required is not available and may entail designing or producing a sophisticated tool. | 1 |
| Opportunity | The time and attack type (remote, physical) needed to break the system. | Large: Attacking the system takes a short time and can be conducted remotely. | 8 |
| | | Medium: Attacking the system needs some time, and either physical or remote access is required. | 3 |
| | | Small: Attacking the system requires much time with physical and remote access to achieve the attack. | 1 |

AS indicates that the component's attack surface sets expose the system to multiple attacks. Medium AS means that the attack surface of the component indicates the possibility of some attacks. Small AS suggests a very low probability of initiating an attack on this component. The AS of component $C$ is then estimated using Equation (9), where $\omega_a$ is a weight assigned by security experts to emphasize the importance of the attack surface, and $L_a$ is the value assigned based on the level.

$$AS(C) = \omega_a L_a \tag{9}$$

**Estimate Attacker Threat.** We take a closer look at the agent that initiates a threat. This step of the framework is vitally important to identify security decay. As discussed earlier, attackers' experience and knowledge are always evolving, which affect the security of the system and make it weaker. To estimate the attacker threat (AT), we employ four parameters: Skill, Knowledge, Equipment, and Opportunity. Table 1 describes the parameters and assigns values[3] based on the defined levels. Similar parameters are utilized in the literature with slightly different definitions [3,12,21]. The AT of component $C$ is estimated using Equation (10), where $P$ is the set of parameters, $\omega_p$ is the weight, and $T_p$ is the value of the parameter.

$$AT(C) = \sum_{p=1}^{P} \omega_p T_p \tag{10}$$

**Estimate Likelihood of an Attack.** An attack takes place by an agent that targets vehicle system vulnerabilities. Hence, to estimate the likelihood of an attack (LA) Equation (11) is used, multiplying the attack surface probability with the attacker threat probability.

$$LA(C) = AS(C) \times AT(C) \tag{11}$$

**Estimate Impact Level.** Attacks' impact may vary significantly; some may cause minor issues that do not necessitate a rapid response, while others can have devastating outcomes that require prompt resolution. We estimate the impact level (IL) with five parameters: Safety, Operational, Financial, Privacy, and Reputational. Vehicle attacks can affect different parties in the vehicle industry, including passengers, drivers, pedestrians, vehicle manufacturers, and associated companies. The five defined parameters estimate the impact considering all these parties. For example, the safety parameter evaluates the direct physical damage caused by an attack on the vehicle users, while the reputational parameter considers the indirect harm to manufacturers. The parameters are presented in detail in Table 2 with values[3] based on the impact level. We define the safety impact levels based on ISO 26262 [11], a well-established safety standard for vehicles. Similar parameters are utilized in the literature, but we tailor the parameters' level to fit the vehicle industry [4, 20, 22, 31]. To estimate the IL of an attack on Component $C$, Equation (12) is used. $F$ is the set of attack impact parameters, $\omega_f$ is the weight of the parameter assigned by security specialists, and $I_f$ is the value of this parameter.

$$IL(C) = \sum_{f=1}^{F} \omega_f I_f \tag{12}$$

**Identify Security Risk.** The final security risk (SR) of a component is obtained using Equation (13), which links the likelihood of an attack with the impact level. The security measures applied to protect the vehicle can lessen the threat. Thus, when estimating the SR, security specialists have to analyze and review the security controls adopted within a component to assess their ability to protect the attack surface and diminish attackers' capabilities. According to the analysis, a vehicle component's SR is classified into three levels: low, moderate, and severe. Low level means that the component is not under risk. This could be due to the security measures applied or because an attack probability is very low. Moderate level indicates that an attack risk exists. However, countermeasures can be used to lessen the risk. Severe level indicates that the component is facing very high risk, and the result of an attack may be critical. Applying security measures at this level might not be sufficient.

