



Cyber-security of connected vehicles : contributions to enhance the risk analysis and security of in-vehicle communications

Khaled Karray

► To cite this version:

Khaled Karray. Cyber-security of connected vehicles : contributions to enhance the risk analysis and security of in-vehicle communications. Cryptography and Security [cs.CR]. Université Paris Saclay (COmUE), 2019. English. NNT : 2019SACLT023 . tel-02947030

HAL Id: tel-02947030

<https://pastel.hal.science/tel-02947030>

Submitted on 23 Sep 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Cyber-security of Connected Vehicles: Contributions to enhance the Risk Analysis and Security of in-Vehicle Communications.

Thèse de doctorat de l'Université Paris-Saclay
préparée à Institut Mines-Télécom - Télécom ParisTech
Ecole doctorale n°580 Sciences et Technologies de l'Information et de la communication (STIC)

Spécialité de doctorat : Réseaux, Information, Communications

Thèse présentée et soutenue à Télécom ParisTech, le 01-04-2019, par

Khaled KARRAY

Composition du Jury :

Sjouke MAUW Professeur, Université de Luxembourg	Rapporteur
Vincent NICOMETTE Professeur, INSA de Toulouse (LAAS)	Rapporteur
Houda LABIOD Professeur, Télécom ParisTech (LTCl)	Président
Eric DEQUI Maître expert Cyber sécurité, PSA-GROUPE	Examineur
Witold KLAUDEL Expert Cyber sécurité, GROUPE RENAULT	Examineur
Sylvain GUILLEY Directeur technique Secure-IC Professeur, Télécom ParisTech (LTCl)	Directeur de thèse
Jean-luc DANGER Professeur, Télécom ParisTech (LTCl)	Co-Directeur de thèse
Abdelaziz MOULAY ELAABID Expert Cyber sécurité, PSA-GROUPE	Encadrant industriel
Ludovic APVRILLE Professeur, Télécom ParisTech (LTCl)	Invité

Titre : Cybersécurité des véhicules connectés: contributions pour améliorer l'analyse des risques et la sécurité des communications embarquées.

Mots clés : Analyse de risque, Arbre d'attaques, Obfuscation sur CAN, Detection d'intrusion.

Résumé : Au cours de la dernière décennie, les progrès technologiques ont rendu la voiture de plus en plus autonome et connectée au monde extérieur. D'un autre côté, cette transformation technologique a soumis les véhicules modernes à des cyber-attaques avancées. Les architectures cyber-physiques des systèmes automobiles n'ont pas été conçues dans un souci de sécurité. Avec l'intégration de plates-formes connectées dans ces systèmes cyber-physiques, le paysage des menaces a radicalement changé. Dernièrement, plusieurs atteintes à la sécurité visant différents constructeurs automobiles ont été signalées principalement par la communauté scientifique. Cela fait de la sécurité une préoccupation essentielle, avec un impact important, en particulier sur la future conduite autonome. Afin de remédier à cela, une ingénierie de sécurité rigoureuse doit être intégrée au processus de conception d'un système automobile et de nouvelles méthodes de protections adapté

aux spécificités des systèmes véhiculaire doivent être introduite. La modélisation des menaces et l'analyse des risques sont des éléments essentiels de ce processus. Pour ce faire, les arbres d'attaque se sont avérés un moyen raisonnable de modéliser les risques. Néanmoins, étant donné la diversité des architectures, élaborer des arbres d'attaque pour toutes les architectures peut rapidement devenir un fardeau. Cette thèse aborde la problématique de la sécurité des véhicules connectés. L'approche présentée consiste à améliorer la méthodologies d'évaluation de la sécurité par la génération automatique d'arbres d'attaques pour assister à l'étape d'analyse de risques. On propose aussi de nouvelles méthodes de protections des réseaux internes véhiculaire capable de faire face aux attaques cyber-physiques existante.

Title : Cyber-security of Connected Vehicles: Contributions to enhance the Risk Analysis and Security of in-Vehicle Communications.

Keywords : Risk Analysis, Attack Tree, Obfuscation on CAN, Intrusion Detection.

Abstract : During the last decade, technological advances have made the car more and more connected to the outside world. On the flip side, this technological transformation has made modern vehicles subject to advanced cyber attacks. The cyber-physical architectures of automotive systems were not designed with security in mind. With the integration of connected platforms into these cyber-physical systems, the threat landscape has radically changed. Lately, multiple security breaches targeting different car manufacturers have been reported mainly by the scientific community. This makes security a critical concern, with a high impact especially on future autonomous driving. In order to address this gap, rigorous security engineering needs to be integrated into the design process of an

automotive system and new protection methods adapted to the specificities of the vehicle systems must be introduced. Threat modeling and risk analysis are essential building blocks of this process. In this context, attack trees proved to be a reasonably good way to model attack steps. Nevertheless, given the diversity of architectures, it can quickly become a burden to draw attack trees for all architectures. This thesis tackles the issues of security of connected vehicles. The proposed approach allows enhancing the threat analysis with the automated generation of attack tree used to assist in the risk assessment step. We also propose novel and efficient protection mechanisms for in-vehicle communication networks capable of coping with existing cyber-physical attacks.

Rremerciments

Ma thèse s'est déroulée en partie à **Telecom Paris** au sein du département COMELEC, en partie à **PSA-Groupe** au sein du service de Cyber Sécurité et de sureté de fonctionnement. Avant de remercier les personnes qui m'ont aidée dans ma thèse, j'aimerais remercier **PSA-Groupe** et **Telecom Paris** pour avoir financé ces travaux de recherche.

Tout d'abord, je tiens à remercier **Dr. Moulay Abdelaziz Elaabid**, **Prof. Jean-Luc Danger** et **Prof. Sylvain Guilley** pour m'avoir proposé cette opportunité et avoir rendu possible cette expérience enrichissante. Tous les trois ont activement participé à l'encadrement et au bon déroulement de cette thèse.

Prof. Sylvain Guilley s'est avéré un directeur de thèse remarquable. Je le remercie pour sa grande disponibilité, en particulier dans les moments clés. Les discussions avec Sylvain ont toujours été bénéfiques et prolifiques. Sa faculté d'écoute et de compréhension en font un excellent directeur de thèse.

Je tiens également à remercier mon encadrant de thèse **Prof. Jean-Luc Danger** pour le suivis continu, son approche pédagogique ainsi que sa grande disponibilité. Je le remercie aussi de sa bonne humeur et de la bonne ambiance de travail qu'il a su instaurer. Je tiens aussi à souligner la facilité avec laquelle il est possible de discuter avec lui. Les réunions avec Jean-Luc ont toujours été un plaisir et l'assurance de conseils avisés.

Enfin et surtout, je remercie **Dr. Moulay Abdelaziz Elaabid** pour avoir assuré l'encadrement industriel de la thèse. Je le remercie pour les discussions qui m'ont aidé à mieux comprendre les problématiques et les enjeux industriel. Je le remercie également pour ses qualités personnelles et tous ses conseils qui m'ont aidé à mieux appréhender et naviguer les procédures d'une grande entreprise telle que PSA-Groupe. Son aide m'a également été précieuse dans l'organisation et la valorisation de mon travail.

Je remercie avec une grande considération les membres de mon jury de thèse **Sjouke MAUW**, **Vincent NICOMETTE**, **Houda LABIOD**, **Eric DEQUI**, **Witold KLAUDEL** et **Ludovic APVRILLE** pour avoir accepté et pris le temps d'évaluer mes travaux. Un merci particulier à **Prof. Sjouke MAUW** et à **Prof. Vincent NICOMETTE** pour avoir accepté d'être rapporteurs de ma thèse.

Un grand merci à mes collègues de Telecom Paris et de PSA-Groupe : Youssef Souissi, Oualid Trabelsi, Sofiane Takarabt, Wei Cheng, Alexander Schaub, Michaël Timbert, Xuan Thuy Ngo, Sébastien Carré, Annelie Heuser, Laurent Sauvage, Nicolas Bruneau, Margaux Dugardin, Eric Dequi, Jean de Baudreuil, Moulay Abdelaziz Elabid, Béatrice Gardin-Balaÿ, Yves Le Bobinnec, Cédric Wilwert, avec qui j'ai partagé les longs déjeunés et les pause cafés, ceux qui m'ont aider comprendre le monde de la recherche et de l'industrie.

Plus personnellement, je remercie tous mes amis et ma famille pour leur soutien inconditionnel. Mes remerciements les plus profonds vont évidemment à mes parents Walid KARRAY et Hekma MARRAKCHI, qui m'ont toujours encouragée dans mes études. C'est grâce à eux deux que j'ai cet amour des sciences. Ils m'ont toujours laissée libre de mes choix et ont toujours été de bons conseils et modèles. Merci pour la confiance qu'ils m'ont accordée. Un grand merci à mes deux frères, je sais que je peux toujours compter sur eux.

During the last decade, technological advances have made the car more and more autonomous and connected to the outside world, and this will even accelerate in the near future. On the flip side, this technological transformation has made modern vehicles subject to advanced cyber attacks. The cyber-physical architectures of automotive systems were not designed with security in mind. It is primarily due to the fact cars were closed systems, and internal communication buses were considered to be trusted. With the integration of connected platforms into these cyber-physical systems, the threat landscape has radically changed. Lately, multiple security breaches targeting different car manufacturers have been reported mainly by the scientific community. This makes security a critical concern, with a high impact especially on future autonomous driving which will arrive soon.

In order to address this gap, rigorous security engineering needs to be integrated into the design process of an automotive system. Threat modeling and risk analysis are essential building blocks of this process. The security experts have to elaborate attack scenarios and assess their impact and likelihood. This imposes a step to consider every possible way to attack the vehicular system to reach a particular goal. In order to do this, attack trees proved to be a reasonably good way to model attack steps and to assist the designer in risk assessment. Besides, this formalism presents some similarities with Fault Trees that are widely used in safety risk assessment in the automotive industry. This makes it easier to adopt for familiarity reasons. Nevertheless, given the diversity of architectures, it can quickly become a burden to draw attack trees for all architectures. The first part of the thesis tackles this particular problem. An updated state of the art of attacks targeting modern vehicles is presented. This gives a deeper understanding of the threat landscape, the complexity of the attack steps and attackers profiles. A formal model is then established that takes as input the car architectural model in terms of services, hardware, and network connections as well as an attacker model represented as a set of basic attacks. This formal model is then used to automatically generate possible attack paths that can be exploited by a potential adversary. As a result attack trees are automatically generated given an architectural model and an attacker model. These attack trees are then used as a starting point for accurate risk assessment. Consequently, the designer will be able to decide what are the threats that need to be covered and what are the threats

that can be accepted as part of the residual risk and depending on the compromise with the implementation cost.

Following the first contribution, there was an interesting observation that gaining access to internal communication buses of the vehicle is a central step of an important set of attacks. This motivated the second part of the thesis which introduces novel techniques to protect from and detect attacks on internal communication buses. The controller area network being the most used, we concentrate our work on this protocol. For an attacker model that has physical access to the CAN bus, we design the first solution based on identifier randomization that is capable of protecting the system against reverse engineering, replay, and injection attacks. This solution is evaluated using information theoretic metrics such as the entropy and conditional entropy of the identifiers and compared to similar state-of-the-art techniques. It is found that the proposed methods outperform known solutions and give the optimal protection level.

Second, we investigate intrusion detection methods as one of the promising security solutions. We present the different methods that are being used today for detection and prevention. Then we propose a machine learning-based detection method that protects against a more advanced attacker model that has indirect and remote physical access to the internal communication buses. This solution is built by training using CAN log traces collected from several hours of drive tests. It is also tested against state of the art attacks. Results show that it has a high detection rate, and is more robust than known detection mechanisms.

Au cours de la dernière décennie, les progrès technologiques ont permis à la voiture d'être de plus en plus connectée au monde extérieur. En contrepartie, ces nouvelles fonctionnalités de communication ont ouverts la voie aux cyber-attaques sur les véhicules modernes. Le problème vient du fait que les architectures des systèmes automobiles n'ont pas été conçues dans un souci de sécurité. En effet, les voitures ont toujours été considérées jusqu'à lors comme des systèmes isolés avec des bus de communication internes jugés de confiance. Avec l'intégration de plates-formes connectées dans ces systèmes cyber-physiques, le paysage des menaces a radicalement changé. Dernièrement, plusieurs atteintes à la sécurité visant différents constructeurs automobiles ont été signalées. La sécurité est ainsi devenue une préoccupation essentielle, en particulier sur la future conduite autonome.

Afin de remédier à cela, une ingénierie de sécurité rigoureuse doit être intégrée au processus de conception d'un système automobile. La modélisation des menaces et l'analyse des risques sont des éléments essentiels de ce processus. Les experts de la cyber-sécurité doivent élaborer des scénarios d'attaque et évaluer leur impact et leur probabilité de succès. Cela impose de considérer tous les moyens possibles d'attaquer le système véhiculaire pour atteindre un objectif particulier. Pour ce faire, les arbres d'attaque représentent un moyen pour modéliser les attaques et aider le concepteur à évaluer les risques et s'en prémunir. Néanmoins, étant donné la diversité des architectures, élaborer des arbres d'attaque pour toutes les architectures peut rapidement devenir un fardeau. La première partie de la thèse aborde ce problème particulier. Un modèle formel est ainsi établi, qui prend en entrée le modèle architectural de la voiture ainsi qu'un modèle d'attaquant. Ce modèle formel est ensuite utilisé pour générer automatiquement des chemins d'attaque pouvant être exploités par un adversaire. En conséquence, les arbres d'attaque sont automatiquement générés. Ces arbres d'attaque sont ensuite utilisés comme point de départ pour une évaluation précise des risques. Par conséquent, le concepteur devient en mesure de décider quelles sont les menaces à couvrir et celles qui peuvent être acceptées comme faisant partie du risque résiduel.

Suite à la première contribution, on a pu noter que l'accès aux bus de communication internes du véhicule est un moyen d'accès commun à un ensemble important d'attaques.

Cela a motivé la deuxième partie de la thèse qui introduit de nouvelles techniques pour protéger et détecter les attaques sur les bus de communication internes. Pour un modèle d'attaquant disposant d'un accès physique au bus CAN, nous avons étudié des solutions basées sur l'obfuscation des identifiants, capable de protéger le système contre les attaques en rétro-conception et d'injection de trames. Cette solution est évaluée à l'aide de métriques issues de la théorie de l'information telles que l'entropie et l'entropie conditionnelle des identifiants. Elle est ensuite comparée à des techniques de l'état de l'art. Il a été constaté que les méthodes proposées surpassent les solutions connues et donnent de meilleurs résultats.

De façon à contrer les attaques tirant parti des accès distants, nous avons étudié des méthodes de détection d'intrusion faisant parties des solutions de sécurité prometteuses. Ces différentes méthodes peuvent être utilisées à la fois pour la détection et la mise en place de protections en cas d'alarme. Nous avons notamment proposé une méthode de détection basée sur l'apprentissage automatique. Cette solution repose sur un entraînement utilisant les traces du bus CAN collectées pendant plusieurs heures de tests de conduite. La méthode a également été évaluée contre des attaques simulées. Les résultats montrent qu'elle présente un taux de détection très élevé et est ainsi plus robuste que les mécanismes de détection connus.

List of Figures	xv
List of Tables	xix
List of Abbreviations	1
1 Introduction	3
1.1 Context: security of connected vehicles	3
1.2 The automotive industry challenges	5
1.3 Motivation and goals	5
1.4 Contributions	6
1.4.1 Formal modeling approach for automatic attack tree generation	6
1.4.2 CAN identifier randomization strategy	6
1.4.3 Prediction-based intrusion detection system	7
1.5 Outline	7
2 State of the Art	9
2.1 Introduction	9
2.2 Cyber-physical architecture of connected cars	11
2.2.1 Sensors	11
2.2.2 Actuators	11
2.2.3 Electronic Control Units	11
2.2.4 Communication interfaces	12
2.2.4.1 In-vehicle shared communication buses	12
2.2.4.2 Diagnostics interface	12
2.2.4.3 Communication with the outside world	13
2.2.5 Overall architecture	13
2.2.6 Aftermarket and diagnostics Devices	13
2.3 Vulnerabilities and threats survey	14
2.3.1 Vulnerabilities and attack vectors	14

CONTENTS

2.3.1.1	Direct physical access	15
2.3.1.2	Indirect physical access	16
2.3.1.3	Wireless access	16
2.3.2	Threats	17
2.3.2.1	Security violations	17
2.3.2.2	Attacker motivation	17
2.4	Countermeasures and Security methodologies	18
2.4.1	Countermeasures	18
2.4.1.1	Architecture	18
2.4.1.2	Asset protection and data security	20
2.4.1.3	Policy enforcement and run-time protections	20
2.4.2	Threat Analysis and Risk Assessment	21
2.5	In-Vehicle Secure Communication survey	25
2.5.1	Controller Area Network Overview	27
2.5.2	CAN Weaknesses	31
2.5.2.1	Denial-of-Service	31
2.5.2.2	Reverse engineering	32
2.5.2.3	Fuzzing attack	33
2.5.2.4	Impersonation attack	33
2.5.2.5	Exhaustion attack	33
2.5.3	Protection mechanisms	34
2.5.3.1	Payload protection	35
2.5.3.2	Identifier protection	37
2.5.3.3	Intrusion Detection and Prevention Systems	38
2.5.4	Advantages and disadvantages	40
2.6	Conclusion	42
3	Risk analysis and Attack tree generation	43
3.1	Introduction	43
3.2	Attack trees	45
3.2.1	Presentation and formal definition	45
3.2.2	Attack tree generation problem	48
3.2.2.1	Attack trees in the automotive domain	48
3.2.2.2	Attack tree generation in other application domains	48
3.3	A case study: speed acquisition and display system	49
3.3.1	Description	49
3.3.2	Goal of the attacker: forge displayed vehicle speed	50
3.4	Cyber-physical architecture formal model	50
3.4.1	Data	50
3.4.2	Communication mediums	51
3.4.3	Hardware components	51
3.4.4	Service components	52
3.4.5	Attacker	53
3.4.6	Architecture	54
3.4.7	Security properties	56
3.4.8	Case-study formal model	57

3.5	Graph transformation system	58
3.5.1	Definition	58
3.5.2	Labeled transition system	60
3.5.3	Tool support: GROOVE	60
3.5.4	Case-study graph transformation system	60
3.6	Attack tree transformation	62
3.6.1	Attack graph generation	62
3.6.2	Attack tree generation:	64
3.6.3	Case-study generation of attacks	65
3.7	Security analysis and Countermeasure	68
3.7.1	Security analysis	68
3.7.2	Countermeasures:	72
3.8	Conclusion	73
4	Identifier Randomization: an Efficient Protection against CAN-bus At-	77
	tacks	
4.1	Introduction	77
4.2	General formalism of ID-based protection	78
4.3	Evaluation metrics	80
4.3.1	Reverse-engineering attack	80
4.3.2	Replay and injection attacks	81
4.4	Proposed solutions	81
4.4.1	The IA-CAN Approach	81
4.4.2	Equal Intervals	83
4.4.3	Frequency Intervals	86
4.4.4	Dynamic Intervals	88
4.4.5	Arithmetic Masking	90
4.5	Comparison	93
4.6	Conclusion	96
5	On-board Intrusion Detection and Prevention system	97
5.1	Introduction	97
5.2	Machine learning algorithms	99
5.2.1	Learning strategy	99
5.2.2	Parametric and non-parametric models	99
5.3	Principle and problem formulation	100
5.3.1	Signal types	100
5.3.2	Intrusion detection principle	101
5.3.3	Mathematical formulation	103
5.3.4	Real-valued signal	104
5.3.5	Categorical signal	105
5.4	Validation metrics	105
5.4.1	Regression metrics for real-valued signals:	105
5.4.2	Classification metrics for categorical signals:	107
5.5	Supervised learning algorithms	108
5.5.1	K-Nearest Neighbor	108

CONTENTS

5.5.1.1	KNN for regression	109
5.5.1.2	KNN for classification	109
5.5.2	Decision tree	110
5.5.2.1	Regression Trees	111
5.5.2.2	Classification Trees	112
5.5.3	Artificial Neural Network	112
5.5.3.1	MLP for Regression	115
5.5.3.2	MLP for Classification	115
5.6	Data collection and feature engineering	116
5.6.1	Experimental set-up	116
5.6.2	Data collection	116
5.6.3	Feature engineering	118
5.7	Experimental validation and discussion	119
5.7.1	Predicting a real-valued signal	119
5.7.1.1	Speed signal	119
5.7.1.2	Capturing nominal behavior of the speed signal	120
5.7.2	Predicting a categorical signal	127
5.7.2.1	Brake lights command signal	127
5.7.2.2	Capturing nominal behavior of the brake-lights-command signal	127
5.7.3	Unification of detection rule	131
5.8	Evaluation against attacks	131
5.8.1	Simulation of attacks	132
5.8.2	Attacks against real-valued signal	132
5.8.2.1	Random speed injection attack	132
5.8.2.2	Speed offset injection attack	133
5.8.2.3	Speed Denial of service (signal drop) attack	134
5.8.3	Attacks against categorical signal	134
5.8.3.1	Random command injection attack	135
5.8.3.2	Inverse command injection attack	135
5.8.3.3	Denial of service (force to 0) attack	135
5.9	Alerts handling	136
5.9.1	Prevention mechanism	136
5.9.2	False positives reduction strategy	137
5.10	Conclusion and discussion	138
6	Conclusion	139
6.1	Summary	139
6.2	Perspectives and future research directions	140
6.2.1	On risk assessment	140
6.2.2	On in-vehicle secure communications	140
	Appendices	145
	Transforamtion rules	145
	Entropy computations	151
	Bibliography	159

List of Figures

1.1	Conceptual diagram of the connected vehicle environment	4
2.1	Functionalities and services of Modern cars [1]	10
2.2	Example of cyber-physical architecture [95]	14
2.3	The EASIS project proposal architecture	19
2.4	SAE-j3061 concept phase steps [32]	22
2.5	CAN layer model	27
2.6	CAN frame	28
2.7	CAN controller	29
2.8	Error handling in the CAN bus	30
2.9	Attacker models	35
2.10	High-level synthesis of detection mechanisms applied to the CAN frame	38
3.1	Attack tree decomposition principle	46
3.2	Attack tree for unauthorized active braking [52]	47
3.3	Architectural graph of the speed acquisition and display case-study	61
3.4	Speed acquisition rule (R1)	61
3.5	Speed send to CAN rule (R2)	62
3.6	CAN send rule (R3)	62
3.7	Example of attacker rule	63
3.8	Application of transformation rules (state space exploration)	63
3.9	Query: False speed	64
3.10	Examples of Attack graph to Attack tree transformation	65
3.11	Attack tree automatically produced from the input model Figure 3.3	67
3.12	Attack tree automatically produced from Architecture-1	73
3.13	The overall approach to attack tree generation	74
4.1	Controller Area Network with original identifier distribution	78
4.2	CAN-ID randomization principle	80
4.3	Illustration of the IA-CAN identifier transformation approach	82
4.4	IA-CAN transformation: Original (Left) Randomized (Right)	83