$$SR(C) = LA(C) \times IL(C) \tag{13}$$

## 4 Case Study

This section demonstrates the use of the AVSDA framework. We utilize OpenPilot (Version 0.7.9) [5], an open-source driver assistance system in our case study [6]. OpenPilot is an Autopilot system that can perform various functionalities, including Adaptive Cruise Control, Automated Lane Centering, Forward

Table 2: Impact level parameters.[3]

| Parameter | Definition | Level | Value |
|---|---|---|---|
| Safety | Safety of vehicle passengers, pedestrians, and road users. | High: Life-threatening injuries with the possibility of casualties. | 8 |
| | | Medium: Critical injuries with the possibility of survivals. | 3 |
| | | Low: Moderate injuries with the assurance of survivals. | 1 |
| | | None: No injuries. | 0 |
| Operational | Interruption of vehicular services. | High: Loss of significant subsystem in the vehicle that causes poor driving conditions. | 8 |
| | | Medium: Some functionalities within the vehicle system may not be operating correctly without affecting passengers' safety and driving conditions. | 3 |
| | | Low: Minor operations are interrupted that do not affect the vehicle performance (e.g., audio services, calling services). | 1 |
| | | None: No interruption | 0 |
| Financial | Direct and indirect financial losses affecting the vehicle owner and manufacturer. | High: Enormous financial damages that leave the vehicle manufacturer with bankruptcy risk. | 8 |
| | | Medium: Significant financial losses that slightly affect the financial situation of the manufacturer. | 3 |
| | | Low: Minor financial losses that do not affect the manufacturer operation. | 1 |
| | | None: No losses. | 0 |
| Privacy | Damages caused by data misusage, including users and manufacturer information. | High: Data leakage and privacy violations affecting a high number of users. | 8 |
| | | Medium: Data leakage and privacy violations affecting a small number of users. | 3 |
| | | Low: Minor privacy violation without any data leakages. | 1 |
| | | None: No data misusage. | 0 |
| Reputational | Damages that affect the reputation of the manufacturer organization. | High: Loss of a large number of customers and shareholders with the inability to recover and restore a good reputation. | 8 |
| | | Medium: Loss of some customers. | 3 |
| | | Low: Some unsatisfied customers that can be compensated. | 1 |
| | | None: No damages. | 0 |

Collision Warning, and Lane Departure Warning (LDW). Hence, the Autopilot system offers SAE level three [8] driving features that can be integrated with different car models such as Honda and Toyota. OpenPilot has one component only, Autopilot. We apply the AVSDA phases on the Autopilot component, illustrating the usefulness of this framework. Such an examination can verify the metrics' effectiveness by comparing the files' vulnerability scores with the number of discovered vulnerabilities in every file. We finalize this section by demonstrating the importance of applying AVSDA periodically.

### 4.1 Vulnerability Analysis

We show that the vulnerability analysis phase of AVSDA can be automated efficiently to identify potentially weak components. We apply five steps of the vulnerability analysis phase to OpenPilot, and the results are summarized in Table 3. We assign the weights of the parameters based on the security criticality with respect to the architecture of OpenPilot.[4] For example, since Nesting Depth (ND) and Number of Children (NOC) of Code Complexity (CX) are associated with vulnerabilities, we apply weights of 2 and 3, respectively. This assessment verifies the applicability of the designed steps.

[4] Security experts can change these values if needed.

Table 3: Vulnerability analysis of OpenPilot.

| Steps | Value | Details |
|---|---|---|
| Compute Code Complexity (CX) | $CX(OpenPilot) = 52,608 + 2(6,298) + 3(6,148) = 83,648$ | The system has a total of 52,608 SLOC, 6,298 ND, and 6,148 NOC. Since ND and NOC are associated with vulnerabilities, we give them weights of 2 and 3, respectively. |
| Measure Component Coupling (CC) | $CC(OpenPilot) = 9$ | OpenPilot communicates with the Engine Control Module, Brake Control Module, Safety System, Seat Control Unit, Powertrain Control Module, Transmission Control, Telematics Control Unit, Active Front Steering, and Battery Junction Box. |
| Identify Input and Output Data Vulnerability (DV) | $DV(OpenPilot) = 242 + 2(407) + 3(397) + 5(1) + 5(1) + 15 + 2(73) + 5(211) + 5(1) = 3,478$ | OpenPilot's defined I/O are all fluctuating. The system receives and sends data using serial (inputs: 242 and outputs: 15), Controller Area Network (CAN) (inputs: 407 and outputs: 73), Global Positioning System (GPS) (inputs: 397 and outputs: 0), Vehicle to Infrastructure (V2I) (inputs: 1 and outputs: 211), and User to Vehicle (U2V) (inputs: 1 and outputs: 1) communications. We assign different weights for these communication means as they pose different risks. |
| Discover Past Security Issues (PSI) | $PSI(OpenPilot) = (0.5)^2 + 18(0.5^1) + 50(0.5^0) = 59.25$ | There are 69 reported bugs in OpenPilot reported since 2018 (1 in 2018, 18 in 2019, and 50 in 2020). Higher weight is assigned to attacks that occurred in 2020. |
| Compute Component Maturity (CM) | $CM(OpenPilot) = 100(\frac{30,688}{52,608}) = 58$ | Within OpenPilot, 30,688 SLOC is modified. None of the applied changes are labeled as security enhancement or maintenance. |

We then quantitatively evaluate the vulnerability analysis phase's effectiveness in identifying vulnerabilities in OpenPilot files. Such an examination can verify AVSDA metrics' effectiveness by comparing the files' vulnerability scores with the number of discovered vulnerabilities in every file. In total, as of October 2020, OpenPilot has 425 files and 60 documented resolved bugs. We reviewed the reported bugs and linked 24 bugs to the system files. We compare the performance of the used metrics in the AVSDA vulnerability analysis phase with two other sets of metrics. One set is code complexity and churn metrics [33], and the other set is code complexity, code coupling, and cohesion metrics [16]. We identify true-negative, true-positive, false-positive, and false-negative cases. Then, we measure accuracy, precision, and recall rates.

The results are summarized in Table 4. The AVSDA metrics outperform the other approaches in accuracy, precision, and recall. Our metrics achieve a 78% recall ratio indicating that they can identify vulnerable files efficiently. AVSDA had a notably high ratio of 94% for accuracy, indicating that the overall vulnerability identification is correct. Though the precision ratio of AVSDA is better than the other approaches, it is relatively low. This means that the number of files recognized as vulnerable and do not possess any vulnerability is high. While this causes extra unneeded work, having more false-positive cases to enhance the true-positive results is better in vulnerability identification.

We further analyze the performance of AVSDA by examining the relationship between the average metrics ratio and the number of bugs reported in a file as shown in Fig. 2. The highest number of reported bugs in a file is 6, which AVSDA identifies as the most vulnerable with a ratio of 1. As shown in Fig. 2, the Security Vulnerability (SV) assigned by the AVSDA metrics is proportional to the files'

Table 4: Comparison of AVSDA metrics with other metrics.

| | **AVSDA Metrics** | **Complexity, and Code Churn Metrics [33]** | **Complexity, Coupling, and Cohesion Metrics [16]** |
|---|---|---|---|
| True-negative | 390 | 395 | 344 |
| True-positive | 11 | 7 | 10 |
| False-negative | 3 | 7 | 4 |
| False-positive | 21 | 16 | 10 |
| Accuracy | 94% | 94% | 83% |
| Precision | 34% | 30% | 12% |
| Recall | 78% | 50% | 71% |