LIST OF FIGURES

4.5	Illustration of the Equal intervals identifier transformation	85
4.6	Equal intervals transformation: Original (Left) and Randomized (Right) . . .	85
4.7	Illustration of the Frequency Intervals identifier transformation	87
4.8	Frequency intervals transformation: Original (Left) and Randomized (Right)	88
4.9	Illustration of the Dynamic intervals identifier transformation at $t + 1$ (Left) and $t + 2$ (Right)	90
4.10	Dynamic intervals transformation: Original (Left) and Randomized (Right) .	91
4.11	Illustration of the Arithmetic masking identifier transformation	92
4.12	Arithmetic masking transformation: Original (Left) and Randomized (Right)	93
4.13	Conditional entropy $H(id_r id_o) = f(N)$	96
5.1	Machine learning taxonomy	100
5.2	Example of real-valued and categorical signals	101
5.3	Prediction principle	102
5.4	Model choice depending on the target signal type.	103
5.5	Gaussian shaped prediction error	107
5.6	Example of a decision tree	111
5.7	Illustration of a feed-forward artificial neural network structure with an input layer, one hidden layer, and an output layer.	113
5.8	One hidden neuron	114
5.9	Multilayer perceptron structure	114
5.10	Illustration of the data acquisition plateforme	117
5.11	Parsing the log file and building the training data.	118
5.12	Accuracy (Acc^{reg} %) of KNN algorithm as a function of the number of neighbors	121
5.13	Accuracy (Acc_{tp}^{reg} %) of KNN algorithm as a function of the number of neighbors	121
5.14	Mean of the prediction error μ_ϵ of KNN algorithm as a function of the number of neighbors	121
5.15	Standard deviation of the prediction error σ_ϵ of KNN algorithm as a function of the number of neighbors	121
5.16	Accuracy (Acc^{reg} %) of decision tree algorithm as a function of the tree depth	123
5.17	Accuracy (Acc_{tp}^{reg} %) of decision tree algorithm as a function of the tree depth ($tp = \pm 5 \text{ km/h}$): True Negative.	123
5.18	Mean of the prediction error μ_ϵ of decision tree algorithm as a function of the tree depth	123
5.19	Standard deviation of the prediction error σ_ϵ of decision tree algorithm as a function of the tree depth	123
5.20	Accuracy (Acc^{reg} %) of Neural-Network with logistic perceptron algorithm as a function of the number of neurons in the first layer.	124
5.21	Accuracy (Acc_{tp}^{reg} %) of Neural-Network with logistic perceptron algorithm as a function of the number of neurons in the first layer.	124
5.22	Mean of the prediction error μ_ϵ of Neural-Network with logistic perceptron algorithm as a function of the number of neurons in the first layer.	124
5.23	Standard deviation of the prediction error σ_ϵ of Neural-Network with logistic perceptron algorithm as a function of the number of neurons in the first layer.	124
5.24	Accuracy (Acc^{reg} %) of Neural-Network with Relu perceptron algorithm as a function of the number of neurons in the first layer.	125

5.25	Accuracy ($Acc_{t_p}^{reg}$ %) of Neural-Network with Relu perceptron algorithm as a function as a function of the number of neurons in the first layer.	125
5.26	Mean of the prediction error μ_ε of Neural-Network with Relu perceptron algorithm as a function of the number of neurons in the first layer.	125
5.27	Standard deviation of the prediction error σ_ε of Neural-Network with Relu perceptron algorithm as a function of the number of neurons in the first layer.	125
5.28	Accuracy Acc of the KNN classification algorithm as a function of the number of neighbors	128
5.29	Accuracy Acc of the decision tree classification algorithm as a function of the tree depth	129
5.30	Accuracy Acc of the Neural network with logistic perceptron classification algorithm as a function of the number of neurons in the first layer	129
5.31	Accuracy Acc of the Neural network with Relu perceptron classification algorithm as a function of the number of neurons in the first layer	130
5.32	Illustration of <i>Man-in-the-middle attack</i> principle on the Speed signal	132
5.33	Evolution of the original speed signal (blue) and random speed signal (red) over time.	133
5.34	Alerts raised by the decision tree (depth=40) detection rule against random speed injection attack	133
5.35	Evolution of the original speed signal (blue) and offset speed signal (red) over time.	133
5.36	Alerts raised by the decision tree (depth=40) detection rule against speed offset injection attack	133
5.37	Evolution of the original speed signal (blue) and frozen speed signal (red) over time.	134
5.38	Alerts raised by the decision tree (depth=40) detection rule against random speed injection attack	134
5.39	Alerts raised by the decision tree (depth=40) detection rule tested on three different attacks on the <i>brake-lights-command</i> signal. On top is the ground truth command, in the middle is the attack command and on the bottom is the Alerts raised by the detection rule when receiving the attack signal. . . .	136
5.40	Comparison between the ground truth signal and the predicted signal (signal predicted with decision tree (depth=40) regression algorithm	137
5.41	Characterization of the ratio of false alerts (in percentage%) raised as a function of the number of successive false positives.	137

LIST OF FIGURES

List of Tables

2.1	In-vehicle communication bus technologies	12
2.2	Severity rating of threats in the EVITA approach	23
2.3	Rating of aspects of attack potential [52]	24
2.4	Controllability classes	25
2.5	Risk level as a function of the attack probability, severity level, and controllability	26
2.6	Summary of CAN Weaknesses	34
2.7	Intrusion detection techniques applied to CAN	41
3.1	Rating of the basic attacks	69
3.2	Estimates of attack potential of the identified attacks	71
4.1	Comparison between different randomization strategies	94
5.1	Detection metrics	119
5.2	Linear Regression detection rule [results are averaged over 10 runs]	122
5.3	Prediction accuracy of detection rules for $tp = \pm 5 \text{ km/h}$ trained and tested with data captures from three different drive tests	126
5.4	Logistic Regression detection rule	128
5.5	Prediction Accuracy of detection rules for the <i>brake-lights-command</i> signal . .	130
5.6	Prediction Accuracy of the unified detection rules for the <i>speed</i> (Decision Tree with Tree depth= 40)	131
5.7	Prediction Accuracy of the unified detection rules for the <i>brake-lights-command</i> (Decision Tree with Tree depth= 40)	131
5.8	False Negative rate of simulated attacks on the <i>Speed</i> signal	134
5.9	False Negative rate of simulated attacks on the <i>brake-lights-command</i> signal	135

LIST OF TABLES

List of Abbreviations

ECU	Electronic Control Unit
CAN	Controller Area Network
MOST	Controller Area Network
CPS	Cyber Physical System
SOF	Start Of Frame
EOF	End Of Frame
DLC	Data Length Code
RTR	Remote Transmission Request
CRC	Cyclic redundancy check
ACK	Acknowledgment
LIN	Local Interconnect Network
MOST	Media Oriented Systems Transport
NFC	Near Filed communication
NIST	National Institute of Standards and Technology
FIPS	Federal Information Processing Standard
CMAC	Cipher-based Message Authentication Code
TEC	Transmit Error Counter
REC	Receive Error Counter
MSB	Most Significant Bit(s)
LSB	Least Significant Bit(s)
CS-MA/CA	Carrier Sense Multiple Access / Collision Avoidance
TPMS	Tire Pressure Monitoring System
V2V	Vehicle-to-vehicle
V2X	Vehicle-to-infrastructure
UDS	Unified Diagnostic Services
DTC	Diagnostic Trouble Codes

LIST OF TABLES

AT	Attack Tree
FTA	Fault Tree Analysis
EBIOS	Fault Tree Analysis
TVRA	ETSI Threat, Vulnerability, and implementation Risk Analysis
ETSI	European Telecommunications Standards Institute
ITS	Intelligent Transportation System
HEAVENS	project Healing vulnerabilities to enhance software security and safety
OCTAVE	Operationally Critical Threat, Asset, and Vulnerability Evaluation
TARA	Threat Analysis and Risk Assessment
EASIS	project Electronic architecture and system engineering for integrated safety systems
RFID	Radio Frequency identification
GPS	Global Positioning System
OTA	Over The Air
RDS	Radio Data system
TMC	Traffic Message Channel
AES	Advanced Encryption Standard
CBC	Cipher Block Chaining
MAC	Message Authentication Code
ID	Identifier
GROOVE	Graphs for object-oriented verification
GTS	Graph Transformation System
LHS	Left Hand Side
RHS	Right Hand Side
LTS	Labeled Transition System
TR	Transformation Rule(s)
NAC	Negative Application Condition
EVITA	project E-safety vehicle intrusion protected applications
SW	Software
HW	Hardware
CPU	Central Processing Unit
ISO	International Organization for Standardization
SAE	Society of Automotive Engineers
OBD	On-board Diagnostics
ML	Machine Learning
Acc	Accuracy
IDS	Intrusion Detection System
IDPS	Intrusion Detection and Prevention System
DT	Decision Tree
KNN	K-Nearest Neighbors
NN	Neural Networks
DoS	Denial of Service
OS	Operating System
HSM	Hardware Security Module
PKI	Public Key Infrastructure

CHAPTER 1

Introduction

This chapter gives a general overview of the context and motivation of the thesis. It introduces the primary research goals and contributions that will follow in the upcoming chapters.

Contents

1.1	Context: security of connected vehicles	3
1.2	The automotive industry challenges	5
1.3	Motivation and goals	5
1.4	Contributions	6
1.5	Outline	7

1.1 Context: security of connected vehicles

The complexity of current automobiles is continuously increasing. The high demand for safety and better user-experience by the consumers is pushing carmakers to integrate more technologies into the cars. As a result, vehicles are becoming more connected to the outside world via wireless links such as WIFI, Bluetooth, NFC, We have schematically depicted in Figure 1.1 the connected vehicle environment to show the complexity and plurality of interactions that a modern vehicle can engage in. The introduction of connectivity features to the automotive industry is enabling numerous useful new services such as advanced navigation and infotainment systems, remote safety services (diagnostics, emergency calls, . . .), car sharing, and more These technologies are making the car more aware of its surroundings and are paving the way towards fully autonomous vehicles with the introduction of multiple sensors and perception equipment.

These significant technological advancements that comes with increased connectivity also come with a downside: the car becomes more exposed to cybersecurity attacks and more

1. INTRODUCTION

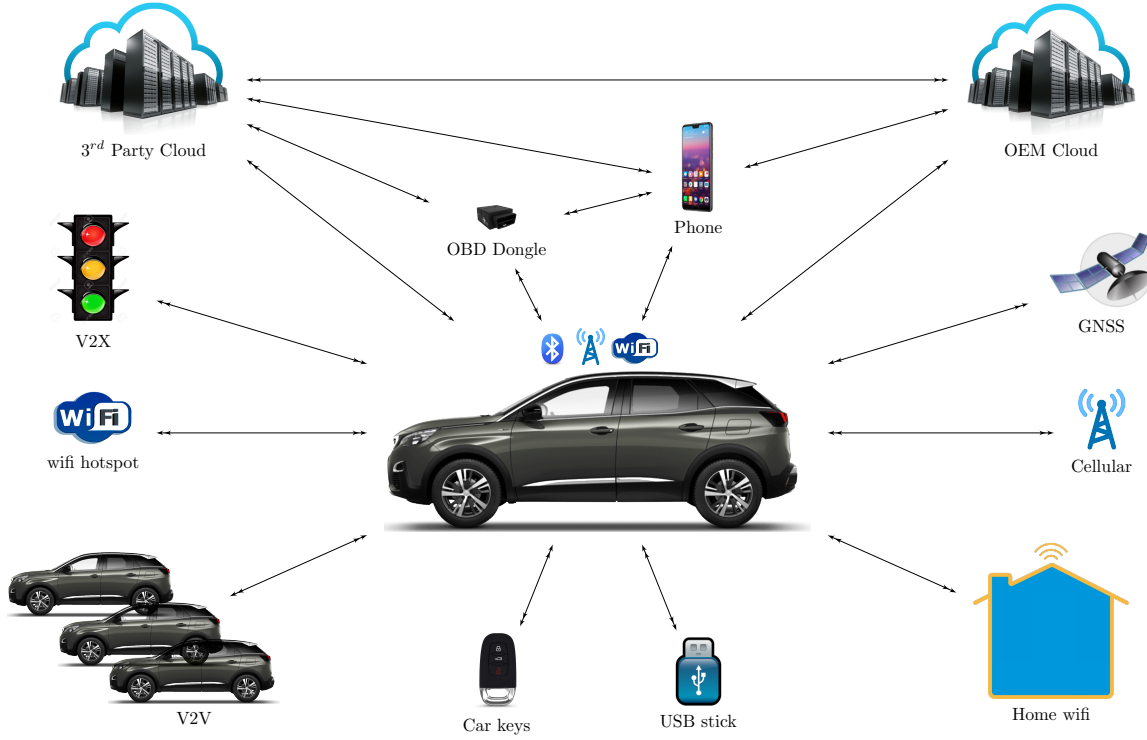


Figure 1.1: Conceptual diagram of the connected vehicle environment

prone to hackers. It induces new risks that may impact the integrity and confidentiality of the data processed by the computers. An investigation conducted in 2015 and ordered by Senator Edward J. Markey (D-Massachusetts) [90] reports that: *Nearly 100% of cars on the market include wireless technologies that could pose vulnerabilities to hacking or privacy intrusions.* In fact, when an industry without experience in security starts connecting equipment to the Internet, it typically makes some mistakes in how to evaluate the need for secure systems and then how to implement them. However, with connected and autonomous automobiles, the stakes for getting security right have never been higher. “What’s the worst that could happen?” is a serious question when we are dealing with a set of computers traveling at 100 km/h, deciding and taking actions on behalf of the “driver”.

Since 2010, the scientific community has continuously been reporting weaknesses and vulnerabilities in vehicles systems and architectures. In Defcon-2015, Charlie Miller and Chris Valasek [95] demonstrated the first end-to-end remote exploitation of an unaltered passenger vehicle. The attack showcased actions that can endanger the passengers’ safety as well as privacy. Following this episode, the targeted car manufacturer issued a recall of approximately 1.4 million cars. Since then, multiple other findings have also been reported targeting other car manufacturers including Tesla [88], BMW [4] and others.

1.2 The automotive industry challenges

The reported attacks showed that there are apparent issues and significant challenges facing the future of the automotive industry with respect to cyber-security. In fact, recall campaigns, although highly costly, are effective for software updates and bug fixing, but are completely useless when the weakness is in the architecture of the ECUs, or the nature of the technology. Clearly, some issues can hardly be solved after vehicle deployment. This is why adopting and integrating security from design approaches is an important challenge towards more secure and safer cars.

A clear understanding of the threats and the risk should be part of the vehicle development process from the early design phase. The latter assumes that there should be tools and methodologies that can assist in risk analysis studies and help to explore possible weaknesses and guide design decisions. Ultimately, these methodologies should help define security concepts, from which can be derived and defined necessary security mechanisms and countermeasures to be deployed.

Security countermeasures are employed ideally to prevent a security attack from taking place or to minimizing and recording the effect of such an attack if prevention cannot be accomplished. In general, no single countermeasure can succeed in thwarting all attacks, so the principle of *defence-in-depth* is often advocated and adopted, where a number of layers of security, using different strategies, are employed. Nevertheless, classical countermeasures are not always adapted to the automotive application domain. In fact, there are unique constraints related to the nature of the car environment such as limited computational and storage capabilities, as well as limited communication throughput, and safety constraints. Thus, the challenge is to adopt novel and cost-effective protection mechanisms that are compliant and compatible with industry standards and needs.

1.3 Motivation and goals

To date, there is no dedicated methodology for systematic threat analysis and risk assessment that is fully adapted to the automotive domain. Although, there are some running trends. One of the trends is to systematically make threat analysis and risk assessment studies using methodologies like EBIOS [3], TVRA [39] or EVITA methodology [2]. However, these methodologies might not be mature enough to cope with the automotive industry requirements. Another trend is to use methodologies that combine safety and security issues. In this context, the ‘*Society of Automotive Engineers* (SAE) issued in 2016 the “*Cybersecurity Guidebook for Cyber-Physical Vehicle Systems*” SAE-J3061. This guidebook, which should be further developed and refined into a full standard, includes recommended practices of how to build a cybersecurity process framework that covers the entire product life-cycle and is designed to facilitate coordination with safety processes.

Thus, the first motivation of this thesis is to propose a formal model of the security that aims, in a first place to validate the attacks in the specific context of the automotive industry, and in a second place to study the resilience of the vehicle facing attack attempts and different

1. INTRODUCTION

opponent strategies. This formal modeling will allow defining the bases of secure design.

A second motivation of the thesis is to define security protection and prevention mechanisms that can be deployed inside the vehicle in order to thwart potential threats. Indeed, in view of the plurality of attack vectors of the connected car, and in view of the high safety standards that the vehicle should guarantee, it becomes essential to study, in this context, the best possible security barriers that can confine possible attacks and prevent them from spreading from one component to the others inside the vehicle. These security mechanisms can give the vehicle resilience capabilities.

1.4 Contributions

Throughout the thesis, three main contributions are presented and detailed. All of which are dedicated to protect against potential threats and to reduce the gap towards more secure and safer vehicles.

1.4.1 Formal modeling approach for automatic attack tree generation

During the design phase of a vehicle, this thesis proposes a formal method to assist in risk assessment step.

- Formal model for the system: A framework for the formal modeling of the system is defined and detailed. This framework allows defining the cyber-physical architecture of a vehicle formally. It defines software, hardware, data and network architectural components as well as the relations between them. It also defines an attacker model as a set of a knowledge database and a set of capabilities in the form of basic attacks.
- Automated attack tree generation: Once defined, the cyber-physical architectural model of a vehicle along with an attacker model is used to build attack trees for a pre-defined attacker goal automatically. These attack trees, allow to better account for the attack likelihood during the risk assessment step.

1.4.2 CAN identifier randomization strategy

In order to protect against an attacker model that has direct physical access to the CAN-bus, we propose in this thesis a protection mechanism that uses an identifier randomization strategy.

- A general formalism of identifier based protections: The general formalism of identifier protection family of protection solutions is established. Explicit constraints are explained and formulated in order to build adequate randomization strategies to protect against reverse engineering of the manufacturer-specific communication protocol used on top of CAN as well as against injection and replay attacks. Evaluation metrics against target attacks are also proposed based on information theory.
- Optimal randomization strategy: Depending on the constraints, multiple randomization strategies are proposed and experimentally tested and compared with state-of-the-art mechanisms and evaluated. An optimal randomization strategy is formally derived and proved, and is used to build more advanced randomization strategy.

1.4.3 Prediction-based intrusion detection system

In order to protect against an attacker model that has indirect and remote access to in-vehicle communication buses through the control over one of the legitimate Electronic Control Units (ECUs)¹, we propose in this thesis an intrusion detection mechanism that uses machine learning techniques that protects against payload injection attacks.

- Formal framework of the detection principle: First, we propose a formal framework of the detection principle using supervised machine learning techniques and outlier detection. The detection principle depends on the type of monitored signal and exploits solely other sensor values/states.
- Training and testing for different machine learning algorithm: Second, we propose and demonstrate how to build and evaluate the detection rules. Use-cases are considered and experimentally validated on multiple driving behaviors.
- Evaluation against attacks: Third, we propose also to evaluate the robustness of the detection rules against attacks. We show how to simulate attacks and report experimental results for example attacks.
- Alert handling: An alert handling strategy is developed. It allows determining, in case of violation, how to handle different situations and how to reduce the error probability in order to provide an appropriate response.

1.5 Outline

This thesis is composed of six chapters. Following the introduction, Chapter 2 review the literature in order, first, to give a clear view of vehicular architecture and involved technologies, then it introduces their weaknesses and the threats that they are exposed to, and finally, it presents available methodologies and countermeasures for securing these systems. A substantial part of this chapter is also dedicated to categories of protection mechanisms for in-vehicle communication buses with a special emphasis on the Controller Area Network. Chapter 3 proposes a methodology of automatic Attack Tree generation based on an attacker and a vehicle architecture formal models. The presented method is designed to support threat analysis and risk assessment study, more precisely to better evaluate the attack likelihood for a better risk assessment. This chapter presents the first contribution of this thesis. Following this contribution, an important observation is made about the importance of securing in-vehicle communication buses. As a consequence, a special focus is devoted to developing innovative countermeasures. In Chapter 4, a first protection mechanism is developed for the Controller Area Network. This protection mechanism is based on an optimal randomization strategy designed to raise the security level against *reverse-engineering* and *injection/replay* type of attacks. This chapter constitutes the second contribution of this thesis. In Chapter 5, the third and final contribution of this thesis is presented. It exposes a novel in-vehicle intrusion detection framework based on supervised machine learning techniques. The approach is also

¹Electronic Control Units are one of the components that constitute the cyber-physical architecture of modern vehicles and will be introduced more in details in chapter 2

1. INTRODUCTION

evaluated on a real-world case-study. Finally Chapter 6 concludes the thesis with a summary of the presented contributions and a discussion on their applicability, benefits, and drawbacks.

CHAPTER 2

State of the Art

This chapter covers the state of the art from different angles addressed in this thesis. It gives background information on the cyber-physical architecture of modern vehicles. It presents a survey of threats and attacks as well as methodologies to approach security in cars.

Contents

2.1	Introduction	9
2.2	Cyber-physical architecture of connected cars	11
2.3	Vulnerabilities and threats survey	14
2.4	Countermeasures and Security methodologies	18
2.5	In-Vehicle Secure Communication survey	25
2.6	Conclusion	42

2.1 Introduction

Modern automobiles are no longer just mechanical structures with battery-powered electric components like alternator and carburetor. They are much complex than that as they integrate multiple advanced functionalities. *Even low-end cars now have 30 to 50 ECUs¹ embedded in the body, doors, dash, roof, trunk, seats, and just about anywhere else the car's designers can think to put them. That means that most new cars are executing tens of millions of lines of software code, controlling everything from your brakes to the volume of your radio [26].* Figure 2.1 gives an overview of the plurality of functionalities that a modern vehicle can offer and that are implemented throughout multiple embedded systems. Needless to say that high-end automotive systems incorporate more embedded systems and communication interfaces and thus are much more complex.