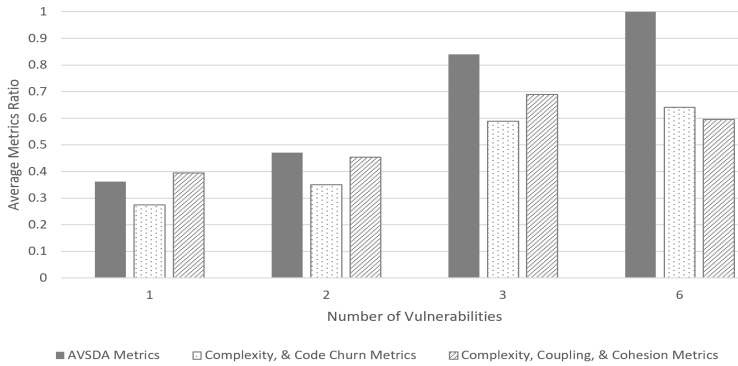


Fig. 2: Relationship between average metrics ratio and number of vulnerabilities.

number of vulnerabilities. The higher the number of vulnerabilities, the higher is the SV value. In contrast, the other two sets of metrics [16, 33] show more arbitrary behavior where files with three reported bugs are assigned a higher vulnerability value than files with six bugs.

### 4.2 Risk Analysis

To validate the risk analysis phase's applicability, we apply the steps to OpenPilot and show their usefulness. We measure the attack surface (AS) of OpenPilot by reviewing the system's inputs, outputs, channels, and methods. All the system inputs are fluctuating and OpenPilot utilizes multiple communication means, increasing the attack surface. For example, users can connect their smartphones to OpenPilot, exposing the vehicle to different remote attacks [34]. OpenPilot uses nonvolatile memory, allowing untrusted data to be stored. Accordingly, AS's level is considered as large (value of 8), and we assign a weight of 3 to emphasize the criticality of the attack surface. According to Equation (9), the estimated value of AS is 24.

Next, we estimate attacker threat (AT) using the parameters of Table 1. Attacking the Autopilot system requires outstanding experience in different domains, including networking and security. Hence, specialist skill (value of 1) is required to pose a risk on the system. Some background knowledge about the

system is required to initiate an attack. Since OpenPilot is open source, we assign the knowledge parameter a value of 8. Moreover, the equipment needed to conduct an attack is standard (value of 8), like a computer and ports. Finally, performing an attack on the Autopilot system requires preplanning, and either physical or remote access is needed (value of 3). With these level values, AT is 20. The likelihood of an attack (LA) can now be estimated 480 based on Equation (11).

Next, we determine the impact level of an attack (IL). Establishing an attack on an autopilot system might not have severe direct consequences but can lead to drastic indirect results. The vehicle can operate without autopilot functionality. However, such functionality communicates with critical components (e.g., engine and brake ECUs). If a malicious attack successfully propagates, the vehicle's safety and operational status are left in critical condition. Hence, the safety and operational parameters are both at a high level (value of 8). The manufacturing company might face significant financial losses (value of 3) when all the models affected by such an attack are recalled. Moreover, OpenPilot collects data, including locations, Controller Area Network (CAN) messages, and road conditions. The leakage of such data can violate the privacy of affected users *only* (value of 3). Finally, such attacks have a moderate reputational impact, with some possible customer loss (value of 3). After estimating all the parameters of Table 2, the impact level (IL) can be determined using Equation (12). We assign a weight of 4 for the safety and operational parameters to emphasize their importance, and the final value of IL is 73.

The last step is determining the security risk (SR) based on the likelihood of an attack, impact level, and practiced security measures. First, we evaluate SR using Equation (13), which results in 35,040. Then we review OpenPilot's security practices to identify the SR level. OpenPilot follows MISRA c2012 [10] software development guidelines, preventing common coding errors. However, this is not enough to mitigate all security issues. Accordingly, we assign a moderate SR level, which indicates that an attack risk exists.