¹ECUs stands for Electronic Control Units and are defined in section 2.2.3

2. STATE OF THE ART

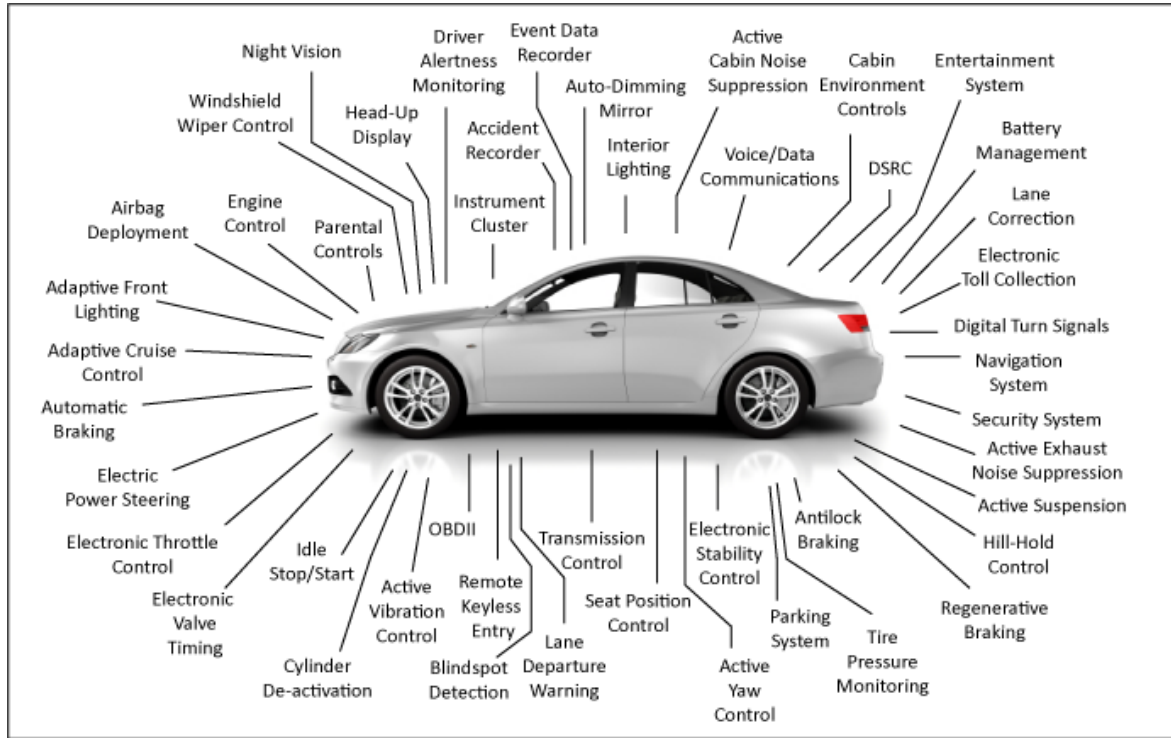


Figure 2.1: Functionalities and services of Modern cars [1]

As a matter of fact, embedded hardware/software systems control innovative functions in cars, support and assist the driver and realize new features for information and entertainment [24]. They constitute the backbone of the high level of safety requirements that the vehicle must respond to as well as user comfort and connectivity with the outside world. This involves the use of advanced technologies based on a computing infrastructure composed of numerous electronic components –named Electronic Control Units (ECUs)– embedded inside the vehicle along with advanced sensors and actuators. In order to coordinate their decision-making process and actions, the embedded systems need to communicate with each other, as well as with the outside world, forming an embedded cyber-physical architecture of the vehicle. However, like any other network, it is not excluded that vehicle embedded software may contain vulnerabilities and that the architecture may contain design flaws that an attacker can exploit.

Until recently, the cyber-physical automotive systems could be considered as a closed and isolated. This assumption reduces significantly the risk of a vulnerability being exploited by a hypothetical attacker. Nevertheless, with the integration of connectivity to the outside world, the assumption is no longer valid, and risk assessment needs to be re-examined. This brings a wide spectrum of challenges for the automotive industry. In the following we introduce in section 2.2 the main components and functions of the cyber-physical architecture of modern vehicles. Next, in section 2.3 we survey the principal threats and review the literature on

the main attack vectors for the connected vehicle. In section 2.4 we give main proposed approaches to deal with these challenges from design step and finally section 2.5 proposes a survey on solutions to secure in-vehicle communications.

2.2 Cyber-physical architecture of connected cars

The cyber-physical architecture of the car is composed of multiple components that could be categorized into four main categories: *sensors*, *actuators* and *Electronic Control Units (ECUs)* all communicating over shared *communication buses*.

2.2.1 Sensors

The set of sensors that we can find inside a modern car have multiple functions. These functions can be categorized into two main categories or roles. The first role of the sensors is to report information about the state of the vehicle. Example: the vehicle speed, vehicle acceleration, state of the vehicle doors (open/closed), engine oil temperature The second role is to report information about the vehicle environment and surroundings. Example: vision radars, ultrasonic sensors and cameras, The data produced by the sensors are sent to Electronic Control Units (ECUs) to be processed in order to produce commands for the actuators.

2.2.2 Actuators

Actuators play an essential role inside the vehicle. They are the components that transform commands into actions necessary in order to the vehicle to accomplish its main function. In general, an actuator requires a control signal and a source of energy. Inside the vehicle, these controls signals (or commands) are sent by Electronic Control Units (ECUs). Examples of actuators in the car include the lights, engine, displays, wheel orientation system,

2.2.3 Electronic Control Units

Electronic Control Units are the most important components of the automotive architecture. They are in charge of processing sensed data through embedded sensors and outside data sources, and transforming them into commands for the actuators. In general, they are composed of hardware electronic components (memories, micro-controllers, . . .) that have a processing capacity, and that embed algorithms (software) needed to ensure the control of every single functionality inside the vehicle. These functionalities include safety functions like antic breaking, lane-keep-assist to more advanced functionalities that ensure the user comfort air-conditioning as well as the navigation system, internet connectivity Another important role for ECUs is to orchestrate communication with each other and the outside world through different types of communication interfaces.

2. STATE OF THE ART

2.2.4 Communication interfaces

2.2.4.1 In-vehicle shared communication buses

Communication buses in the automotive domain were introduced as soon as the number of ECUs embedded in the vehicle has reached a certain level of complexity that made a point-to-point communication approach no longer viable, and impossible to implement and maintain. At that point, the car as a system on its own was isolated from its external world. The choice of communication buses was not motivated by information security, but rather by safety and robustness issues.

As for today, in-vehicle shared communication buses represent the backbone of the architecture and the main medium of communication between the ECUs. Multiple technologies of buses could be found in today's cars. Examples include the Controller Area Network (CAN), Local Interconnect Network (LIN), FlexRay, Media Oriented Systems Transport (MOST) and Ethernet. Each technology has some characteristics that justify its presence between specific ECUs: robustness, implementation cost, throughput... However, the Controller Area Network (CAN) imposed itself as the de-facto communication bus for the automotive applications, and almost all automotive manufacturers are implementing the CAN bus in their cars.

Table 2.1: In-vehicle communication bus technologies

Technology	Standard	Data Rate	Application example
LIN	ISO 9141	$\leq 19.2Kbps$	Sensor and actuator control.
CAN	ISO 11898/11519	$\leq 1Mbps$	Real-time communication: engine management.
FlexRay	ISO 17458	$\leq 10Mbps(\times 2)$	Safety critical: steer-by-wire and brake-by-wire systems.
MOST	ISO 7498-1	$150Mbps$	Multimedia content, navigation.
Ethernet	IEEE 802.3	$100Mbps^1$	Advanced safety features: 360-degree surround view parking assistance, rear-view cameras.

2.2.4.2 Diagnostics interface

The diagnostics interface (also known as On-board diagnostics (OBD) interface) is a standard interface that gives access to the status of the various vehicle subsystems. It is designed in order to be used by the repair technician, the car owner or any other entity that may be interested in diagnostics information and that provides real-time data in addition to a standardized series of diagnostic trouble codes (DTCs), which allow one to identify malfunctions in vehicle components rapidly.

In order for this diagnostic to be possible, ECUs and diagnostics equipment implement the Unified Diagnostic Services (UDS) communication protocol (specified in ISO 14229-1). This protocol allows a request-response type of interrogation to the ECUs and implements multiple services. These services include Diagnostics session, Programming Session, ECU reset, firmware updates, memory read, memory write, Some of those sessions require a *Security Access*, although as we will see in the next section, it has been proven that it is not well secured.

Technically, the diagnostics interface is designed to communicate with the outside world (a diagnostic device) and gives access to the in-vehicle communication buses at the same time. Depending on the architecture, the diagnostics interface can be directly connected to communication buses, but can sometimes be connected to a *Gateway* ECU that handles diagnostics communication (requests and responses) and re-routes them to appropriate sub-networks. The latter design paradigm has been proven to be more efficient from a security point of view, although it could induce more implementation cost.

2.2.4.3 Communication with the outside world

These communication interfaces are relatively new the automotive industry. They have been introduced in order to enhance the safety and autonomy of the vehicle but also to make the car more comfortable and convenient as well. They encompass all technologies that are related to infotainment systems like Bluetooth, wifi, cellular . . . , but also safety related like Vehicle-to-vehicle (V2V) communication, and vehicle-to-infrastructure (V2X) communications. Generally speaking, such communication technologies can be implemented in multiple ECUs inside the vehicle. Nevertheless, multiple communication interfaces are sometimes grouped in one or two ECUs (This is, in particular, the case for the Head-unit, Telematics-units or infotainment system for some architectures) depending on the architecture.

Besides their importance with respect to road-safety applications, these communication interfaces (in particular cellular and internet connectivity) enable multiple useful services like remote diagnostics, over-the-air updates, car-sharing,

2.2.5 Overall architecture

The cyber-physical architecture of the vehicle is composed of the previously mentioned components. Electronic control units constitute the main part. They implement sensors and communication interfaces and share in-vehicle communication buses that are used to transfer information and data from one unit to the others. Figure 2.2 gives an example a vehicle cyber-physical architecture¹.

2.2.6 Aftermarket and diagnostics Devices

Aftermarket devices are equipment that can be added to the vehicle but are not part of the car manufacturer official architecture. Examples of such aftermarket devices include

¹Figure 2.2 is a sketch of the architecture of the Jeep Cherokee vehicle produced by Fiat-Chrysler and demonstrated vulnerable by Miller et al. [95]

2. STATE OF THE ART

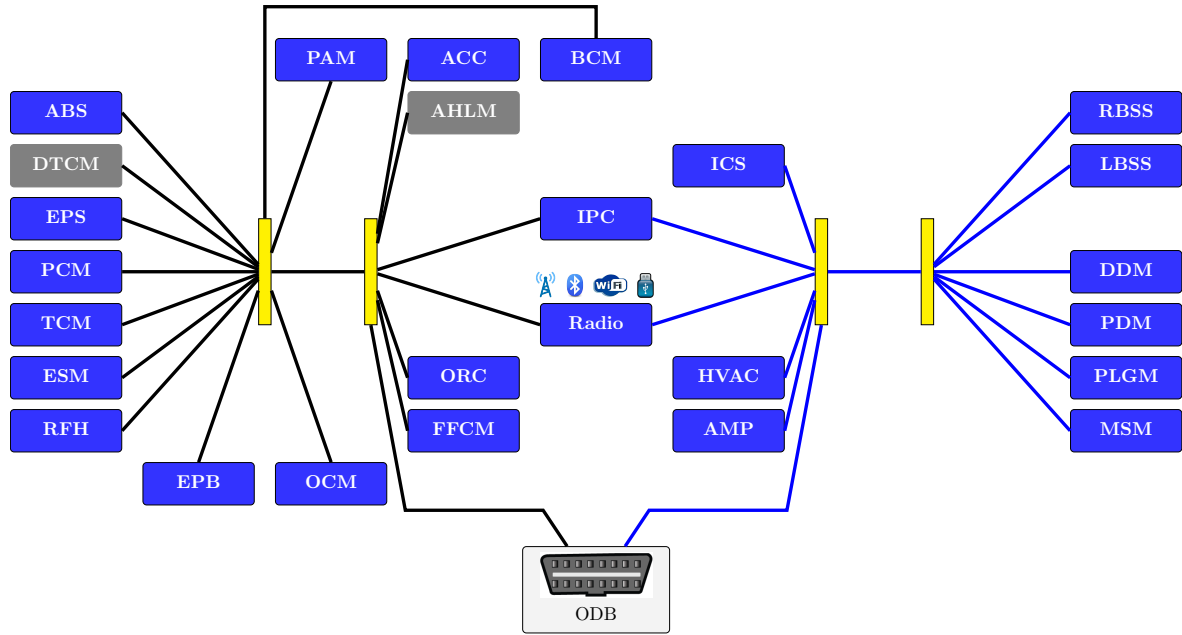


Figure 2.2: Example of cyber-physical architecture [95]

insurance dongles produced by insurance companies to keep track of vehicle-related and driver related information. These insurance dongles are generally also equipped with long-range (cellular) connection and can be remotely managed by the insurance companies. Additionally, examples include diagnostic devices designed to have short-range wireless (Wifi, Bluetooth) connection to the user smartphone in order to report some diagnostics information on user request.

In general, these devices are designed to connect to the vehicle via the diagnostic port and use diagnostic commands. They nevertheless have access to information communicated over in-vehicle communication networks as the OBD port is sometimes wired directly to in-vehicle communication networks.

2.3 Vulnerabilities and threats survey

2.3.1 Vulnerabilities and attack vectors

Given the main architectural elements presented in the previous section, an attacker can leverage multiple entry points of the vehicle in order to reach his/her goal. In fact, in the last years, researchers began to report multiple important issues related to the design and implementation of different architectures. These entry points can be categorized according to the proximity of the attacker into three main access types: Direct physical access, Indirect physical access, and remote access. In what follows we analyze the access vectors of today's cars from the attacker point of view supported by examples of attack principles reported in research work whenever suitable.

2.3.1.1 Direct physical access

The first attack vector that the adversary might be tempted to use is the direct physical access to the car architectural components. This includes direct physical access to communication buses as well as to electronic control units.

An early work of Hoppe et al. [54] pointed out the importance of the threats of direct access to in-vehicle bus networks, namely frame injections on the CAN bus were carried out on a simulated bench. In particular, they demonstrated that this access could be leveraged to target the electric window lift, warning lights, and airbag control system.

These observations have been further confirmed later by Koscher et al. [79] and Checkoway et al. [28] that performed CAN frame injections on a real vehicle with direct physical access to the communication bus of the car. In fact, since during design phase, in-vehicle communication buses were assumed to be “trusted”, an adversary directly connected to them can have read and write access. This gives the attacker the possibility of impersonating any sensor ECU, to issue commands in order to control actuators and receive all information that is being transmitted on the communication medium. The principle of these attacks will be further detailed in section 2.5 and possible countermeasures will be discussed.

Additionally, direct physical access to the OBD-port, in general, can be leveraged to launch other types of attacks that use diagnostics commands. In fact, diagnostics protocols are implemented inside almost all ECUs. They support multiple functions including re-configuration, re-programming, reset, log recovering, actuators tests However, most critical functionalities are only accessible through authenticated sessions (namely Security Access). An attacker leveraging direct physical access to the in-vehicle communication bus can use diagnostic sessions in order to activate one of these functionalities. In [94, 95] Miller et al. used diagnostics session in order to activate the brakes. In practice, an attacker may also be tempted to by-pass or break authentication mechanism in order to gain access to more advanced functionalities like ECU reprogramming and re-configuration and even reset which can enable more advanced scenarios.

Moreover, gaining direct access to in-vehicle networks constitutes a privacy threat, as the attacker can read all the information communicated on the network. In this context, there are some tools that are designed to help the attacker in the procedure of reverse-engineering protocols of in-vehicle networks [14, 118]. Besides, even user-related information can be leaked via in-vehicle networks. In [91, 92] Martinelli et al. explain how it is possible to derive the driver profile directly from data collected on in-vehicle communication networks.

In the same category of attacks, direct physical access to embedded systems (ECUs electronic boards) can reveal much information and allows the attacker to manipulate lots of functionalities in the car. Examples of such attacks include memory dump attacks that allow the attacker to recover software for reverse-engineering purposes. It also allows learning configuration parameters which constitute a privacy violation from the automotive manufacturer point of view. Access to external memory components of the ECUs can also grant the attacker write access that can be exploited in order to modify the ECU software. These

2. STATE OF THE ART

types of attacks are especially used for car tuning. Car tuning community are seen as hackers from the automotive manufacturer point of view. Their primary objective is to augment the capacity of the car by modifying specific engine parameters as well as activating some non-free functionalities in the car.

2.3.1.2 Indirect physical access

Direct physical access to the internal communication buses or ECUs of the vehicle may seem a strong assumption as an attacker model. In fact, from the car manufacturer point of view for whom the primary concern is the safety of the passengers, an attack leveraging direct physical access to in-vehicle communications might not represent a high risk due to the limited window of opportunity and the limited accessibility to the specific sub-networks. However, an attacker might have access to a device that has access to the vehicle equipment. Such an attack vector is known as the indirect-physical access to the car. In general, these devices would have to carry an attack payload that ultimately would exploit a vulnerability in an ECU connected to in-vehicle networks.

In [28] Checkoway et al. explain and demonstrate how they were able to send arbitrary CAN messages on the CAN bus using a carefully crafted version of an audio file stored on a CD and played on the CD player of a car. Their attack leveraged a code vulnerability in the WMA parser of the media player. In [42] Foster et al. analyze an aftermarket dongle that connects to the vehicle via the On-Board-Diagnostics (OBD) port. They prove that an attacker can use this category of devices to reach the communication buses of the vehicle. The same attack principle was already explained in [28] using a diagnostic “PassThru” device that connects to the car also via the OBD port.

2.3.1.3 Wireless access

Besides the physical access to the vehicle, the attacker can also leverage wireless access vectors. As a consequence of the introduction of remote connectivity to the car, additional attack vectors are also added. Reachability of these attack vectors depends on the technology used. In general, we can split them into two main categories: short-range wireless vectors and long-range wireless vectors.

Short-range wireless access: A short-range wireless attack vector is a wireless communication access that has a *limited* reachability range. Examples of these communication technologies that we find in today’s cars include: Wifi, Bluetooth, RFID ... These communication channels constitute an entry point for the attacker. In fact, Ishtiaq et al. [58] show that in the case of Tire Pressure Monitoring System (TPMS) the wireless access vector is not well secured which poses privacy issues, besides they were able to perform message spoofing to trigger system warnings. In [95] Miller et al. analyze the Bluetooth and wifi interfaces of a car and expose how and why it is practically possible to use those interfaces to take Control of the car.

Long-range wireless access: The car attack surface also includes long-range access vectors. These attack vectors are more or less security sensitive depending on whether they are

addressable or not and how the received data on those channels is being processed. Examples of non-addressable attack vectors include Global Positioning System (GPS), Satellite Radio, Digital Radio, Radio Data system (RDS), Traffic Message Channel (TMC) ... Besides of the non-addressable attack vectors, the car is also equipped with long-range addressable wireless channels. An example is the cellular network which is an important feature of today's and future connected cars that supports a broad range of applications that are safety-sensitive (diagnostics, crash reporting, Over The Air software updates ...) and essential for user comfort (navigation system, internet access, ...). These long-range wireless technologies open the vehicular architecture as well and constitute long-range attack vectors. It has been practically demonstrated in [95] that it is possible to use these attack vectors to gain control of a vehicle. In general, the recent works performed in this area have shown that combinations of exploits and misconfigurations are the typical means by which an attacker breaks into a car communication bus.

2.3.2 Threats

2.3.2.1 Security violations

The exploitation of the previously mentioned vulnerabilities and flaws correspond to security violation from the car manufacturer point of view. We can thus classify the reported attacks into three main categories according to the violation of a security principle.

- *Privacy/Confidentiality violation:* This type of violation includes violation of the driver privacy, for instance by intruding on the user profile private data stored in the infotainment system, by identifying the driver, tracking the vehicle, by listening to conversations inside the vehicle leveraging access to the microphone, Also intrusion on the privacy and confidentiality of certain manufacturer specific information. For instance reverse-engineering protocols, firmware, and intellectual property violation.
- *Integrity violation:* This type of violation include attacks that make modifications to the vehicle configuration for instance by modifying the engine performance (ECU tuning), by adding functionalities to the vehicle or by tempering with control information (sensors and actuators) in order to influence the behavior of the car from the inside or the outside.
- *Availability violation:* This type of violation include attacks that target the availability of certain services like communication between ECUs, communication with the outside world or even some safety services

2.3.2.2 Attacker motivation

The different reported attacks exploit multiple vulnerabilities and design flaws. These flaws can be exploited for multiple purposes and to reach a number of different goals depending on the attacker motivation. In general, we can categorize these attacks, with regards to the attacker motivation, in the following way:

- *Financial gain:*
In this category, the attacker's motivation is to gain money from the attack. This

2. STATE OF THE ART

category includes flaws that would allow a car theft, flaws that would allow augmenting the value of a car for instance by reducing the odometer value,...

- Information gain:

In this category, the attacker's motivation is to gain information either about the car or about the owner/driver of the car or even about the car manufacturer regarding intellectual property.

- Functionality gain:

In this category, the attacker's motivation is to add some functionalities that are not part of the initial design of the car. This category includes, for instance, adding activating services for free, augmenting the capabilities of the car through the modification of performance parameters (also known as ECU tuning, or car tuning)

- Harm/control:

In this category, the attacker's motivation is to harm the safety of the driver for instance by tampering with the normal functioning of the car internally or externally.

2.4 Countermeasures and Security methodologies

All of the attacks described in the previous section are made possible due to flaws and vulnerabilities introduced during different steps of the design of the vehicle. Some of them could be avoided only through the consideration of security issues at an early design phase and adopting *security-by-design* approaches. Others arise later during product development and implementation. Finally, some others could not be avoided and need run-time monitoring capabilities in order to protect against them. In this what follows we report the main countermeasures to be integrated into different steps of the vehicle production lifecycle.

2.4.1 Countermeasures

During the design phase of the vehicle, multiple decisions regarding services to be implemented, the ECUs that will hold those services, the expected architecture of the overall system, are made. Additionally, multiple parameters regarding hardware capabilities are fixed. Once these choices are made and parameters defined, they can not be changed during the vehicle lifetime. Some of them are important from a security point of view.

2.4.1.1 Architecture

The network architecture of in-vehicle communication buses and ECUs connected to them can significantly influence the capability of an attacker to reach specific components. In [128] Miller et al. present a ranking of the easiest "attackable" architecture amongst a list of candidate architectures. Take as an example the architecture proposed by the EASIS¹ project presented in figure 2.3 and compare it with the architecture introduced in the previous section (figure 2.2). The architecture of figure 2.3 takes into consideration isolation of domains with different criticality levels (Powertrain, Chassis-and-safety, Body electronic module) from

¹Electronic architecture and system engineering for integrated safety systems

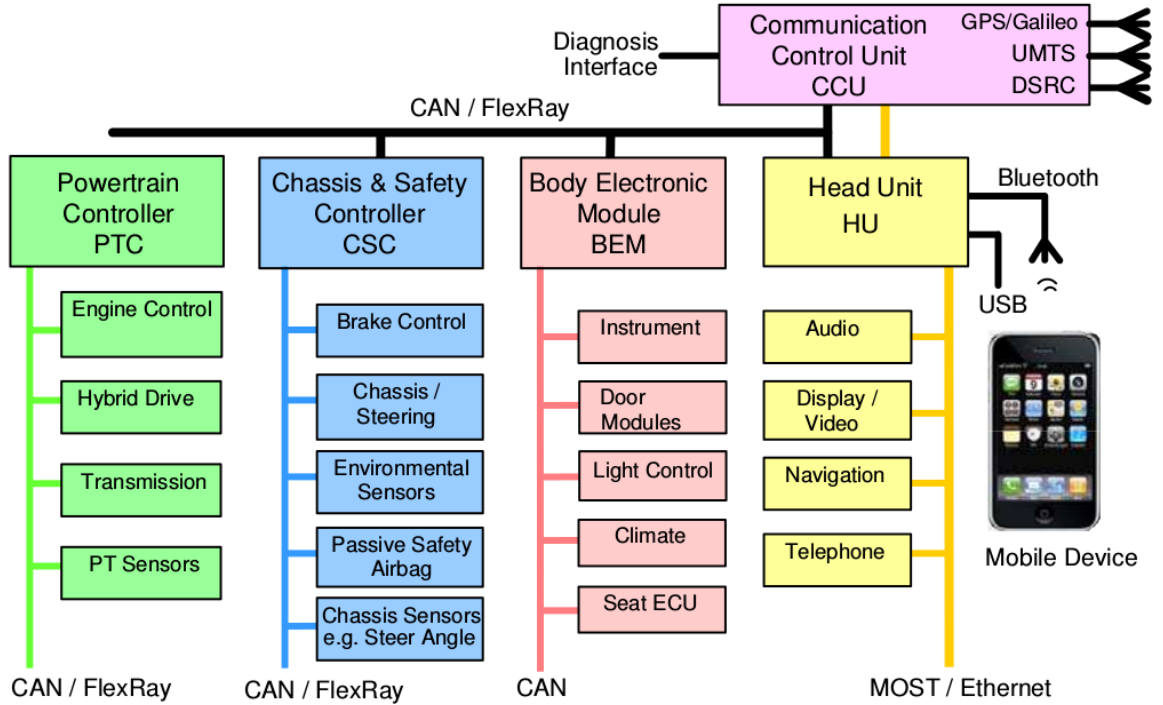


Figure 2.3: The EASIS project proposal architecture

connectivity to the outside world (communication control unit including radio, cellular, car2X and diagnostics interface) and infotainment system (head unit include connectivity to user-devices). On the other hand, the architecture of figure 2.2 connects the Radio ECU (which is connected to the outside world via cellular, wifi and Bluetooth) to both CAN sub-networks. It is easier for an attacker to reach safety-critical functions from remote entry point (and even from direct physical access to the OBD port) in the architecture of figure 2.2 than in the architecture of figure 2.3. In chapter 3, the developed formal model takes as input this architecture and use it to depict possible attack paths. Ultimately it intends to study the impact of architectural changes during the design phase amongst other aspects.

Similarly, from a software point of view, the separation between software components can influence the security of the system. This is, in particular, the case for ECUs that should accommodate applications from different domains (safety-critical, connectivity, diagnostics, ...). In this context, as part of the OVERSEE¹ project [9], Groll et al.[46] propose an application isolation solution based on virtualization where applications of different levels of criticality, run in guest Operating System (OS) above a hypervisor responsible for securing communications between different guest systems.

Additionally, access rights of software components to ECUs assets whether they are basic hardware modules such as CPU, memory, and peripherals or data assets have an impact on privacy and confidentiality of assets. As an attacker that exploits a vulnerability in a software

¹Open Vehicular Secure Platform project [9]

2. STATE OF THE ART

component have the same access rights as that software component, it is sometimes required that software components or applications are granted minimum access rights. These access rights can be subject to the context and state of the vehicle. In chapter 3 we develop a formal model that can help to define these access rights based on a risk minimization approach.

2.4.1.2 Asset protection and data security

Asset protection is a set of preventive techniques generally based on cryptographic primitives whose goal is to protect confidentiality and integrity and authenticity of data assets, software assets, and data flows. Depending on the attacker model and threat, some of these protection mechanisms can be more relevant than others. Previous works have already identified the main security functions and components to be deployed depending on the security need. Wolf et al. [131] identify core security technologies and relevant security mechanisms for critical vehicular applications Kleberger et al. [74] focus in particular on in-vehicle network. Studnia et al. [123] presented the main threats, challenges and security tendencies in deploying security components in automotive systems. The EVITA project[2] have also identified main security needs and mechanisms based on which three Hardware Security Modules (HSM) components have been defined to be integrated in ECUs with different security needs. In this section we highlights the main tendencies.

First, in order to protect ECUs from executing attack payloads, the integrity of the executed firmware should be guaranteed. The validation of ECU code is generally done via firmware integrity checks, secure boot mechanisms [19, 96]. Protecting the confidentiality of the ECU's firmware for intellectual property reasons is also necessary [30, 84]. Additionally, secure software update procedures of ECUs (over UDS or over-the-air), preventing chip tuning, preventing the unauthorized change of the mileage, or assembling non-original parts are necessary to ensure firmware integrity during the vehicle lifetime.

Second, protection of data storage and data flows is necessary in order to thwart privacy violations. Generally symmetric and asymmetric cryptographic techniques for data encryption (like AES¹, ECC²) are used. Encryption keys have also to be secured, generally in secure storage techniques like tamper-proof memories. However, one constraint is the management of cryptographic keys. The systems have to implement protocols to distribute and generate keys. In particular, when deploying certificates, the deployment of public key infrastructure (PKI) that can distribute and manage keys is a must.

In-vehicle security mechanisms are presented more in details in section 2.5. In chapter 4, in order to protect the CAN-bus from an attacker with direct physical access, we investigate obfuscation strategies over the identifier field of the CAN frame.

2.4.1.3 Policy enforcement and run-time protections

In addition to the above-mentioned protection mechanisms, which define a security policy, there are also other mechanisms whose role is to reinforce this policy during run-time.

¹Advanced Encryption Standard

²Elliptic Curve Cryptography

At the software level, well-known solutions like intrusion detection systems based on functions-calls [130], system-calls [53], and Control-Flow-Graph and Control-Flow-Integrity [10] can be deployed. Schweppe al. [115] and Bouard et al. [23] proposed a security framework with taint tracking tools that allows to dynamically monitor data flows within and between ECUs to enforce security and privacy of data assets.

At the network architecture level, network firewalls that can monitor legitimate frames in sub-networks, and routing frames between them. Network intrusion detection systems also can monitor the behavior of specific nodes and frames. These techniques are presented more in details in section 2.5. In chapter 5, in order to monitor the behavior of the CAN network frames exchanged between multiple network nodes, we propose a prediction-based intrusion detection system trained with machine learning techniques.

2.4.2 Threat Analysis and Risk Assessment

To be truly effective, the security mechanisms in cars must be specified with an overall vision of the system and at an early design phase. For this purpose, the *defense-in-depth* strategy is generally advocated. It consists of multiplying security mechanisms to protect the overall system. In practice, this means deploying security mechanisms (complementary and/or redundant) at several levels of the system, instead of merely protecting its entry points. This implies a step during the design phase, where security engineers come-up with different ways the system could be compromised in order to build strategies to protect against intrusions namely a Threat Assessment and Risk Analysis step (TARA).

In this context, one of the first initiatives to integrate the TARA step into the design phase of the vehicle was proposed by the SAE International¹ that published in 2016 the “*Cybersecurity Guidebook for Cyber-Physical Automotive Systems*” SAEJ3061 [32]. Additionally, an on-going effort is put into a new standard ISO/SAE-21434 [7] that is expected to become the major standard for cybersecurity engineering in the automotive domain. A final version is presumed to be published by early 2020. Meanwhile, the SAEJ3061 guidebook provides high-level guidance, best practices, tools and methods to integrate cybersecurity into the existing development process of an organization. More precisely, it divides the vehicle life-cycle into *concept-phase*, *product development*, *production*, *operation* and *service*. In the *concept-phase* the goal is to define cybersecurity goals and strategy that ultimately should be translated into detailed technical requirements to be pushed into *product development*. The proposed steps to be followed during the *concept phase* (see figure 2.4) are centered around the main step that is the “Threat analysis and risk assessment”. The goal of a TARA step is to identify threats, estimate their potential and based on the risk they present formulate security requirements to be implemented.

Multiple TARA methods and techniques have been proposed and represent the state-of-the-practice in the automotive industry as part of the decision basis for elaborating cybersecurity requirements. Examples include methods like:

¹Society of Automotive Engineers

2. STATE OF THE ART

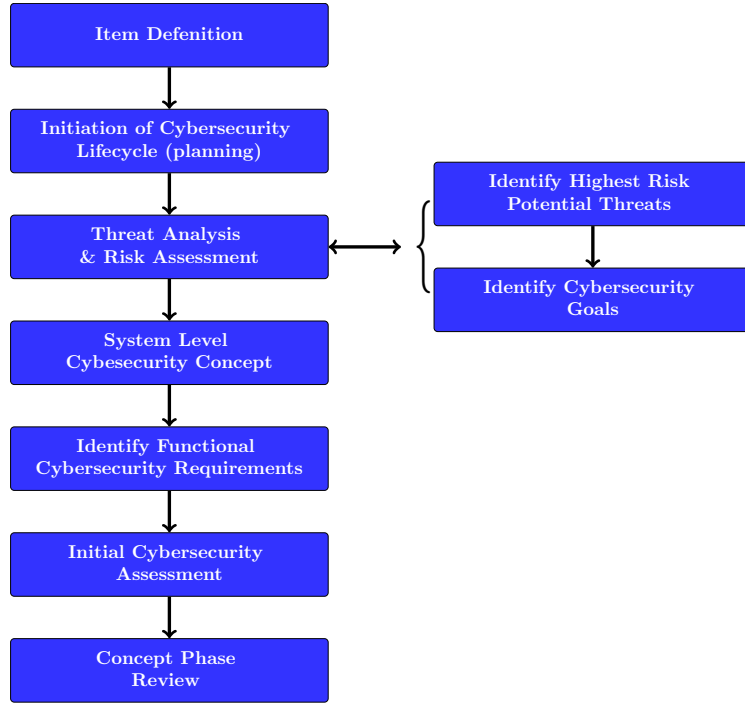


Figure 2.4: SAE-j3061 concept phase steps [32]

- ETSI Threat, Vulnerability, and implementation Risk Analysis (TVRA) standard [39],
- EBIOS *Expression des Besoins et Identification des Objectifs de Sécurité* [3],
- Operationally Critical Threat, Asset, and Vulnerability Evaluation (OCTAVE) [13],
- E-Safety Vehicle Intrusion Protected Applications (EVITA) project [2],
- HEAVENS security model [6],
- Attack Trees [114].
- ...

The EVITA approach: It has been developed in the context of a European project. In what follows we present the EVITA methodology as reported by Henniger et al. [52], as being the only approach that was developed precisely for risk analysis in the automotive domain. In this approach, the goal is to identify the most relevant security requirements based on the risk posed by potential attacks. The risk here is defined as a function of the *severity* of the attack from the stakeholders' point of view and the estimated *probability of success* of an attack. Some threat may impact the safety of the vehicle; in this case, the risk assessment includes an additional *controlability* parameter.

1. *Severity:*

The EVITA risk assessment approach considers the severity of a threat in four different

Table 2.2: Severity rating of threats in the EVITA approach

Severity Class	Aspect of security threats			
	Safety	Privacy	Financial	Operational
0	No injuries	No unauthorized access to data	No financial loss	No impact on operational performance
1	Light or moderate injuries	Anonymous data only (no specific driver of vehicle data)	Low-level loss (\approx 10 euros)	Impact not discernible to driver
2	Severe injuries (survival probable); light/moderate injuries for multiple vehicles	Identification of vehicle or driver; anonymous data for multiple vehicles	Moderate loss (\approx 100 euros); low losses for multiple vehicles	Driver aware of performance degradation; indiscernible impacts for multiple vehicles
3	Life threatening (survival uncertain) or fatal injuries; severe injuries for multiple vehicles	Driver or vehicle tracking; identification of driver or vehicle for multiple vehicles	Heavy loss (\approx 1000 euros); moderate losses for multiple vehicles	Significant impact on performance; noticeable impact for multiple vehicles
4	Life threatening or fatal injuries for multiple vehicles	Driver or vehicle tracking for multiple vehicles	Heavy losses for multiple vehicles	Significant impact for multiple vehicles

dimensions. First, in terms of the safety of the vehicle, its occupants and other road users. Second, in terms of the privacy aspect of data assets whether this data is relative to the vehicle users (personal data) or the vehicle manufacturer and equipment suppliers (i.e. intellectual property). Third, in terms of financial losses that result from a successful attack on the vehicle owner, ITS¹ operators and car manufacturer. Fourth, in terms of interference with the operational performance of the vehicle and ITS functions. Table 2.2 presents the definition of the respective severity classes.

2. *Probability of success:*

Also called “Attack potential”, is defined with regards to five different aspects and should be estimated for each attack scenario. Here an attack scenario represents the technical details of how the defined threat could be accomplished. In this sense, for each defined threat there could be multiple attack scenarios. The first aspect is the *time* necessary to conduct the attack. It includes the time spent on reverse-engineering the system, identifying possible vulnerabilities and developing the attack method to

¹Intelligent Transportation System

2. STATE OF THE ART

leverage them. Second, the *expertise* required in order to conduct the attack scenario. Four different expertise levels are defined: Layman, Proficient, Expert and Multiple experts. Intermediate expertise levels can also be selected. Third, the level *knowledge of the system* required describes how easy it is to access the needed information. Fourth, the *window of opportunity* during which the attack is possible. Finally, the necessary *equipment* required to perform the attack steps of the attack scenario. The rating details of the attack potential aspects are presented on table 2.3.

Table 2.3: Rating of aspects of attack potential [52]

Factor	Level	Value
Elapsed Time	\leq 1-day	0
	\leq 1-week	1
	\leq 1-month	4
	\leq 3-months	10
	\leq 6-months	17
	$>$ 6-months	19
	Not practical	∞
Expertise	Layman	0
	Proficient	3
	Expert	6
	Multiple experts	8
Knowledge of the system	Public	0
	Restricted	3
	Sensitive	7
	Critical	11
Window of opportunity	Unnecessary/unlimited	0
	Easy	1
	Moderate	4
	Difficult	10
	None	∞
Equipment	Standard	0
	Specialized	4
	Bespoke	7
	Multiple bespoke	9

These attack potential aspects should be summed up in order to come up with the attack probability. Intuitively, a small attack potential sum is an indicator of an attack scenario rather easy to perform, thus will be associated with a high success probability. Inversely, a high attack potential sum is an indicator of an attack scenario rather difficult to perform and will be associated with a low success probability.

3. *Controllability:*

It is a parameter that is specified for attacks with safety impact. It expresses the

probability that the driver can influence the severity of the outcome. Table 2.4 reports the controllability classes and their meaning.

Table 2.4: Controllability classes

Controllability	Meaning
1	Avoidance of an accident is normally possible with a normal human response.
2	Avoidance of an accident is difficult, but usually possible with a sensible human response.
3	Avoidance of an accident is very difficult, but under favorable circumstances some control can be maintained with an experienced human response.
4	Situation cannot be influenced by a human response.

The controllability is estimated for each threat regardless of the attack scenarios and attack steps. Note that the score of the controllability increases as the situation becomes uncontrollable.

4. *Risk:*

Finally, after the estimation of the previous factors, the EVITA risk assessment methodology defines a mapping table that assigns for each threat the associated risk. The mapping is presented in table 2.5. The intuition is that as the probability of success increases (similarly the severity level and controllability level), the risk level also increases. The risk is a four dimension vector that indicates a risk level for each dimension of the severity.

The outcome of this procedure is a list of threats ranked according to their contribution to the overall risk. Managing the risk involves threat processing through the appended security countermeasures specification, the purpose of which is to limit or even completely cancel the threat and thus reduce the risk. Finally, the result should be a set of strong security requirements on how the system should behave, what are the security concepts and security measures to be implemented in order to guarantee the required level of security (at least from a design perspective).

2.5 In-Vehicule Secure Communication survey

As explained in the previous sections, in-vehicle communication buses constitute the backbone of the vehicle as they ensure most of the communications between ECUs. In fact, the vehicle as a system would not be able to function or even start without at least the core electronic components successfully communicating over internal networks. They are a central building block for nearly all automotive functions that span through multiple ECUs. There are multiple

2. STATE OF THE ART

Table 2.5: Risk level as a function of the attack probability, severity level, and controllability

Controlability	Severity	$P = 1$	$P = 2$	$P = 3$	$P = 4$	$P = 5$
C=1	$S = 1$	0	0	1	2	3
	$S = 2$	0	1	2	3	4
	$S = 3$	1	2	3	4	5
	$S = 4$	2	3	4	5	6
C=2	$S = 1$	0	1	2	3	4
	$S = 2$	1	2	3	4	5
	$S = 3$	2	3	4	5	6
	$S = 4$	3	4	5	6	7
C=3	$S = 1$	1	2	3	4	5
	$S = 2$	2	3	4	5	6
	$S = 3$	3	4	5	6	7
	$S = 4$	4	5	6	7	7+
C=4	$S = 1$	2	3	4	5	6
	$S = 2$	3	4	5	6	7
	$S = 3$	4	5	6	7	7+
	$S = 4$	4	5	6	7+	7+

communication bus technologies used for this purpose. Table 2.1 gives a summary of the main communication bus technologies used in today's cars. The Controller Area Network (CAN) bus is by far the most commonly used one.

The constant evolution of the threat landscape facing modern connected vehicles makes the adoption of *defense in depth* necessary. This imposes to no-longer consider in-vehicle networks as a *trusted domain*, but rather as the last line of defense against elaborated attacks that need to use internal communication networks in order to spread from one node to another. For these reasons, it is necessary to propose solutions that can overcome the weaknesses of these communication buses.

It is worth noting that there is a tendency to substitute old technologies like CAN and LIN, and move towards integration of more advanced ones like Ethernet, especially to comply with growing data and throughput need. This tendency is mainly driven by the need for more multimedia content use and the integration of autonomous driving technologies that generate a considerable amount of data. The good news is that these new technologies can offer the capability to integrate security features that can cope with integrity, authenticity and confidentiality needs. Although, the trend is slowed by the low cost of classical technologies and high implementation cost of relatively new communication technologies.

In this section, we present a survey of the proposed security methods and their principles that can be applied to the internal communication network of the vehicle, in particular, applied to the CAN-bus. We begin by presenting an overview of the CAN protocol in section 2.5.1. Then we present the protocol weaknesses in section 2.5.2. We identify three state-of-the-art protection mechanisms, namely *payload protection*, *identifier protection* and *intrusion*

Data Link (Layer 2)	ISO 11898-1	
Physical (Layer 1)	ISO 11898-2 [CAN High speed]	ISO 11898-3 [CAN Low speed]

Figure 2.5: CAN layer model

detection systems that we present in details in sections 2.5.3, as well as their limitations. We conduct a comparative study and discuss the effectiveness of these protection mechanisms in section 2.5.4.

2.5.1 Controller Area Network Overview

The Controller Area Network (CAN) is an asynchronous, serial field-bus system. It was introduced in 1983 by Bosch company for the networking of control devices in automobiles. The aim of this communication bus was the reduction of cables length and weight. Since 1991 [120], CAN is internationally standardized as ISO 11898 and defines the Layer 2 (Data Link Layer) [60] and Layer 1 (Physical Layer) [61, 62] of the OSI reference model presented in figure 2.5. The physical layer can be realized in two versions: high-speed CAN (ISO 11898-2) [61] and low-speed CAN (ISO 11898-3) [62]. Usually, these layers are implemented respectively in a CAN transceiver and a CAN controller. It is known to be very resistant to electromagnetic interference and thus found wide adoption in vehicular onboard networks.

A CAN frame (figure 2.6) is composed of multiple fields:

- SOF: is the Start Of Frame bit that signals to all CAN controllers that a new frame is about to be sent on the bus.
- ID: is the arbitration field, also called identifier as it identifies the data payload. It is composed of 11 bits for standard CAN and 29 bits for extended CAN frames. Its main function is to prioritize physical access to the bus. Prioritization will be detailed in the rest of the section. The identifier is relative to the frame data in the sense that it is not used to identify the sender or the receiver but rather the content of the frame. The proper intended use of the CAN protocol makes an important hypothesis on the use of identifiers within the same network: There must not be two CAN-controllers that can send the same identifier, i.e. each identifier is sent by *at most* one ECU.
- RTR, IDE, DLC: these are control fields that indicate the type of the frame and the length of the data it contains.
- Data: is the data field. It contains the payload information (also called signals) shared between the sender and the receivers.
- CRC: is the Cyclic Redundancy Check field. It ensures the integrity of the data field.
- ACK: is the acknowledgment field, sent by all the controllers that *correctly* received the frame. This field attests that the CRC field was correctly validated and no errors occurred during the transmission.

2. STATE OF THE ART

- EOF: is the En-Of-Frame field.

S O F	Identifier	R T R	I D E	r	DLC	Data	CRC	ACK	EOF
1	11 bits	1	1	1	4 bits	0-8 bytes	16 bits	2	7 bits
	Arbitration Field	Control-Field				Data-Field	Check-Field		

Figure 2.6: CAN frame

Each ECU that communicates on the CAN bus is equipped with a CAN-controller and a CAN-transceiver (figure 2.7). The CAN bus is event triggered protocol. On the application level, whenever the ECU software wants to send a message on the CAN bus, it needs to specify the identifier (ID) and the data payload to the CAN controller (layer 2). This information is temporarily stored in a buffer waiting to be sent on the bus. The CAN controller then constructs the appropriate CAN frame by adding the remaining fields then passes it to the CAN transceiver (layer 1) whose role is to send the frame on the communication bus physically. The CAN-transceiver then needs to check if the bus is free (no information is being sent) in order to trigger a frame sending process.

The CAN bus implement a *Carrier Sense Multiple Access / Collision Avoidance* (CS-MA/CA) media access control method. The collision avoidance property is implemented by means of the arbitration field (frame identifier) that defines the priority of the message. In fact, whenever a node needs to send a frame on the bus, it needs to check whether the bus is free, then it starts sending. Because each identifier is sent by *at most* one ECU, two ECUs that want to send a message at the same time will necessarily present different frame identifiers. The arbitration phase is processed at the bit level according to a straightforward rule: in case of collision, the dominant bit (“0”) wins the arbitration over the recessive bit (“1”). This means that in case two ECUs want to send frames at the same time, and in case of collision at the bit level between a dominant bit (“0”) and a recessive bit (“1”), the dominant bit is physically transmitted. Thus, each CAN controller implements a loop-back that monitors the bus for possible collisions and that decides whether to continue sending or to drop and retry when the bus becomes available again. Given that the fact frame sending process is executed in a serial fashion (one bit at a time) starting from the Most Significant Bit (MSB), the arbitration rule translates to the following: in case of collision, the lowest identifier (in arithmetic representation) wins the arbitration.

Usually, for safety reasons, safety-critical signals are assigned to priority message identifiers. The more the signal is safety-relevant, the higher the priority is assigned to its identifier.