### 4.3 Framework Application Frequency

The AVSDA framework should be applied periodically to identify security decay. For example, consider a vehicle attack technique becomes publicly available with a video explaining how to accomplish the attack. The effect of such an incident on the vehicle system's security can be detected by the AVSDA framework. Attacker threat skill parameter is changed from a specialist to a non-specialist with a value of 8. Accordingly, AT rises to 27, increasing the likelihood of an attack to 648. The system's security risk extends from 35,040 to 47,304, indicating a security decay and the need for applying robust security measures to defend the vehicle.

## 5 Conclusion

We propose an Autonomous Vehicle Security Decay Assessment (AVSDA) framework that estimates the security decay of vehicle software systems by quanti-

tatively measuring systems' vulnerabilities and risks. The AVSDA framework is composed of two phases. The vulnerability analysis phase uses security metrics to identify vulnerable components. The risk analysis phase carefully evaluates attack likelihood by identifying the attack surface and estimating attackers' threats. The framework further analyzes attacks' severity by assessing their impact. The final step of the risk analysis phase defines security risk based on the applied security measures. The AVSDA framework should be applied periodically to recognize changes in the security risk and possible decay.

Though AVSDA is highly effective at identifying likely locations of software defects that lead to vulnerabilities, a software focus risk analysis cannot address certain classes of vehicle attacks that do not target vulnerabilities. For example, sensor spoofing attacks, sybil attacks, and replay attacks are not software defects related attacks and cannot be estimated by the AVSDA framework.

We evaluated AVSDA vulnerability analysis phase metrics' performance by experimenting with their usefulness in identifying vulnerabilities of OpenPilot, an Autopilot system. The results show that the framework is capable of identifying vulnerabilities with an accuracy rate of 94%. The case study shows the efficiency of AVSDA in systematically estimating security risks and discovering security decay.

## Acknowledgment

This work is partially supported by Irdeto, the Natural Sciences and Engineering Research Council of Canada (NSERC), and the Canada Research Chairs (CRC) program.

## References

1. AUTOSAR enabling continuous innovations, https://www.autosar.org/
2. Black hat USA 2015: The full story of how that jeep was hacked, https://www.kaspersky.com/blog/blackhat-jeep-cherokee-hack-explained/9493/
3. ISO/IEC 18045:2005 information technology — security techniques — methodology for it security evaluation, https://www.iso.org/standard/30830.html
4. ISO/SAE DIS 21434 road vehicles cybersecurity engineering, https://www.iso.org/standard/70918.html
5. Openpilot, https://comma.ai/
6. Openpilot source code, https://github.com/commaai/openpilot
7. Q1 2019 sees a rapid growth of automotive cyber incidents, https://www.upstream.auto/blog/q1-2019-sees-a-rapid-growth-of-automotive-cyber-incidents/
8. Society of automotive engineers, https://www.sae.org/
9. UNECE WP.29-Introduction, https://unece.org/wp29-introduction
10. What is MISRA?, https://www.misra.org.uk/MISRAHome/WhatisMISRA/tabid/66/Default.aspx
11. What is the ISO 26262 functional safety standard?, https://www.ni.com/en-ca/innovations/white-papers/11/what-is-the-iso-26262-functional-safety-standard-.html