CAN Bus Error Handling mechanism: Error handling is one of the most important features that are built into the CAN protocol, and that makes it attractive from a safety

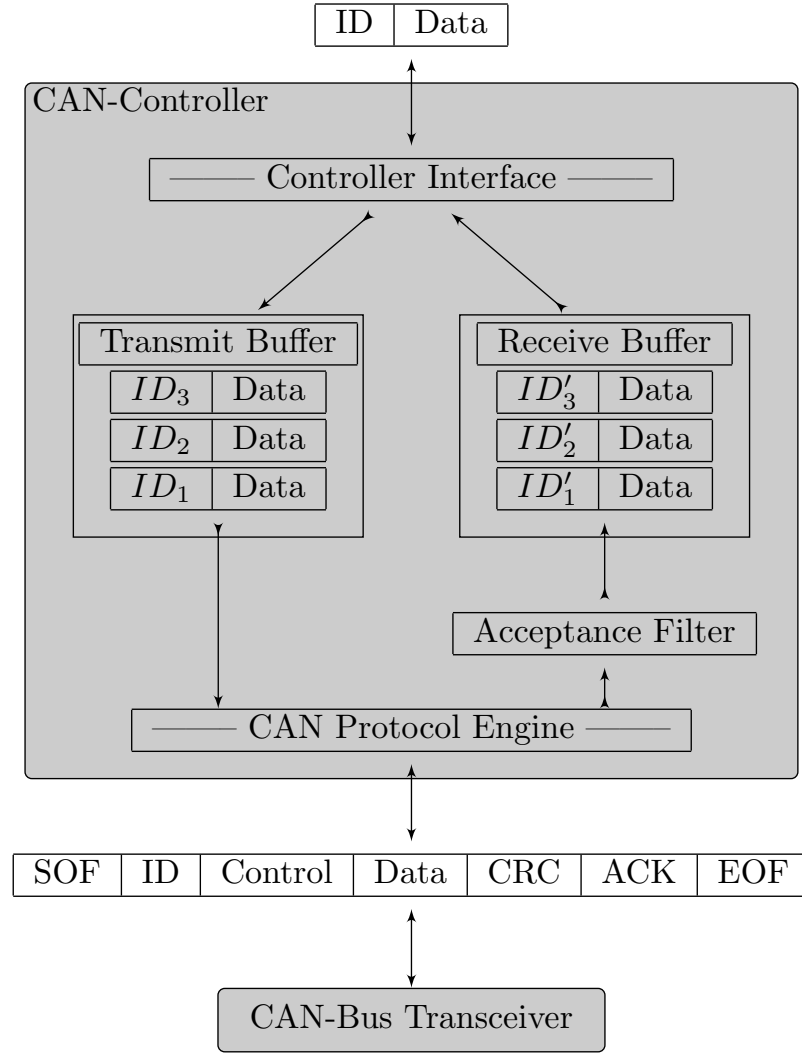


Figure 2.7: CAN controller

point of view. It aims at detecting errors in messages appearing on the CAN bus, in order for the transmitter to re-transmit an erroneous message. All the CAN controllers connected to the bus implement these error handling mechanisms and will try to detect errors within a message. An error flag is transmitted by any node that discovers an error, thus destroying the bus traffic. The other nodes will detect the error caused by the Error Flag (if they have not already detected the original error) and take appropriate actions (i.e., discard the current message).

Each node maintains two error counters: the *Transmit Error Counter* (TEC) and the *Receive Error Counter* (REC). In principle, a transmitter detecting a fault increments its Transmit Error Counter. Moreover, a receiver detecting a fault increments its Receive Error Counter. A node starts in Error Active mode. When any Error Counter raises over a certain “error threshold”, the node will first become “error passive”, that is, it will not actively destroy

2. STATE OF THE ART

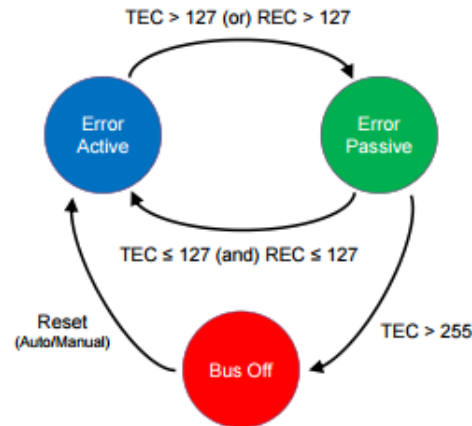


Figure 2.8: Error handling in the CAN bus

the bus traffic when it detects an error, and then turn into “bus off” mode, which means that the node does not participate in the bus traffic at all. Figure 2.8 gives more details about this process.

Five different error detection mechanisms are implemented:

1. Bit Monitoring: Each transmitter on the CAN bus implements a loop that reads back the transmitted bits. If the transmitted bit differs from the one that is read on the bus, a Bit Error is raised. During arbitration, no bit errors are raised. However, this mechanism serves the transmitter to know it won the arbitration or not.
2. Bit Stuffing: Each node that transmits five consecutive bits of the same level (five consecutive “0” bits, or five consecutive “1” bits) is required to add a sixth bit of the opposite level. At the reception, the receivers will remove this extra bit. Thus, if more than five consecutive bits of the same level occurs on the bus, a Bit Stuffing Error is raised.
3. Frame Check: Some parts of the CAN message have a fixed format defined by the standard (for example the CRC Delimiter, ACK Delimiter, End of Frame). If a CAN controller detects an invalid value in one of these fixed fields, a Form Error is raised.
4. Cyclic Redundancy Check: Each message features a 15-bit Cyclic Redundancy Checksum (CRC). All the nodes that are receiving the message have to calculate the checksum based on the data field. A CRC Error is raised if a difference is found between the calculated checksum and the received one.
5. Acknowledgement Check: All nodes on the bus that correctly receive a message are expected to send a dominant bit as Acknowledgement in the message. Thus, if the transmitter can not detect a dominant level in the ACK slot, it means that no node has received the message correctly and an Acknowledgement Error is raised.

2.5.2 CAN Weaknesses

The CAN bus thus presented has a good robustness property that matches the high safety requirements needed and that allows it to be used in vehicles. Nevertheless, it has weak security properties that caused it to be leveraged to mount cyber-physical attacks against modern cars. Some of these weaknesses are inherent to the bus technology itself, and some others are due to the way it is used in the automotive industry. It is important to make the difference between these two types of weakness as the ones that are inherent to the technology cannot be removed and fixed unless the standard is revised. For the others, protection solutions can be developed.

In what follows we give an overview of these weaknesses in the form of attacks. These attacks suppose that the attacker has gained access to the communication bus. This access can be either granted through direct physical access to the bus, where an attacker plugs a controlled device equipped with a CAN controller, or through an indirect or remote physical access where an attacker somehow gains control of a legitimate ECU and uses it to launch the attacks.

2.5.2.1 Denial-of-Service

There are two possible ways for an attacker to perform a Denial-of-Service attack over the CAN bus depending on the targeting impact.

ID-based Denial-of-Service: One of the first attacks that have been published and that target the CAN bus is the ID-based DoS [29, 54, 94]. The principle is simple and consists in exploiting the principle of arbitration implemented in the CAN bus. In fact, as we already stated in section 2.5.1, during the arbitration phase, the lower the identifier, the higher the priority. This means that the lowest identifier (in arithmetic representation) will *always win* the arbitration phase. The attack principle is then to continuously send a message with the identifier $[ID = 0]$, which is the lowest possible identifier. Even if the message will not be effectively consumed by any ECU, it will nevertheless cause other messages with higher identifiers to *always loose* the arbitration phase in favor of the attacker message. The effect is that all messages will be blocked and no message other than the attacker message will be sent on the communication bus. This attack is inherent to the used technology as it exploits the arbitration principle that is implemented by the standard itself.

Error-based Denial-of-Service: Recall from section 2.5.1 that one of the hypotheses that the standard makes on the use of the CAN protocol is that within the same network there must not be two CAN-controllers that can send the same identifier. In fact, this assumption guarantees that at the end of the arbitration phase, only one node will have the possibility to send further information (i.e. continue to send “data” content of the frame). However, if this assumption is not respected, it can jeopardize the integrity of the protocol and generate errors due to conflicts in the data fields. An attacker can use this behavior to his advantage. More concretely, if a legitimate ECU_i wants to send a frame with an identifier ID_i , and if an attacker succeeds to synchronize with ECU_i and also tries to send another frame but with same identifier ID_i . During arbitration, since there is no difference between

2. STATE OF THE ART

the identifiers, both CAN controllers of ECU_i , and the attacker will have the illusion that they have won the arbitration and thus can proceed with sending the remaining fields of the frame. If the data payload of the frame of ECU_i and the data payload of the attacker frame are not the same then necessarily a bit error will be generated that will translate in a bad Acknowledgment (ACK) field. In general, this will cause ECU_i to send the frame again. However, it will also trigger the controller error counter, as the ECU will believe that the CAN transceiver somehow caused the error. With enough persistence in repeating the same process successfully, the attacker can cause the error-counter to overflow, which the CAN-controller at the ECU level will interpret as if CAN-transceiver is generating too many errors and will automatically disconnect causing the ECU to be no-longer be part of the network and out of service. This attack has been demonstrated by Palanca et al. [104] and is relatively sophisticated and can target a specific ECU, knowing at least one of the identifiers that it sends.

The weakness is inherent to the used technology as the error counting process is specified in the CAN standard. One possible remedy to this attack is to try to reconnect the CAN-controller after being forced to disconnect. It has nevertheless one drawback which occurs when the ECU is not under attack but rather a safety hazard like short-circuit happened at the transceiver level. This safety hazard will cause the ECU implementing the remedy to continuously try to reconnect which in return will short-circuit the entire communication bus.

2.5.2.2 Reverse engineering

The automotive industry uses the CAN protocol for periodic and event-based messaging between ECUs. Each car manufacturer specifies a protocol on top of CAN, shared between all the ECUs, for them to understand the information (sensor values and commands) being sent from one node to another. This protocol is proprietary. If an attacker were to understand the content of the protocol he/she can mount attacks like *ECU impersonation* and *ECU exhaustion* that we explain after. The first thing that the attacker needs to collect is the used identifiers and their periodicities. This is relatively easy and straightforward since the attacker can simply listen on the bus and capture traffic. The analysis of the traffic can reveal this type of information. This knowledge can be used later to target specific functionalities. Next, an attacker can also try to understand the content of the data payload for each identifier. This is relatively more complex to do since he/she needs to perform more complex analysis on the captured data using more advanced statistical tools like signal correlation analysis [14] and comparison of multiple CAN log captures [118].

From a security point of view, the attack constitutes a confidentiality violation. In fact, since the protocol is proprietary and not public, it can be considered as confidential information. Besides, reversing the protocol gives access to more sensitive information of the car manufacturer, but also private information. For instance, Martinelli et al. [91, 92] have shown that CAN traffic can be exploited for instance to identify the driver.

The attack is not inherent to the CAN protocol, and car manufacturers can protect at least the data payload. In section 2.5.3 we examine the possible ways of protection against reverse engineering.

2.5.2.3 Fuzzing attack

The fuzzing attack constitutes in sending multiple messages in the CAN bus in order to confuse ECUs, or in order to discover and test functionalities. The fuzzing attack can be considered as a technique to reverse-engineer (in an active way) the car manufacturer protocol on top of CAN as it allows to trigger functionalities and observe the effect on the car. During the attack, the attacker can try to change and modify the different fields of a CAN frame. For instance, send multiple identifiers, send the same identifier with multiple data length codes or with multiple data fields. All of these fields can be carefully chosen in order to build a smart fuzzing strategy. Such attacks are already supported in some tools dedicated to the CAN protocol [14].

The attack is not inherent to the CAN protocol. Protection techniques against this type of attacks can be adopted, to some extent, in order to stop/block frames.

2.5.2.4 Impersonation attack

As mentioned previously, the CAN protocol does not implement proper source/destination addressing in the sense that the node does not have identities on the network. The identifiers are rather used to identify the message itself but not the destination. It indicates the source as the CAN network assumes that each identifier is sent at most by one node. However, technically there is nothing that blocks a second node from sending a message with the same identifier as the first node. From an attacker point of view, this makes an ECU impersonation possible if there is no source control at the data payload level. In fact, and as was reported by multiple research papers [28, 29, 55, 94, 104], car manufacturers do not implement source authentication. The principle of the attack is thus to capture a message and replay it on the CAN bus. At receiving end, the ECUs will assume that it is an authentic message sent by the legitimate ECU, and will process the data allowing the attacker to manipulate sensor values, and command actuators. This technique was tested successfully on multiple cars and was used to reproduce effects on cars such as activating the horn, activating lights and in some cases even cause safety-critical scenarios like activating the breaks and opening the doors at high speed. The attack can even be used to impersonate the diagnostic tool and open diagnostic sessions with ECUs that give it the ability to perform any type of action including the most safety-critical ones. The attacker can either replay a message recovered from network captures, or completely forge a message. These two alternatives are respectively called *Message Replay Attack* and *Message injection Attack*.

This attack is not inherent to the technology but rather to the way the CAN protocol is used. Car manufacturers can use the CAN protocol differently and introduce security features in order to block this type of attacks. In fact, from a security standpoint, it constitutes an authenticity violation. In the following sections, we will expose possible methods that can block this type of attack and their drawbacks.

2.5.2.5 Exhaustion attack

To exhaust an ECU in this context is to give it more information than what it actually needs, for instance, with an update frequency superior to the original update frequency. The intent of the attack is to overwrite the original messages by issuing even more messages than the

2. STATE OF THE ART

Table 2.6: Summary of CAN Weaknesses

Weakness	Ref.	Violation type	Inherent	Not inherent
ID-based DoS	[29, 94]	Availability	✓	
Error-based DoS	[104]	Availability	✓	
Reverse-engineering	[14, 118]	Confidentiality		✓
Fuzzing	[14]	Authenticity		✓
Impersonation attack	[54, 94]	Authenticity		✓
Exhaustion attack	[54]	Availability & Authenticity		✓

legitimate ECU.

This attack is not inherent to the CAN standard and is related to the way the protocol is used in the automotive industry. Possible remediation strategies to this weakness exist and will be presented in section 2.5.3.

Table 2.6 gives a summary of the attack types over the CAN protocol. Attacks that are not inherent to the CAN standard are related to the way the automotive industry uses the standard. In the next section, we investigate possible protection methods for these attacks.

2.5.3 Protection mechanisms

Knowing the CAN weaknesses, and in order to protect in-vehicle communication buses, it is necessary to understand the attacker model and attacker objectives. In section 2.3 we exposed the attack surface of connected cars. From this attack surface, we concluded that an attacker can have three types of access to in-vehicle communication buses: direct physical access, indirect physical access, and remote access. Depending on the access type, the attacker has some capabilities with regards to the in-vehicle communication buses. An attacker has direct physical access when he/she can connect a controlled attack device directly to the communication bus. In this case, the attacker can read frames communicated on the communication bus and can interact with other ECUs by crafting and sending frames. An attacker has indirect and remote physical access to the communication bus when he/she somehow succeeds in taking partial or full control over a legitimate ECU, for instance leveraging a software or hardware vulnerability. In this case, the attacker can, not only, read and write frames but also modify the content of legitimate frames and impersonate the controlled ECU. Figure 2.9 illustrates the difference between these two types of attackers: in figure 2.9-a we represent an attacker with direct physical access to the CAN bus. This attacker is capable of sending an entire *extra* CAN frame (in red) to accomplish the attack. In figure 2.9-b we represent an attacker that has indirect/remote access to the CAN bus leveraging the control of one of the legitimate ECUs. This attacker is capable of not only sending an extra message but also directly modify the content (payload in red) of a legitimate message.

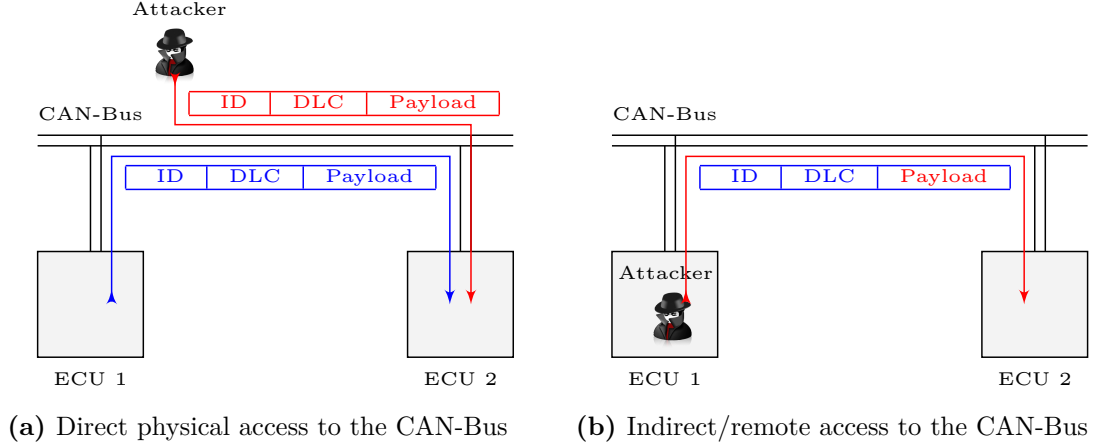


Figure 2.9: Attacker models

Protection solutions have been proposed, depending on the adopted attacker model, in order to neutralize their capacity of extracting information and/or harming the system. We identified in the literature three main families of protection mechanisms that are proposed for this purpose.

- Payload protection: in reference to the frame payload, this protection mechanism uses cryptographic primitives in order to protect the data part of the frame.
- Identifier protection: in reference to the frame identifier, this protection mechanism uses randomization strategies to protect the identifier part of the frame.
- Intrusion detection system: this protection mechanism uses multiples detection principles each one designed to trigger one specific bad behavior.

In what follows, each family of protection solutions is detailed.

2.5.3.1 Payload protection

One of the first and obvious family of solutions that were proposed to secure the CAN bus is the payload protection solutions. In fact, when dealing with an authenticity and/or confidentiality violations, the first step towards overcoming this weakness is to protect the payload data with cryptographic mechanisms. Nilsson et al. [102] proposed a delayed data authentication using compound message authentication codes. Their idea is to use Cipher-Block Chaining Message Authentication Code (CBC-MAC) along with the KASUMI encryption algorithm [40] in order to send message authentication codes over consecutive CAN frames to authenticate the messages. In [50] Hartkopp et al. proposed a security concept that uses symmetric authentication measures such as the Cipher-based Message Authentication Code (CMAC) algorithm to authenticate signals and time-stamps to guarantee the freshness of the messages. In [129] Van Herrewege et al. propose a broadcast CAN authentication solution

2. STATE OF THE ART

that uses the least significant bits of HMAC [80] using group authentication keys. In [47] Groza et al. present an ECU authentication solution called *LiBra-CAN* that rely on Mixed Message Authentication Codes (M-MACs). The proposed solution needs to assign pre-shared encryption keys to groups of ECUs. In [97] Mundhenk et al. introduce an *Open source simulator for real-time performance analysis of automotive networks*. This simulator was used later in [98] to evaluate and compare between multiple ECU authentication and authorization strategies.

The payload protection family targets the capacity of the attacker that has direct physical access to the communication bus, to read and/or write data. It neutralizes its capability to *understand* the content of the payload data, and to *impersonate* an ECU depending on the used cryptographic primitives.

Limitations:

The main limitation of this type of protection is that the produced data is larger than the original data which causes a bandwidth overhead on the communication link. For instance, Van Herrewege et al. [129] introduce an additional 112 bits authentication code (including a counter and a message signature) in the frame payload for each message that needs authentication. In fact, here the goal is to protect the confidentiality/authenticity of the sent data, we have to send a data authentication code along with the original data. State-of-the-art encryption solutions suggest using no less than 128 bits encryption key with a block cipher of minimum 64 bits (128 bytes for AES-128). Similarly and according to NIST [37] and FIPS [41] recommendations, cryptographic authentication codes should have a minimum length of 64 bits, when no additional measures to limit the validation rate are taken. This involves a significant increase in data size. Although both Hartkopp et al. [50] and Schweppe et al. [116] argue that since CAN network has limited speed, the CMAC can be further be reduced to 32 bits if session keys are used. While the impact of this transformation could be negligible for only one message, the generalization of the use of such solutions to all the messages will cause a significant network overhead. This will have practical side effects such as increasing the delay of messages and increasing errors on the CAN network. Furthermore, the payload protection mechanism does protect the confidentiality and integrity of the data but does not protect against reverse-engineering. Since the identifiers are kept unchanged, the attacker can still reverse-engineer part of the car manufacturer protocol. Besides, the attacker can still perform exhaustion attack on the ECUs, by sending messages with correct identifiers but wrong payload, thus forcing dummy and useless processing on the receiving side and leading to a DoS attack. In fact, before accepting the message, each ECU will have to perform a message verification step in which it will re-compute the authentication code. This step will introduce processing delays that can be more or less important depending on whether the security function is software or hardware enabled (cf. [98] for a comparison between hardware/software introduced delays). Additionally, in order to use payload protection mechanisms, we have to build key management strategies, including key distribution and key agreement protocol runs for each new session. This adds not only a deployment complexity to the existing system but also delays and possible de-synchronization issues.

2.5.3.2 Identifier protection

Recall from section 2.5.1 that each car manufacturer define a proprietary protocol on top of CAN and that is implemented inside all ECUs. This protocol defines which identifiers are sent across the CAN network and for each identifier the content of the payload information of the corresponding frame. The idea behind identifier protection solutions is to make the message identifiers not fixed, but instead continually changing in a way that the sending and receiving ECUs can synchronously agree on the same identifier for a given message. The underlying principle is also called an *obfuscation process* or *randomization process*.

Humayed et al. [56] proposed a solution called ID-Hopping to counter Denial of Service attacks directed against a specific message. Their method works closely with an intrusion detection mechanism¹ which is needed to identify the existence of an undergoing attack against a specific message. Once the attack has been detected, the ID-Hopping mechanism is activated. Its role is to assign a new but previously defined, identifier as a substitute identifier for the attacked message. Another interesting solution to protect the CAN protocol is to regularly randomize the identifier. The constraint is that both sender and receiver need to use the same identifier. Han et al. [48, 49] proposed a candidate randomization function that includes a random number at the least significant bits of the identifier. In [87] Madl et al. propose to re-assign a CAN profile, defined as the set of message identifiers used by the car manufacturer, to the car so that each vehicle at a given time would be using a different CAN profile. This would increase the complexity of large scale attacks against a fleet of vehicles.

This security principle is efficient to protect the CAN bus from identifier reverse-engineering, large scale attacks as well as injection and replay attacks. In fact, if the identifier is not fixed across vehicles, a large scale attack that could affect all the cars is no longer possible as the identifiers of messages will change from one vehicle to another. Moreover, if the identifiers are not fixed within the same vehicle, a frame replay attack will not succeed because the identifier is continuously changing, and thus no ECU will catch the replayed frame. A frame injection attack neither will work, because the attacker will have to “predict” what will be the next identifier to be injected in order for the attack to be successful.

Limitations:

It is clear that this protection technique allows protecting the identifier but does not allow to protect the payload data of the frame. Nevertheless, in principle, it could be combined with other protection mechanisms (payload protection and intrusion detection system) Additionally, depending on the used protection strategy, the level of protection against replay/injection attacks and reverse-engineering is not the same. For instance, the proposition of Humayed et al. [56] that allows agreeing on a new yet predefined identifier in case of attack, does not protect against identifier reverse-engineering per se (as defined in section 2.5.2.2) as well as injection and replay attacks. While, the proposition of Han et al. [48, 49] that allows a dynamic identifier set offers a higher level of protection against these attacks. In chapter 4 we visit more in details this family of protection mechanisms and study their effectiveness.

¹Intrusion detection systems are another family of protection solutions and will be presented in the next section

2. STATE OF THE ART

2.5.3.3 Intrusion Detection and Prevention Systems

Another family of protection solutions is known as in-vehicle network Intrusion Detection and Prevention Systems (IDPS). Their role is to monitor in-vehicle networks (CAN for instance) and perform an analysis of the passing traffic on the entire sub-network for suspicious behavior. Once an attack is identified, or abnormal behavior is sensed, the alert can be raised and prevention counter-measures, if any, can be adopted.

Detecting intrusions on the in-vehicle communication buses is important as it can prevent attacks from spreading to other ECUs. The main difference from the previously introduced protection mechanisms is that IDPS systems do not introduce fundamental design changes to the vehicle. They can be backward compatible with existing equipment. Nevertheless, they need to be placed at a strategic point(s) within the network to monitor the traffic to and from all ECUs.

Many mechanisms have been proposed to detect possible intrusions on the CAN bus. They are more or less effective depending on the attacker model (Figure 2.9) In general, state-of-the-art detection mechanisms can be categorized into two main classes: *Rule-based* detection mechanisms and *Anomaly-based* detection mechanisms. Table 2.7 gives a listing of state of the art intrusion detection mechanisms applied to the CAN bus. Figure 2.10 gives a high-level overview of these mechanisms applied to the CAN protocol. Prevention on the CAN network can be either by filtering out the attack frames or by killing the suspicious frames by causing a frame error. Advanced prevention methods that rely on error correction also exist. In all cases, prevention mechanisms can be adopted when the confidence level is very high.

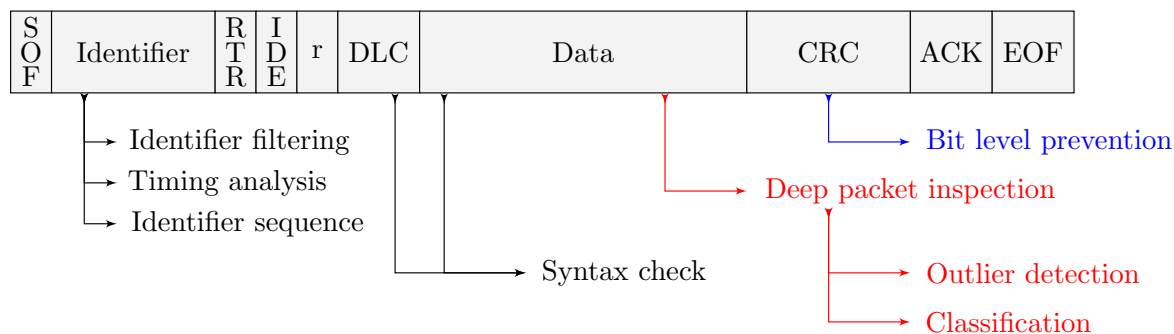


Figure 2.10: High-level synthesis of detection mechanisms applied to the CAN frame

Rule-based Intrusion detection *Rule-based* intrusion detection mechanisms, tend to define specific rules of how the network traffic should be. These rules can be extracted directly from specifications, or known attack *signatures*, or observed behavior of the network traffic. Any message that does not comply with these predefined rules is reported as intrusions. Therefore possible prevention mechanisms could be applied to it. Examples of such rules can be defined for the Gateway ECU that implements routing functionalities between different network segments or syntax of messages as defined by the manufacturer . . .

- **Identifier filtering:**

Based on the identifier, an intrusion detection system can establish a list of *allowed* and *forbidden* identifiers, based on which it can decide which frames to accept and which frames to filter. This technique is best known as *identifier filtering* or *identifier white-listing* [95, 100]. Such a white list can also depend on the context of the vehicle: for instance, the intrusion detection system may allow specific identifiers when the vehicle is on parking state, and reject them when the vehicle is moving.

- **Diagnostics:**

In order to detect attacks that try to open diagnostics sessions while the vehicle is moving, Miller et al. [94] proposed to define detection rules of diagnostics messages based on the state of the vehicle. The idea is based on the fact that the vehicle needs to implement a diagnostics policy that defines when certain diagnostic requests are allowed or not. This diagnostics policy can be featured inside each ECU or on a Gateway ECU whose role is to re-direct diagnostics requests to the appropriate sub-network. Implementing these rules can help enforce diagnostic policy and thwart attacks that try to exploit diagnostics commands.

- **Identifier timing analysis:**

In order to detect attacks that uses periodic messages, multiple methods [29, 54, 100] proposed mechanisms that take advantage of the identifiers *timing analysis*. In fact, since the goal of the CAN-IDPS is to protect against injection of extra packets onto the network, and given the fact that most normal frames have predefined frequencies (set-up during the design phase), then if the particular message does not respect its frequency, it should be reported as an intrusion. In other words, if the same message is received outside of its acceptance time-window, the system shall consider it as an intrusion and shall filter it out. Taylor et al. [124] proposed a detection algorithm based on the comparison of previous and current frame timings to implement this principle. Cho et al. [29] proposed an ECU authentication method based on identifier timing and used to detect frames sent by a possible attacker. Song et al. [119] proposed a light-weight intrusion detection algorithm based on the analysis of time intervals of CAN messages. Lee et al. [82] proposed a detection algorithm based on request/response message timing analysis.

- **Identifier sequence:**

In order to detect message injections it is also possible to focus on identifier sequences. In [125] Taylor et al. train long short-term memory networks in order to detect the correct identifier sequence. Marchetti et al. [89] proposed to use identifier sequence to define identifier transition matrix used to detect possible false transitions.

- **Word sequence:**

The same idea can also be defined to payload content and not only to identifier sequence. This method is called "word sequence". Studnia et al. [121, 122] proposed a language-based intrusion detection system that is based on word sequences.

- **Syntax check:**

Besides the identifier of the messages, the data length code (DLC) can also be exploited

2. STATE OF THE ART

to detect bad behavior. In fact, each manufacturer sets-up a proprietary protocol over the CAN standard. This protocol consists of creating a mapping between identifiers and payload information (sensor values for instance), also called signals, shared across all ECUs. This mapping defines a syntax that can be checked based on the payload length of each message. Müter et al. [100] proposed that messages that violate this syntax (i.e., messages sent with the wrong DLC or wrong signals) are then flagged as intrusions.

Anomaly-based Intrusion detection *Anomaly-based* detection mechanisms tend to define statistical measures computed over a window of time and used to classify normal and suspicious behavior. First, the goal is to create a model of trustworthy activity during a training phase, based on the statistical measure. Then in the operational phase, compare new behavior against this model.

- ***Entropy:***

An early work of Müter et al. [99] proposes to use the entropy of the CAN frames as a measurement to classify normal and abnormal behaviors observed on the CAN bus. Their approach is to measure the entropy based on frame bits over a window of time. This first phase serves to characterize the level of entropy (upper and lower bound) during the normal functioning of the CAN bus. The idea is that certain attacks will modify the level of entropy. For instance, during a Dos-attack (section 2.5.2.1), the entropy will be low, and inversely during a fuzzing-attack (section 2.5.2.3) the entropy will be very high. Thus the lower and upper entropy bounds will serve during deployment to detect these kinds of attacks.

- ***Hamming distance:***

Recently, Dario et al. [34] proposed an intrusion detection algorithm that identifies anomalies by computing the Hamming distance between consecutive payloads. Similar to the entropy, this Hamming distance is compared with minimum and maximum thresholds defined during the set-up phase, and that defines the normal behavior.

- ***Deep learning:***

Kang et al. [70] train a deep neural network structure to classify normal *versus* attack packets using probability-based feature vectors of packet payload bits. Training data were generated by the *Open Car Test-bed and Network Experiments (OCTANE)* packet generator [22]. Normal and attacked packets were necessary in order to train the algorithm.

- ***Hidden Markov model:***

Narayanan et al. [101] propose to build Hidden Markov Model of the normal behaviour of the car based on sensor values (or signals). Their work shows that it is possible to detect data manipulation attacks like speed discontinuity.

2.5.4 Advantages and disadvantages

The presented protection mechanisms presented in the previous section have different capabilities to thwart attacks on in-vehicle communication buses. The payload protection mechanisms

Table 2.7: Intrusion detection techniques applied to CAN

Ref.	Type	Measure	Application
Müter et al. [100]	Rule-based	vehicle state	All frames
Miller et al. [95]	Rule-based	Vehicle state	Diagnostics
Taylor et al. [124]	Rule-based	Identifier timing	Periodic and remote frames
Cho et al. [29]	Rule-based	Identifier timing	Periodic and remote frames
Song et al. [119]	Rule-based	Identifier timing	Periodic and remote frames
Lee et al. [82]	Rule-based	Identifier timing	Periodic and remote frames
Marchetti et al. [89]	Rule-based	Identifier sequence	Frame identifiers
Taylor et al. [125]	Rule-based	Identifier sequence	Frame identifiers
Studnia et al. [121, 122]	Rule-based	Word sequence	Frame identifier and payload
Müter et al. [100]	Rule-based	Syntax check	Frame length (DLC)
Müter et al. [99]	Anomaly-based	Entropy	Frame bit distribution
Dario et al. [34]	Anomaly-based	Hamming distance	Frame payload data
Kang et al. [70]	Anomaly-based	Deep neural network	Frame bit distribution
Narayanan et al. [101]	Anomaly-based	Hidden Markov model	Frame payload data

target, in general, an attacker that has direct physical access to the communication bus, as well as an attacker that has indirect or remote access to one of the legitimate ECUs, provided that the controlled ECU does not hold the shared secret. Nevertheless, even for an attacker that has direct physical access, some attacks are still possible like partial reverse-engineering and exhaustion attacks. Payload protection also has an impact on the data throughput that depends on the used cryptographic primitives. Moreover, they introduce a processing time for data verification that also depends on whether the cryptographic primitives are software or hardware based.

Similarly, identifier protection techniques have the ability to protect against the same types of attackers (i.e. attacker with physical access and indirect/remote access to a legitimate ECU provided that the controlled ECU does not hold the shared secret). Additionally, depending on the obfuscation strategy it can also protect against reverse-engineering (which can thwart a large scale attack against a fleet of vehicles) and exhaustion attack provided that the identifier verification procedure is faster than payload verification. Generally speaking, identifier protection does not have an effect on the data throughput but still introduce a processing time for verification that also depends on whether the obfuscation strategy is based on software or hardware processing components. Chapter 4 will be dedicated to this type of protection mechanisms.

Intrusion detection systems offer protection against an attacker that has direct physical access. All of these detection mechanisms focus on the integrity of the protocol (timing

2. STATE OF THE ART

analysis, syntax check, ...). Some intrusion detection techniques also offer protection against indirect/remote access to one of the legitimate ECUs. These are the mechanisms that perform *deep-packet-inspection* type of detection. Most of them are outlier detection techniques that are based on a statistical measure (entropy, hamming distance) that are effective against fuzzing and DoS attacks, but may fail against carefully crafted packets. Some of them are trained with machine learning techniques and need normal and attacked packet in order to recognize attacks. In chapter 5 we build an intrusion detection system in order to address this limitation.

2.6 Conclusion

In this chapter, we gave an overview of threats posed on the cyber-physical architecture of modern cars. We first presented the cyber-physical architecture of modern vehicles as well as possible attack vectors and examples of reported attacks. We presented possible countermeasures along with design methodologies to handle security problems. Additionally, we dedicated an important part in order to study possible countermeasures for in-vehicle communication buses. In view of state of the art, in chapter 3 we investigate and develop a framework for the formal modeling of the cyber-physical architecture of the connected vehicle. The framework allows defining formally the main components in the system architecture as well as an attacker model. It is used to generate attack trees useful for risk analysis step automatically. Similarly, given the state of the art in-vehicle protection mechanisms, in chapter 4 we investigate obfuscation procedures for the CAN-bus. These obfuscation procedures are applied to the identifier part of the frame and target an attack with direct physical access to the communication bus. Multiple strategies are compared based on their capacity to protect against reverse engineering of the manufacturer-specific communication protocol used on top of CAN as well as against injection and replay attacks. Finally, in chapter 5 we introduce a prediction-based intrusion detection system that uses machine learning technique in order to protect against an attacker that has indirect and remote access to one of the legitimate ECUs. We demonstrate the feasibility of the intrusion detection system on two safety-critical information and test its capacity to withstand multiple attacks.

In this chapter, we introduce the concept of attack trees as well as a formal modeling framework of the car architecture to help deduce minimum attack trees and exploit them to compute risks. Results presented in this chapter have been scientifically valued by an article published as a book chapter in *Cyber-Physical Systems Security* [71].

Contents

3.1	Introduction	43
3.2	Attack trees	45
3.3	A case study: speed acquisition and display system	49
3.4	Cyber-physical architecture formal model	50
3.5	Graph transformation system	58
3.6	Attack tree transformation	62
3.7	Security analysis and Countermeasure	68
3.8	Conclusion	73

3.1 Introduction

In chapter 2 we presented a survey of a wide range of possible attacks against modern vehicles. These attacks leverage multiple attack vectors, exploit a plurality of hardware and software vulnerabilities, and they chain multiple, sometimes complicated, steps that can take advantage of core components of the vehicle. Goals of the attacker start from a simple gain of information (privacy violation) to gaining full control of the vehicle by impersonating ECUs. However, in the automotive domain, security violations have a direct impact on the users' safety.

Facing this increasingly growing threat, car manufacturers have to guarantee a certain security level of the equipment embedded in the automobile. Thus, the management of risk is becoming the primary concern of automotive manufacturers, especially for the future fully

3. RISK ANALYSIS AND ATTACK TREE GENERATION

connected and autonomous cars. The obvious approach to this problem is to conduct a security assessment study. The goal of the security assessment is to identify the assets and the associated threats regarding availability, integrity, and confidentiality [16, 59]. There are available methods like EBIOS, TVRA, EVITA, and others that could be adapted to conduct such a study in the automotive domain. Ultimately the study will help the manufacturer decide where to best implement the security mechanisms. To do so, the risk is evaluated relative to each threat based on its impact and its likelihood.

$$Risk = \sum_i Impact(thr_i) \times P_{occ}(thr_i) \quad \text{where:}$$

- $\{thr_i\}$ is the set of identified threats.
- $Impact$ is a function that evaluates the impact of a given threat.
- P_{occ} is the likelihood or the probability of occurrence of the given threat.

While the impact of the threat has to be defined by domain experts, determining what threats are *likely* to occur is a little more complicated and strongly depends on the given architecture. Eventually, security experts have to imagine every possible way the attacker can exploit the system (called attack scenarios) in order to reach their objective. A fairly good way to model these attack scenarios and to document them is to use attack trees or the attack graphs. These formalisms are presented as one of the possible solutions put forward in the new *Cybersecurity Guidebook for Cyber-Physical Vehicle Systems (SAE-J3061)* [31].

The attack tree formalism allows understanding effectively how the attacker can reach its objectives. It also allows easy visual inspection as well as documentation of the possible attack paths. Moreover, it presents some advantages in particular in the automotive domain as it presents some similarities with *Fault Tree Analysis* and can be leveraged to address safety and security issues altogether based on similarities in the engineering processes [33]. These similarities have been, in part, put forward in the SESAMO¹ project whose goal was the analysis of safety and security cross influences in embedded systems at the architectural level including in the automotive domain. From a threat analysis point of view, it is used to refine the risk assessment step through the elucidation of the basic attacks that serve a particular scenario. They require nonetheless a lot of work and expertise to elaborate. This expertise encompasses knowledge of the vehicle architecture, software components, and protocols used as well as of basic attacks that target each of these items. That is why the automated generation of attack paths can be of great importance in this context. However, a formal model of the system under evaluation is required for this task.

Chapter contributions. In this chapter, we **first** propose a method to model elements of the cyber-physical architecture of the vehicle using graphs. The model captures the security policy implemented as well as vulnerability information and access rights. Besides we consider an attacker model as a set of attacks originating from *all* the attack vectors (short range, long range, and indirect physical access). The system and attacker are modeled with behavioral rules using graph transformation system.

¹SEcurity and SAFety Modeling: <http://sesamo-project.eu/>

Second, we use the model to generate possible attack paths (combinations of actions) that can be used by the attacker to drag the system into a vulnerable state. Thus the generated attacks are more detailed, and we can capture more information about the possible attacker actions. The simulation of this behavioral model will allow us to find *all* vulnerable states and to retrace attacker actions that allowed him to reach it. Using this information we generate an attack tree that summarizes all possible steps that allow the attacker to reach his goal. Based on such a model we can try to answer questions like:

- Is a vulnerable configuration/state reachable from an initial state? In other words is an attack scenario achievable on the proposed architecture?
- Which sequence of basic attacks the attacker has to perform in order to reach such vulnerable state?

In what follows, section 3.2 gives some preliminary notions and definitions of attack trees as well as some background on the attack tree generation problem. In section 3.3 we introduce a case study that deals with speed acquisition and display system, and that will be treated as an example throughout the chapter. Section 3.4 presents the formal framework that we established in order to model a given cyber-physical architecture of a vehicle. Section 3.5 introduces a graph transformation system as the underlying modeling language used, and the tool used for the generation the system state space. In section 3.6 we show how to transform the state space into an attack tree and how to perform the security analysis and introduce countermeasures of the generated attacks in section 3.7. And finally section 3.8 concludes.

3.2 Attack trees

3.2.1 Presentation and formal definition

Attack trees are a very famous graphical security model [27, 44, 77, 93, 108, 114] whose main purpose is to define and analyze possible attacks on a system in a structured way. They are generally used in industry for managing threats and perform a security risk assessment. One of their benefits is that they constitute a very useful tool to document very complex attack scenarios in a compact way. Additionally, they are very effective in representing threat scenarios in a structured way that open the door to more complex analysis and possibly the definition of effective countermeasures.

Attack trees have been popularized by Schneier [114] as a useful way to document and understand attacks on a given system and most importantly is a way of making decisions about how to improve the security of the target system. Figure 3.1 illustrates the concept and main elements of an attack tree. An attack tree is a set of leaf nodes structured using the conjunction [AND] and disjunction [OR] operators. The root node in an attack tree represents the attack goal (or threat), and leaf nodes represent atomic attacker actions (also called basic attacks). Each intermediate node in the tree is either a [AND] node or a [OR] node.

- The [AND] node has child nodes that represent different steps of achieving the goal. In this case, all child nodes must be *all* executed by the attacker in order for the parent node to be reached.

3. RISK ANALYSIS AND ATTACK TREE GENERATION

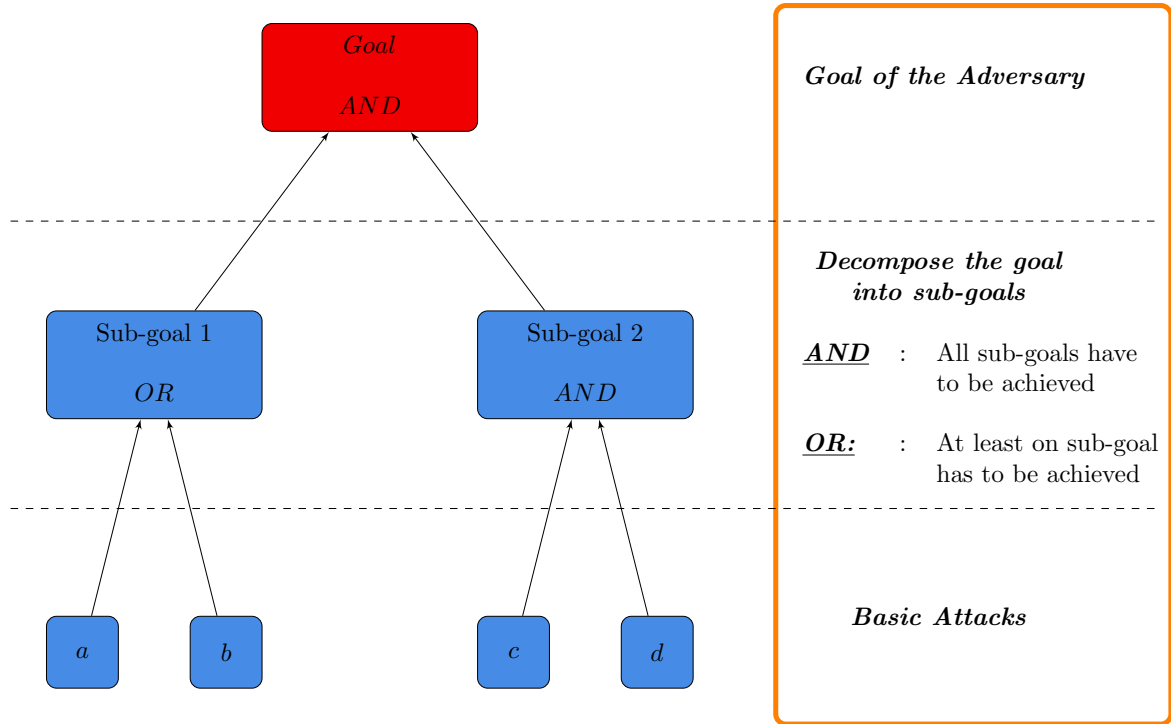


Figure 3.1: Attack tree decomposition principle

- The [OR] node has child nodes that represent different ways of achieving the goal. In this case, only one child node needs to be executed in order for the parent node to be reached.

Optionally, another type of node can be added to enrich the decomposition, denoted [SAND] (for Sequential AND) [68] and expresses the fact that not only child nodes must *all* be executed, but also must be executed sequentially in the right order (relatively to time or to a specific condition).

Graphical security models such as attack trees and attack graphs have been used for security analysis of SCADA systems [25, 127], network administration [15, 67, 117] automotive systems [12, 27, 52, 112, 113], socio-technical systems [36, 38, 83], ... More formally, and as formulated by Mauw et al [93], an attack tree can be defined with the following definition 3.1.

Definition 3.1. Let \mathcal{A} be the set of possible atomic attacker actions (or basic attacks), the components of the attack tree \mathcal{T} are $\mathcal{A} \cup \{AND, OR\}$, and the following grammar generates the attack tree:

$$t := a \mid OR(t, .., t) \mid AND(t, ..t) \quad \text{where } a \in \mathcal{A}$$

Later work of Jhawar et al. [68] extended this formal definition to include also a sequential combination of basic attacks. In this thesis, we consider only the basic form (i.e., with conjunctive and disjunctive combinations only).