12. Alberts, C.J., Dorofee, A.J.: Managing information security risks: the OCTAVE approach. Addison-Wesley Professional (2003)
13. Antinyan, V., Staron, M., Sandberg, A.: Evaluating code complexity triggers, use of complexity measures and the influence of code complexity on maintenance time. Empirical Software Engineering **22**(6), 3057–3087 (2017)
14. Ben Othmane, L., Ranchal, R., Fernando, R., Bhargava, B., Bodden, E.: Incorporating attacker capabilities in risk estimation and mitigation. Computers & Security **51**, 41–61 (2015)
15. Burton, S., Likkei, J., Vembar, P., Wolf, M.: Automotive functional safety= safety+ security. In: Proceedings of the First International Conference on Security of Internet of Things. pp. 150–159 (2012)
16. Chowdhury, I., Zulkernine, M.: Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities. Journal of Systems Architecture **57**(3), 294–313 (2011)
17. Committee, S.V.E.S.S., et al.: SAE j3061-cybersecurity guidebook for cyber-physical automotive systems. SAE-Society of Automotive Engineers (2016)
18. Durisic, D., Nilsson, M., Staron, M., Hansson, J.: Measuring the impact of changes to the complexity and coupling properties of automotive software systems. Journal of Systems and Software **86**(5), 1275–1293 (2013)
19. Giger, E., Pinzger, M., Gall, H.C.: Comparing fine-grained source code changes and code churn for bug prediction. In: Proceedings of the 8th Working Conference on Mining Software Repositories. pp. 83–92 (2011)
20. Henniger, O., Apvrille, L., Fuchs, A., Roudier, Y., Ruddle, A., Weyl, B.: Security requirements for automotive on-board networks. In: 2009 9th International Conference on Intelligent Transport Systems Telecommunications, (ITST). pp. 641–646 (2009)
21. Islam, M.M., Lautenbach, A., Sandberg, C., Olovsson, T.: A risk assessment framework for automotive embedded systems. In: Proceedings of the 2nd ACM International Workshop on Cyber-Physical System Security. pp. 3–14 (2016)
22. Kotenko, I., Chechulin, A.: A cyber attack modeling and impact assessment framework. In: 2013 5th International Conference on Cyber Conflict (CYCON 2013). pp. 1–24. IEEE (2013)
23. Macher, G., Armengaud, E., Brenner, E., Kreiner, C.: A review of threat analysis and risk assessment methods in the automotive context. In: International Conference on Computer Safety, Reliability, and Security. pp. 130–141. Springer (2016)
24. Macher, G., Armengaud, E., Brenner, E., Kreiner, C.: Threat and risk assessment methodologies in the automotive domain. Procedia Computer Science **83**, 1288–1294 (2016)
25. Manadhata, P.K., Wing, J.M.: An attack surface metric. IEEE Transactions on Software Engineering **37**(3), 371–386 (2010)
26. Medeiros, N., Ivaki, N., Costa, P., Vieira, M.: Software metrics as indicators of security vulnerabilities. In: 2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE). pp. 216–227. IEEE (2017)
27. Mössinger, J.: Software in automotive systems. IEEE software **27**(2), 92–94 (2010)
28. Moukahal, L., Zulkernine, M.: Security vulnerability metrics for connected vehicles. In: 2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C). pp. 17–23
29. Moukahal, L.J., Elsayed, M.A., Zulkernine, M.: Vehicle software engineering (VSE): Research and practice. IEEE Internet of Things Journal **7**(10), 10137–10149 (2020)

30. Nighswander, T., Ledvina, B., Diamond, J., Brumley, R., Brumley, D.: GPS software attacks. In: Proceedings of the 2012 ACM conference on Computer and communications security. pp. 450–461 (2012)
31. 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 Project (2009)
32. Salfer, M., Eckert, C.: Attack surface and vulnerability assessment of automotive electronic control units. In: 2015 12th International Joint Conference on E-Business and Telecommunications (ICETE). vol. 4, pp. 317–326. IEEE (2015)
33. Shin, Y., Meneely, A., Williams, L., Osborne, J.A.: Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities. IEEE Transactions on Software Engineering **37**(6), 772–787 (2010)
34. Sommer, F., Dürrwang, J., Kriesten, R.: Survey and classification of automotive security attacks. Information **10**(4), 148 (2019)
35. Tangade, S.S., Manvi, S.S.: A survey on attacks, security and trust management solutions in VANETs. In: 2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT). pp. 1–6. IEEE (2013)
36. Thing, V.L., Wu, J.: Autonomous vehicle security: A taxonomy of attacks and defences. In: 2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData). pp. 164–170. IEEE (2016)