However, a single attack tree may be represented differently. That is to say that two attack trees with different structures may represent the same set of attacks. In [93] Mauw et al. introduced an equivalence relation between such attack trees by defining reduction rules whose ultimate goal is to define an attack tree with its associated normal form. Informally the set of normal forms is defined as the set of attack trees with depth less or equal to one. In the rest of the chapter, we will only consider the normal form of an attack tree.

Attack trees have found their way to practice because they are well designed to support risk assessment studies. In fact, one of the main purposes it serves is to point out the set of attacks that are more likely to occur. Even more, they allow a wide range of qualitative and quantitative analysis methods [78, 81], which makes them perfect for analyzing the cost of attacks, skills needed, time, probabilities Most of these analysis techniques follow a bottom-up approach and propagate values from the leaves to the top of the tree. In figure 3.2 we report an example borrowed from the work of Henniger et al. [52] of analysis performed on an attack tree that describes possible ways to trigger unauthorized active braking. In this example, authors computed the attack potential of the basic attacks defined as the effort required to create and carry out an attack (in terms of needed expertise, knowledge of the system, equipment, . . .) according to ISO/IEC-18045 [64], and ISO/IEC-15408 [65] and then used the tree structure to propagate these probabilities right to the top in order to compute the risk.

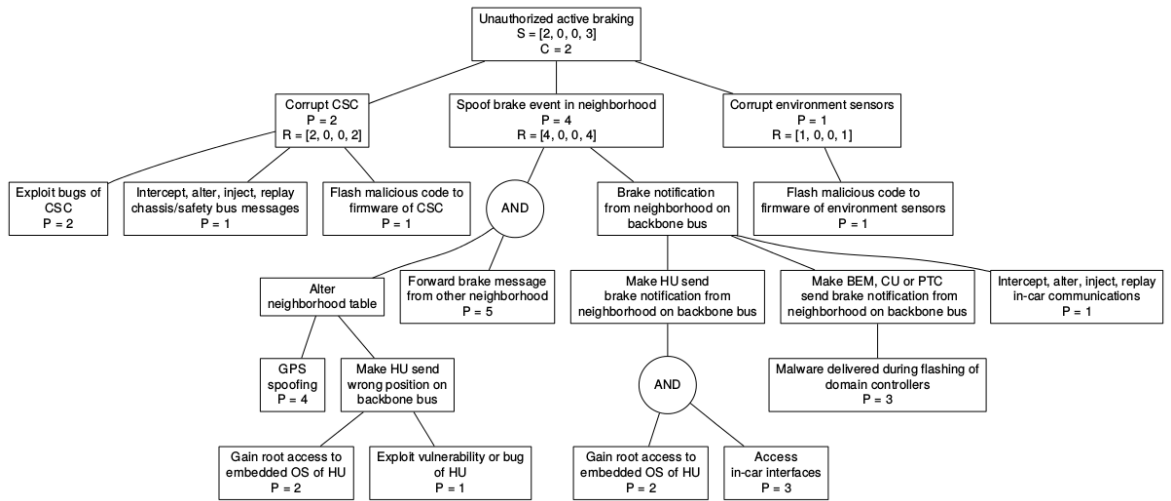


Figure 3.2: Attack tree for unauthorized active braking [52]

Tools that support such evaluation process are available. A number of commercial software applications for tree-like structures can be used for this purpose, like AttackTree [85] and SecurITree [126]. Free open source software solutions are also available. Kordy et al. [76] and Gadyatskaya et al. [43] built a tool, called *ADTool*, for automated bottom-up evaluation of security-relevant measures on attack trees. The tool also supports ranking of attack scenarios based on quantitative attributes entered by the user.

Nevertheless, the elaboration of attack trees can be a tedious task and error-prone for large

3. RISK ANALYSIS AND ATTACK TREE GENERATION

and complex systems. This is why automated techniques to generate such representations of attacks have been proposed.

3.2.2 Attack tree generation problem

3.2.2.1 Attack trees in the automotive domain

In the automotive domain, little work has been conducted in such direction. To the best of our knowledge the work of Salfer et al. [112, 113] is the only one that proposes such an approach. In [113], Salfer et al. present automated attack tree generation as a reachability problem of assets inside the cyber-physical architecture of the vehicle modeled as a graph. Nevertheless, the proposed model focuses on scalability issue using heuristic techniques and does not address the exhaustivity of the attack paths. Lugou et al. and Apvrille et al. [17, 86] use SysML-Sec modeling language to model safety and security aspects of the car architecture and formally prove safety (with Uppaal [20]) and security (with Proverif [21]) properties. In [18] Apvrille et al. explain how to use an input attack graph modeled with SysML-Sec for the verification of a system. The issue of how to create such an attack graph is not addressed, in other words, the attack scenarios are not automatically generated and need to be manually fed to the tool. Cheah et al. [27], present an approach to systematic security evaluation and testing of vehicular systems using attack tree. In their approach, the attack trees are also considered as an input of the evaluation engine.

3.2.2.2 Attack tree generation in other application domains

In contrast, automated generation of attack trees has been addressed in other domains, especially in network security and enterprise security [15, 57, 67, 106].

In [106] Phillips et al. build an attack graph based on topology and vulnerability information whose goal is the analyze network vulnerabilities, they also use the attack graph to identify attack paths with high probability or low cost. The analysis system requires as input a database of common attacks, broken into atomic steps, specific network configuration, and topology information as well as an attacker profile. In [110] Ritchey et al. used a model checker to provide single attack scenarios to depict vulnerabilities due to the configuration of various hosts in a network. The pieces of information about the network are fed to the model checker and then assert that an attacker cannot acquire a given privilege on a given host. The model checker provides a counterexample (the attack steps) in case the assertion is false. As an extension of this work, in [117] Sheyner et al. present an automated method to analyze a network of hosts with known vulnerabilities and produce an attack graph that depicts *all* possible ways for the attacker to reach his goal.

Later works focused on reducing the complexity of the approaches. In [15] Ammann et al. propose a scalable attack graph generation based on the monotonicity assumption (an exploit never invalidates another exploit). In [103] Ou et al. introduced a logic-based approach for network security analysis. The method relies on inference rules implemented on a modified version of the XSB inference engine to depict *all* attack paths combining vulnerabilities in a network. In [57] Ingols et al. use network configuration data to compute network reachability automatically, classify vulnerabilities and builds an attack graph used to recommend actions

to improve network security. In [67] Jajodia et al. use topological information to analyze vulnerability dependencies and assess the impact of individual and combined vulnerabilities on overall security, then identify critical vulnerabilities and provide strategies for protection.

The problem has also been investigated for enterprise security domain [36, 66, 83] also called socio-technical physical system. The goal is to implement enterprise security policy against possible “insider attack” or attacks that leverage certain “trust” relations and social interactions between actors (employees). Thus efforts focused on modeling trust relations and asset mobility. In [66] Ivanova et al. present a general framework of a model for enterprise security and how to transform this model to an attack tree that exploits possible trust relations between actors. In [36] only the modeling aspect of the problem is discussed and also focuses on trust relations and asset mobility. In [83] Lenzini et al. investigate a formal model to evaluate the security of physical systems with objects as assets and people as agents. Their model can detect and quantify attacks with associated probability and cost.

Generation techniques used in the previously mentioned papers rely on models that are not suited for the automotive domain. However, the general approach could be adapted. This approach is more or less the same for all of the presented works: first, the real system is abstracted in a model that captures only the essential aspects. Second, the modeled system is expressed using the language of an inference engine (Model checker, Horn clauses ...). It is then processed by the inference engine whose output is a set of possible attacks to be analyzed. The work presented in this chapter has been partly inspired by the previously mentioned frameworks.

3.3 A case study: speed acquisition and display system

In this section, we introduce a case-study on which the methodology has been applied. The case study will be mentioned at each step of the methodology throughout the chapter in order to clarify based on an example.

3.3.1 Description

To clarify the model let us consider the case-study of Figure 3.3. In this case study, we propose the *speed acquisition and display system* implemented on an architecture composed of 3 ECUs connected to the same CAN-bus. The CAN-bus is connected to an OBD port for diagnostics purposes. The first ECU (denoted ECU_1) is equipped with a CAN controller that allows it to communicate over the CAN-bus, a speed sensor and a processing unit on top of which runs a service whose role is to make the acquisition from the sensor and send the information to other ECUs on the CAN-bus. The second ECU (denoted ECU_2) is also equipped with a CAN controller, a processing unit and a human-machine interface (denoted screen). On the ECU_2 runs a service whose job is to read the speed information from the CAN-bus and to pass it to the screen to be displayed for the driver. ECU_2 is the instrument cluster. Finally, the third ECU (ECU_3) similarly is equipped with a CAN controller in order to be able to communicate over the CAN-bus. Additionally, it is equipped with a cellular network controller that enables communications with the outside world. ECU_3 also has a processing unit on top of which

3. RISK ANALYSIS AND ATTACK TREE GENERATION

runs a diagnostics service whose role is to receive diagnostics commands from the cloud that need to be relayed to the CAN-network and sends back diagnostics responses.

3.3.2 Goal of the attacker: forge displayed vehicle speed

For this particular example, we focus on studying the ability of the attacker to inject an *erronous speed* measurement into the system. Obviously, such an attack will have a substantial safety impact, as the speed of the vehicle is used by many other functions (like the display function for the user) including safety critical functions. We focus in this chapter on the single user-display aspect. Other functions can be studied similarly.

3.4 Cyber-physical architecture formal model

In this section, we present the formal model that we will use in order to define main components of the system, and that will be used to automatically generate attack paths first in the form of an attack graph, and that will be transformed into an attack tree. The formal framework will serve to formally define an abstract representation of the system (and its behavior) as well as the attacker. The latter will be defined with a behavioral model composed of a set of actions (interpreted as basic attacks) that can be guarded by contextual conditions. Finally, the framework is implemented with means of *graph transformation system*.

As introduced in chapter 2, the cyber-physical architecture of the vehicle is composed of four main components that are ECUs, *communication buses*, *sensors*, and *actuators*. The ECUs are composed of electronic components that implement *storage capacity*, *communication interfaces* to the outside world as well as processing capabilities. ECUs also run *software* that implements multiple *services* whose role is to process *data* produced by the interfaces and necessary for the proper functioning of the vehicle. In this section, we present the abstracted formal model that we define for each of the aforementioned elements.

A formal model of a cyber-physical architecture of a connected vehicle is a tuple $\langle Comm, Hardware, Service, Data, Arch, Att \rangle$, where: “*Comm*” models the communication mediums inside the vehicle as well as to the outside world. “*Hardware*” models the hardware components that are physically accessible by an attacker. “*Service*” models the services that are implemented inside the ECUs. “*Data*” models the data components that are exchanged between services and ECUs. “*Arch*” models the structure of the architecture, i.e., the relations that exist between different entities and finally, “*Att*” models the attacker.

3.4.1 Data

Data elements constitute the set of data being communicated and processed between other components of the cyber-physical architecture of the vehicle. They are produced by sensors and communication interfaces, processed by services and consumed by actuators. They can be transported by communication mediums and stored by hardware elements (memories in particular). Formally, data elements *Data* is a tuple $\langle D, Type_D, Modified_D, Class_D \rangle$, where:

- D is a finite set of data elements.
- $Type_D : D \rightarrow T_D$ return the type of the data in a set of predefined data types (T_D).
- $Modified_D$ is a predicate (True/False) that specifies if the data itself has been modified by an attacker.
- $Class_D$ is an attribute that captures the data class depending to its importance. The class of the data is in $\{Public, Restricted, Sensitive, Critical\}$.

3.4.2 Communication mediums

Communication mediums ($Comm$) are very important in order to assure data exchange between at least two components, and they can be shared, or point to point communication mediums. We can find multiple communication mediums used inside the vehicle whether they are used to implement communication inside a single ECU, between ECUs or even between the vehicle and the outside world. Examples include CAN, FlexRay, USB, I2C, UART, WiFi, Bluetooth, Cellular ... Formally, $Comm$ is a tuple $\langle C, CommType, Accessibility \rangle$ where:

- C is a finite set of communication mediums.
- $CommType$ is an attribute that captures the type (name) of the communication medium. Typically CAN, FlexRay, USB, ... The $CommType$ attribute plays an essential role in the interpretation of the attack as it captures the involved access vectors.
- $Accessibility$ is an attribute that captures the accessibility range of the communication medium. Typically in the set $\{ \text{Long-range access, Short-range access, Physical access} \}$. The $Accessibility$ attribute plays an essential role in the assessment of the attack as it captures a component of the attack potential. This accessibility attribute expresses in some sens the window of opportunity from the attacker point of view.

3.4.3 Hardware components

Hardware components represent an essential part any ECU. They are used to model sensors, actuators, storage capacity (memory), communication controllers, etcetera. A particular type of hardware components are sensors that can produce data (i.e. create new data elements), and actuators that can consume data (i.e. delete data elements). Hardware components can be accessed from software components and help data flow between multiple components. Formally, hardware components $Hardware$ is a tuple $\langle H, \Sigma_H, b_H \rangle$, where:

- H is a finite set of hardware components.
- $HardwareType$ is an attribute that captures the type (name) of the hardware component. Typically it could be CAN-Controller, Cellular-Controller, USB-Controller, Memory, Sensor ...
- Σ_H is a finite set of basic hardware actions defined in order to describe the behavior of some of the hardware component with regards to the data. The finite set of basic

3. RISK ANALYSIS AND ATTACK TREE GENERATION

hardware actions is defined in (3.1).

$$\begin{aligned} \Sigma_H = \{ & \\ & ReadFromComm(h, c, d), \\ & WriteToComm(h, c, d), \\ & New(d), Delete(d) \\ & h \in H, c \in C, d \in D \} \end{aligned} \quad (3.1)$$

- $b_H : H \rightarrow \mathcal{L}$ returns the expression that describes the behavior of the hardware component.

Remark 1. *ReadFromComm(h, c, d) is a behavioral rule that allows a data d that is initially located on a communication medium c to be read by the hardware component h after which it changes location. WriteToComm(h, c, d) is the opposite behavioral rule. These two behavioral rules could be conditioned by the type of hardware component, the type of data or other predicates. New(d) is a behavioral rule that is specific to “sensor” that serves to model the fact that sensors produce data. However, the number of data component is bounded which makes the use of this behavioral rule limited. Delete(d) is a behavioral rule that is specific to “actuators” that serve to model the fact that actuators consume data.*

3.4.4 Service components

Service components are elements that describe the software running on top of an ECU. They are used to model the “processing” done on data components. The service layer is modeled with a *data flow diagram* that consists of “processes” (also called services) and data stores. Each process has input data and produces output data. Informally, a service is a process that transforms “input data” into “output data”. More formally we can write:

$$d_1^{in}, d_2^{in}, d_3^{in}, \dots \xrightarrow{s} d_1^{out}, d_2^{out}, d_3^{out}, \dots$$

Where $\{d_1^{in}, d_2^{in}, d_3^{in}, \dots\}$ is the input data set, $\{d_1^{out}, d_2^{out}, d_3^{out}, \dots\}$ is the output data set and the connective \xrightarrow{s} is read : “service s is used to produce”. This paradigm is very convenient to model software components [11] in an abstract way as it can adapt to different stages of software engineering through top-down data flow diagram process decomposition. Additionally, a service component has access to hardware components (for instance memory to store data, communication controllers to send/receive data, ...), expressed in terms of "read" and "write" access rights.

Formally, service components is a tuple $\langle S, Inputs, Outputs, \Sigma_S, b_S \rangle$, where:

- S is a finite set of service components.
- $Inputs(s), (s \in S)$ is a set of input data components types that are used to produce the outputs.
- $Outputs(s), (s \in S)$ is a set of output data components types produced by the service.
- v is an attribute vulnerability to indicate whether there is a vulnerability in the service (optionally this can be expressed as an estimated probability of the presence of a vulnerability [112]).

- Σ_S is a finite set of basic service actions defined in order to describe the behavior of the service component with regards to the data and the hardware. The finite set of basic service actions is defined in (3.2).

$$\begin{aligned} \Sigma_S = \{ & \\ & ReadFromHw(s, h, d), \\ & WriteToHw(s, h, d), \\ & Process(d) \\ & s \in S, h \in H, d \in D \} \end{aligned} \quad (3.2)$$

- $b_S : S \rightarrow \mathcal{L}$ returns the expression that describes the behavior of the service component.

Remark 2. *ReadFromHw(s, h, d) is a behavioral rule that allows a service $s \in S$ to read a data $d \in D$ that is initially located on a hardware component $h \in H$. It is designed in order for the service to collect the set of its input data. WriteToHw(s, h, d) is the opposite behavioral rule that allows the service $s \in S$ to write to the hardware component $h \in H$ the output data $d \in D$. These two behavioral rule could be conditioned by the type of hardware component, the type of data or other predicates. Process(d) is a behavioral rule that is specific to each service and transforms the data inputs into data outputs by deleting the input data elements and creating a new output data elements. Types of data inputs and data outputs is not the same.*

3.4.5 Attacker

The attacker “Att”, is an outsider agent whose goal is to perform an attack. Informally, the attacker in our approach is modeled as a set of behavioral rules (actions or basic attacks) that sets up the level of the attacker (regarding expertise), as well as a set of knowledge (regarding data components). This particular set of rules is a hypothesis on the intruder model and can be replaced by any stronger or weaker model. The attacker can nevertheless expand its knowledge database while performing the attack. The architecture of the system is assumed to be known to the attacker. More formally, an attacker is a tuple $\langle Att, \Sigma_{Att}, b_{Att} \rangle$, where:

- Att is a component that models the attacker.
- Σ_{Att} is a finite set of basic attacks defined in order to describe the behavior of the

3. RISK ANALYSIS AND ATTACK TREE GENERATION

attacker with regards to the system. The finite set of basic attacks is defined in (3.3).

$$\begin{aligned} \Sigma_S = \{ & \\ & ConnectToHw(h), \\ & WriteToHw(h, d), \\ & ReadFromHw(h, d), \\ & ConnectToComm(c), \\ & WriteToComm(c, d), \\ & ReadFromComm(c, d), \\ & Exploit(s), \\ & ExploitRead(s, d), \\ & ExploitWrite(s, d), \\ & Modify(d) \\ & s \in S, h \in H, d \in D \} \end{aligned} \tag{3.3}$$

- $b_{Att} : Att \rightarrow \mathcal{L}_A$ returns the expression that describes the behavior of the attacker.

Remark 3. In order to better account for the attacker's actions and impact on the cyber-physical system, we impose to all the “process” behavioral rules associated with services to be preserving with regards to the “modified” predicate. That is to say that if a service's input data has been “modified” by the attacker, the output data will also have a “modified” predicate. *Example:* consider a service s_1 that takes as input two data elements d_1, d_2 that are processed in order to output a third data element d_3 .

$$(d_1, d_2 \xrightarrow{process(s_1)} d_3)$$

If d_1 has been modified (d_1 or d_2 or both) by the attacker ($d_1(modified = true)$), then the modifier predicate will also be true for the output data d_3 ($d_3(modified = true)$).

3.4.6 Architecture

The architecture “*Arch*” describes the relations between the previously introduced components. It links hardware controllers to communication mediums, services to hardware components, and data to communication, hardware and service components. Formally, *Arch* is a labeled graph $G = \langle V, E, l_V, l_E \rangle$ where:

- V is the set of labeled vertices: $V = D \cup C \cup H \cup S \cup Att$.
- E is the set of labeled edges between vertices:

$$E \subseteq (H \times C) \cup (S \times H) \cup (D \times S) \cup (D \times H) \cup (D \times C) \cup (D \times Att) \cup (Att \times C) \cup (Att \times H) \cup (Att \times S)$$

- $l_V : V \rightarrow \{Data, Comm, Hardware, Service, Attacker\}$ is a vertices labeling function that assigns a label to each vertex $v \in V$ corresponding to its type.

- $l_E : E \rightarrow \{r, w, rw, connected, located, stored\}$ is an edge labeling function that assigns a label to each edge depending on the type of nodes connected by the edge. Informally, for each service component $s \in S$ that has a *read* and/or *write* access on a hardware component $h \in H$ a label is assigned to the edge between s and h whose label is in $\{r, w, rw\}$. For each hardware component $h \in H$ connected to a communication medium $c \in C$, a label “*connected*” is assigned to the edge between h and c . The evolution of the attacker “*Att*” inside the system is tracked with edges from the *Att* component to the communication/hardware/service component reached by the attacker. These edges are labeled with the label *located*. Finally, a data component $d \in D$ can be stored on a communication/hardware/service component. Thus a label “*stored*” is assigned to the edge between the data component and the communication/hardware/service node. Additionally the attacker “*Att*” also can store data.

- $Write : S \times H \rightarrow \{True, False\}$ is a predicate function that indicates if a service component $s \in S$ has a write access to a hardware component $h \in H$.

$$\forall (s, h) \in (S \times H), Write(s, h) = True \iff (s, h) \in E \wedge (l_E((s, h)) = w \vee l_E((s, h)) = rw)$$

- $Read : S \times H \rightarrow \{True, False\}$ is a predicate function that indicates if a service component $s \in S$ has a read access to a hardware component $h \in H$.

$$\forall (s, h) \in (S \times H), read(s, h) = True \iff (s, h) \in E \wedge (l_E((s, h)) = r \vee l_E((s, h)) = rw)$$

- $Connect : H \times C \rightarrow \{True, False\}$ is a predicate function that indicates if a hardware component $h \in H$ is connected to the communication medium component $c \in C$.

$$\forall (h, c) \in (H \times C), Connect(h, c) = True \iff (h, c) \in E \wedge l_E((h, c)) = connected$$

- $Locate : Att \times (C \cup H \cup S) \rightarrow \{True, False\}$ is a predicate function that indicates if the attacker *Att* has reached a communication/hardware/service component.

$$\forall (a, x) \in (Att \times (C \cup H \cup S)), Locate(a, x) = True \iff (a, x) \in E \wedge l_E((a, x)) = located$$

- $Store : D \times (C \cup H \cup S \cup Att) \rightarrow \{True, False\}$ is a predicate function that indicates if a data component $d \in D$ is stored on a communication/hardware/service/attacker component.

$$\forall (d, x) \in (D \times (C \cup H \cup S \cup Att)), Store(d, x) = True \iff (d, x) \in E \wedge l_E((d, x)) = stored$$

The language, \mathcal{L} , of a hardware and service component behavioral expression is generated by the grammar expressed in (3.4).

$$l ::= end \mid \beta.l \mid l +_g l, \text{ where: } \beta \in \Sigma_S \cup \Sigma_H \quad (3.4)$$

Similarly, the language, \mathcal{L}_A , of the attacker behavioral expression is generated by the grammar expressed in (3.5).

$$l ::= end \mid \beta.l \mid l +_g l, \text{ where: } \beta \in \Sigma_{Att} \quad (3.5)$$

3. RISK ANALYSIS AND ATTACK TREE GENERATION

The term "end" expresses inaction, the operator $< . >$ is the sequential composition, and the operator $< +_g >$ is guarded composition. The guard g is a contextual condition, questioning whether a hardware component is connected to a communication medium, whether a data is of the right type Formally, a guard is an expression $< e >$ in the propositional logic language expressed in (3.6).

$$e ::= \text{True} \mid p \mid \neg e \mid e \wedge e \quad (3.6)$$

$$\begin{aligned} p ::= & \text{Type}_D(d) = t \mid \text{Write}(s, h) \mid \text{Read}(s, h) \mid \text{Connect}(h, c) \mid \text{Locate}(a, c) \mid \\ & \text{Locate}(a, h) \mid \text{Locate}(a, s) \mid \text{Store}(d, a) \mid \text{Store}(d, c) \mid \text{Store}(d, h) \mid \\ & \text{Store}(d, s) \\ & \{s \in S, h \in H, d \in D, a \in \text{Att}, c \in C \cup H \cup S \cup D \cup \text{Att}, t \in T_D\} \end{aligned} \quad (3.7)$$

3.4.7 Security properties

Along with the attacker and system formal models, we also formalize the attacker goal (that can also be interpreted as a security property of the system) as a condition or an expression in propositional logic language also expressed with (3.6). The goal is then to check the satisfiability of this security property in the considered model in order to generate the attacker basic actions that allowed it. The security properties that can be expressed by the introduced formal model can be summarized in the following way:

- Check if the attacker can extract information from the cyber-physical architecture. It is particularly helpful to trace the actions that allow the attacker to obtain critical or restricted information which constitutes a *confidentiality/privacy* violation. This property can be expressed as follows:

$$\text{Store}(d(\text{Class}_D = \text{critical/restricted}), a) = \text{True};$$

- Check if the attacker is able to modify a data in the system. It is particularly insightful to trace the actions that allow the attacker to influence the behavior of an actuator which constitutes an *integrity* violation. This property can be expressed as follows:

$$\text{Store}(d(\text{modified} = \text{True}), h) = \text{True};$$

- Check if the attacker is able to exploit a service in the system. This is helpful in order to determine what are the most important. This property can also be interpreted as an availability issue in the sense that if an attacker can somehow take control of a service, the availability of the service is no-longer guaranteed.

$$\text{Locate}(a, s) = \text{True}$$

- Check if the attacker is able gain a read/write access rights to a hardware component.

$$[\text{Locate}(a, s) = \text{True}] \wedge [\text{write}(s, h) = \text{True}]$$

$$[\text{Locate}(a, s) = \text{True}] \wedge [\text{read}(s, h) = \text{True}]$$

3.4.8 Case-study formal model

Coming back to the case-study introduced in section 3.3, we can apply the formal model in order to define an abstract view of the system in the following way:

- In data elements, we define a data components one of type “speed”.

$$D = \{(d_1, type = speed, modified = False, Class = Sensitive)\} \quad (3.8)$$

- In the communication mediums, we define two components: a CAN-bus and a cellular network.

$$Comm = \{(Commtype = CAN, Accessibility = Physical-access), \\ (CommType = Cellular, Accessibility = Long-range)\} \quad (3.9)$$

- In the hardware set, we define four types of hardware components: 3 CAN-Controller (one for each ECU), a cellular network controller (for ECU_3), a screen-hardware (for ECU_2) and a speed-sensor (for ECU_1).
- In the service components, we define three elements denoted S_1 , S_2 and S_3 .
 - S_1 is the service that runs on top of ECU_1 and is responsible for the speed acquisition from the speed-sensor, it has a "read" access to the speed-sensor hardware, and a "write" access to the CAN-controller.
 - S_2 is the service that runs on top of ECU_2 and is responsible for reading the speed information from the CAN-controller and repay it to the screen-hardware to be displayed for the user. Thus it has a "read" access to the CAN-controller, and a "write" access on the screen-hardware.
 - S_3 is the service that runs on top of ECU_3 and is responsible for relaying a diagnostics command from the cellular network to the CAN-bus and responses from the CAN-bus to the cellular network. Thus it has a read and write (rw) access on the cellular network controller and read and write (rw) access to the CAN-controller. Additionally, we suppose that this service has a vulnerability.

3. RISK ANALYSIS AND ATTACK TREE GENERATION

- The architecture of the overall system can be defined by the following predicates:

$$\begin{aligned} \text{Read}(S_1, \text{speed-sensor}) &= \text{True}, \\ \text{Read}(S_2, \text{HW-CAN-2}) &= \text{True}, \\ \text{Read}(S_3, \text{HW-CAN-3}) &= \text{True}, \\ \text{Read}(S_3, \text{HW-cellular-1}) &= \text{True}, \\ \text{Write}(S_1, \text{HW-CAN-1}) &= \text{True} \\ \text{Write}(S_2, \text{HW-screen}) &= \text{True}, \\ \text{Write}(S_3, \text{HW-CAN-3}) &= \text{True}, \\ \text{Write}(S_3, \text{HW-cellular-1}) &= \text{True}, \\ \text{Connect}(\text{HW-CAN-1}, \text{CAN-bus}) &= \text{True}, \\ \text{Connect}(\text{HW-CAN-2}, \text{CAN-bus}) &= \text{True}, \\ \text{Connect}(\text{HW-CAN-3}, \text{CAN-bus}) &= \text{True}, \\ \text{Connect}(\text{HW-cellular-1}, \text{cellular}) &= \text{True}, \\ \text{Store}(d_1, \text{speed-sensor}) &= \text{True} \end{aligned} \tag{3.10}$$

Note that the set of predicates that defines the start architecture can change throughout the system evolution with the application of behavioral rules. In particular, the location of the attacker can be defined as the attacker interacts with the defined system and data can move between service, hardware and communication components.

- In this example, we adopt an attacker model that has the ability to connect to the CAN-bus, the cellular network and read and write on them, to modify data and to exploit vulnerabilities of the services.
- The goal of the attacker is to forge an erroneous vehicle speed to be displayed to the driver.

$$\text{Store}((d, \text{type} = \text{speed}, \text{modified} = \text{True}), \text{HW-screen}) = \text{True}$$

- The behavioral rules implemented for this case-study as well as the behavioral rules of basic attacks are exposed in this thesis Appendix.

3.5 Graph transformation system

In this section, we briefly introduce "graph transformation system" (GTS) as a rule-based modeling approach that allows capturing the structural as well as behavioral aspects of a system. We use it as the underlying formal modeling language supporting the methodology.

3.5.1 Definition

Graphs are helpful data structures that are capable of capturing a broad range of systems. If a system consists of entities and relations between them, it can be easily represented as a graph in which vertices model the entities and vertices model the relations. Dynamic configurations

or states of a graph can model the dynamic behavior of the system. In such systems, entities and relations can evolve, new relations and entities can be created or deleted. These evolutions are captured with "transformations" operated on the graph. A graph transformation system is a formal approach for structural modifications of graphs via the application of transformation rules. It is thus a tuple (G, R) where G is a graph and R is a set of transformation rules. A typed GTS is a GTS where each element of the graph is assigned a type. Transformation rules are then type-preserving. A graph transformation rule TR is a modification that transforms a host graph G_{host} into a result graph G_{result} by adding or deleting nodes and edges. It consists of:

1. A left-hand side graph L specifying a pattern that needs to be present in the host graph in order for the rule to be applicable.
2. A Negative Application Condition (NAC) specifying a sub-pattern that must be absent from the host graph in order for the rule to be applicable.
3. A right-hand side graph R specifying how the resulting graph should be after the transformation, and
4. A mechanism specifying how to transform L into R when the NAC is satisfied. This mechanism determines the elements to be deleted from the host graph and the elements to be added to the resulting graph.

Graph transformation system has been used in multiple applications to model the dynamic behavior of systems [45, 51].

In our approach, we model mainly three types of transformation rules:

- Transformation rules to describe the normal behavior of the hardware components. For each type of hardware node, we define a behavioral model. For instance a *Memory* node used to store *Data*, accepts a *Data* node only from a service that has a write (w) access right on it. It also can transfer data only to a service that has a read (R) access right on it. These transformation rules model the set of basic hardware actions Σ_H .
- Transformation rules to describe the behavior of services (one or multiple rules for each service). These rules are conditioned by the availability of the input data. We assume that as soon as the input data are all available, the transformation rule can be triggered. The effect of this transformation rules will be to delete the input data (consumed by the service) and to create the output data with the correct output type and made available for other services. These transformation rules model the finite set of basic service actions Σ_S .
- Transformation rules to describe the attacker actions: the behavior of the attacker is modeled with transformation rules that represent basic attacks or actions that the attacker can perform on the system to interact with it. These transformation rules model the finite set of basic attacks Σ_{Att} .

3.5.2 Labeled transition system

The application of the transformation rule $\langle TR \rangle$ to the host graph $\langle G_{host} \rangle$ in order to produce the resulting graph $\langle G_{result} \rangle$ is a transition from a state of the system corresponding to $\langle G_{host} \rangle$ to a state modeled by $\langle G_{result} \rangle$. This transition is labeled with the applied transformation rule. Formally, we write the application of this transformation as in (3.11).

$$G_{host} \xrightarrow{TR(L,NAC,R)} G_{result} \quad (3.11)$$

The recursive application of all the modeled transformation rules to the start graph (start state) of the system produces a Labeled Transition System (LTS) $(\mathcal{G}, G_{start}, \xrightarrow{TR})$ where \mathcal{G} is the set of all graph states (or system states), $G_{start} \in \mathcal{G}$ is the start state and " \xrightarrow{TR} " is the labeled transition relation produced by the application of the transformation rules to the system graphs. The label assigned to the transition between two states is the action (hardware/service/attacker behavioral rule) being executed as well as the entities on which it operates. The defined LTS models the evolution of the system behavior as the result of the evolution of the behavior of its components and the attacker.

3.5.3 Tool support: GROOVE

In order to implement this modeling framework on example use-cases, we used a tool that supports graph transformations. GROOVE (GRaphs for Object-Oriented VERification) is an open source tool developed by the University of Twente [5]. It supports formal modeling using graphs and graph transformations along with a user-friendly and intuitive graphical interface. It is built to support specifications of graph grammars and application of transformation rules to graph models. However, the most attractive functionality of GROOVE is the recursive application of transformation rules to a start graph and all graphs generated through the application of these transformations. Thus, the simulator functions as a model checker, i.e. it generates the full state space of a given graph grammar [109]. It results in the generation of a state space, defined above as the LTS (Labeled Transition System). Additionally, the entire LTS is stored, which is an excellent source of information that can be of great value for conducting further analysis. In the next section, we will show how we can exploit the generated TLS in order to generate the attack tree.

GROOVE was previously used to model and explore multiple graph-based transformation systems case-studies [45]. In particular, the security analysis framework called *Portunes* for representing attack scenarios spanning through the physical, digital and social domain. [36]. Besides, the tool allows a flexible way to implement our proper graph grammar, specified in the previous section.

3.5.4 Case-study graph transformation system

Architectural graph: The architectural graph of the case-study introduced in section 3.3 is represented in Figure 3.3. In this architectural graph, we can see labeled vertices or node types (speed data, hardware CAN controllers, services, ...) and relations between them represented with labeled edges between vertices.

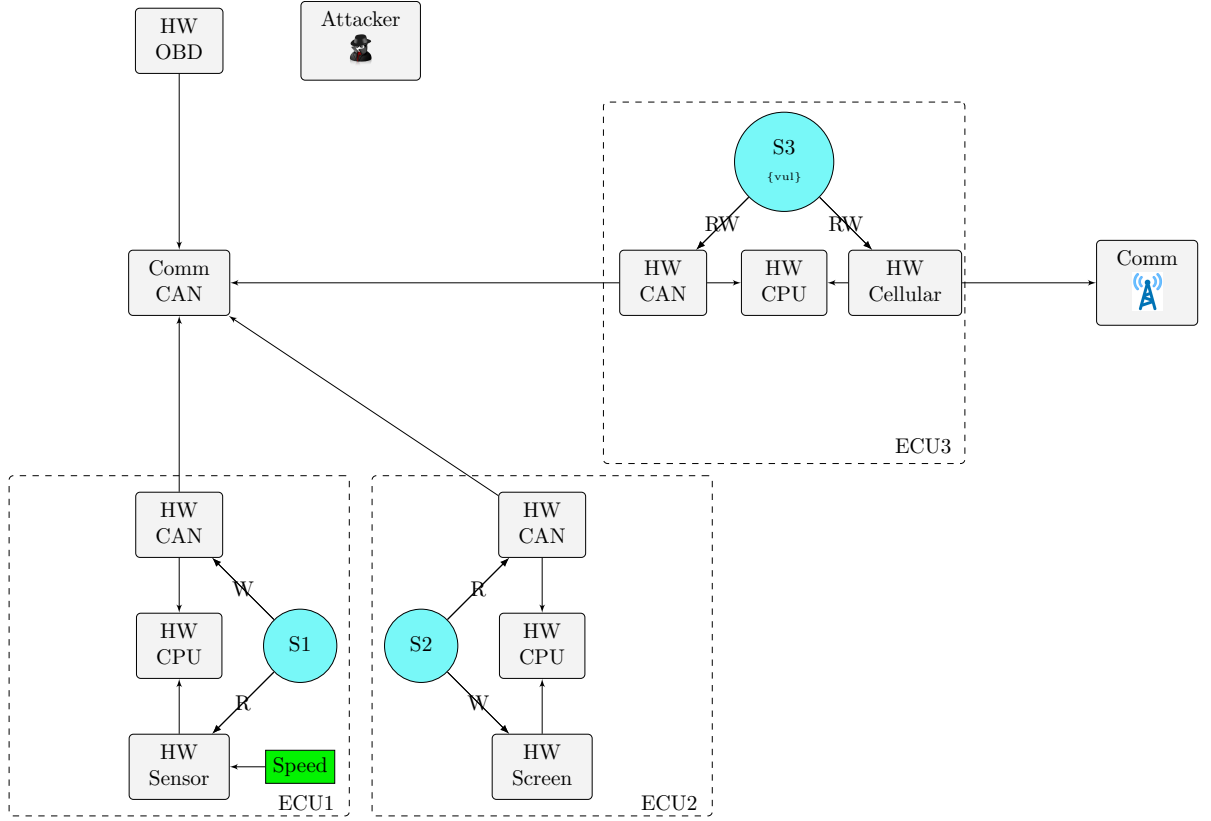


Figure 3.3: Architectural graph of the speed acquisition and display case-study

Transformation rules: As an example let us introduce transformation rules that we model for the architectural graph (Figure 3.3) of the example introduced above.

1. To describe the behaviour of the *Speed-acquisition* service (S_1), we implement the transformation rule of Figure 3.4. This rule means that if the speed data is available, the *speed-acquisition* service (S_1), will read a *speed* data from the *speed-sensor*.

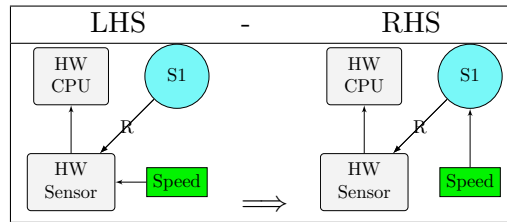


Figure 3.4: Speed acquisition rule (R1)

The transformation rule of Figure 3.5 means that is the speed data is available on the *speed-acquisition* service (S_1), and that service have the *write* access right to the CAN-hardware, then the service can send the data the CAN-hardware.

2. To describe the behavior of the CAN-hardware, we implement the transformation rule

3. RISK ANALYSIS AND ATTACK TREE GENERATION

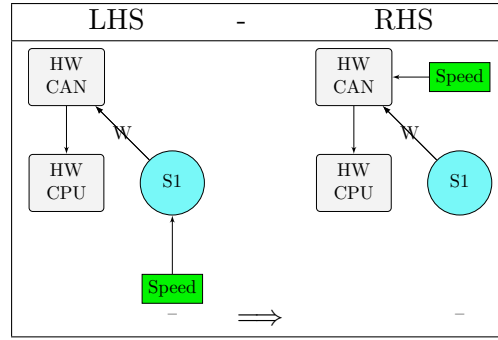


Figure 3.5: Speed send to CAN rule (R2)

of Figure 3.6. Note that at this stage it does not matter if the data is a speed data or not. This is mainly because this behavioral rule is designed to send any data on the CAN bus. The rule is also common to all CAN hardware as opposed to some rules which are sometimes specific to one in particular.

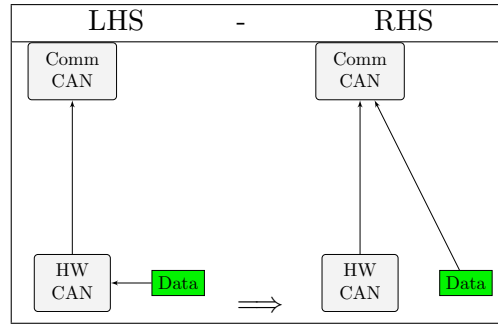


Figure 3.6: CAN send rule (R3)

3. To describe the behavior of the attacker, we model transformation rules that gives her the ability to connect to any communication link and to read and write data on that link. She is also able to exploit vulnerabilities and modify the collected data and replay it. The transformation rule (Figure 3.7) could be read as follows: if the attacker (Att) is connected-to the CAN-network, and if there is a Data packet transiting on the CAN network, then the attacker can copy the Data.

3.6 Attack tree transformation

3.6.1 Attack graph generation

Given a start graph (Figure 3.3 for instance), and a set of transformation rules (Figure 3.4, Figure 3.5, Figure 3.6, Figure 3.7, ...) , the recursive application of the transformation rules on the start graph will generate a state space which represents all possible states that could be generated from the set of transformation rules. In the state space, each state represents

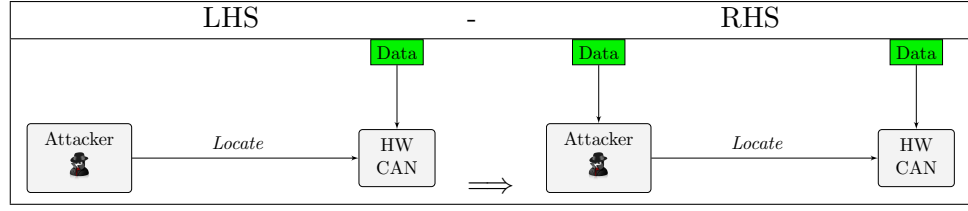


Figure 3.7: Example of attacker rule

a graph, and a transition between two states (source and destination) represents a rule application that allowed the transformation of the graph from the source state to match that of the destination state. For instance, the application of the transformations rules of Figure 3.4 and Figure 3.5 for the modeled example of Figure 3.3 will produce the state space of Figure 3.8.

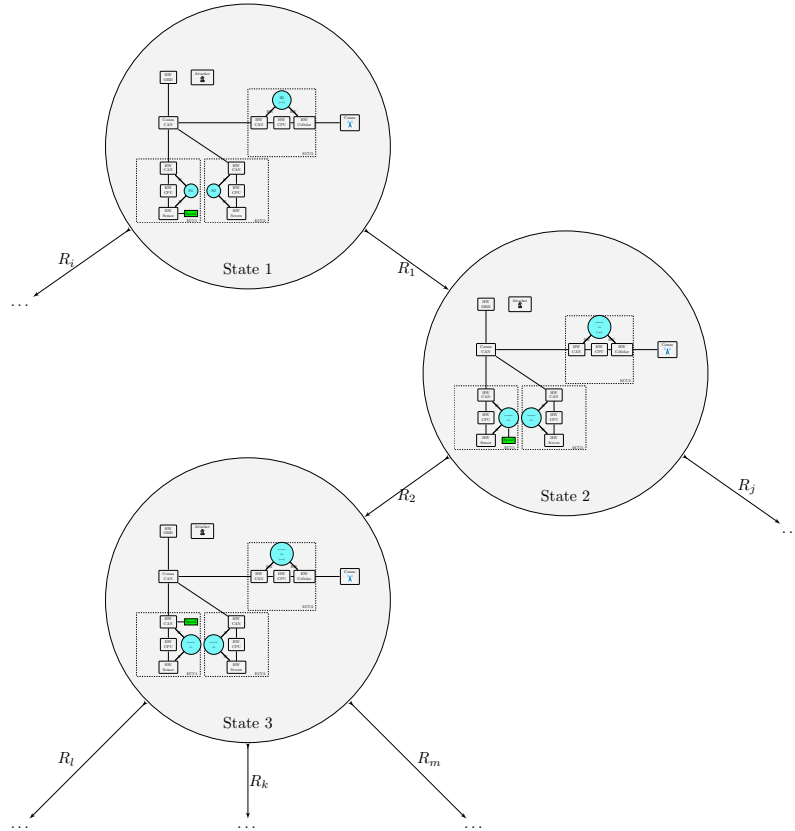


Figure 3.8: Application of transformation rules (state space exploration)

The modeled rules are a combination of attacker actions (or basic attacks) and rules that describe the behavior of the modeled elements. The produced state space contains transitions that model both attacker steps and element behavior. By definition of the attack graph, the state space contains the attack graph.

3. RISK ANALYSIS AND ATTACK TREE GENERATION

Given a particular attack scenario (attacker objective), we have to make a query to find states in the state space where the scenario is realized (the attacker has reached its objective). Thus queries allow detecting a vulnerable state. They are also expressed in the form of a graph. Figure 3.9 gives an example of a query that allows us to detect if there is a state of the system where the screen displays a *modified* speed. In practice, we modeled the architectural

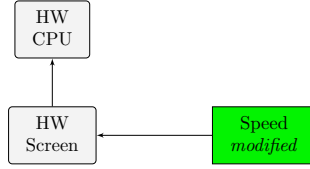


Figure 3.9: Query: False speed

graph and transformation rules using GROOVE [5]. The tool also allows the transformation of the input model and produces the associated LTS as the result of the recursive application of transformation rules. It also allows detecting states where the query (attacker objective) is true. In the next section, we will process this state space to capture only attacker actions in the form of an attack tree.

3.6.2 Attack tree generation:

The generated state space is quite complex and large. For instance, for the introduced case-study we obtained 768 states and 2860 transitions. Besides, it includes transitions that describe the behavior of the system as well as transitions that describe basic attacks performed by the attacker. The entire LTS contains too many information which makes it unusable in practice as "*too much information kills the information*". A convenient way to keep only the relevant information is to transform this state space into an attack tree that only captures attacker actions. The attack tree procedure will allow us to discard all the transitions that are not basic attacks and to output, in the form of a tree, a compact representation of complex attacks. Based on the attack graph (state space) generated in the previous section, in this section, we introduce two main transformations to this attack graph in order to produce the corresponding attack tree.

Let $G = (V_{LTS}, E_{LTS}, label)$ be the Labeled Transition System produced by the recursive application of transformation rules with a start state S_0 . This graph, as mentioned before, contains the attack graph. Let $L_V = \{S_v^1, S_v^2, \dots, S_v^n\}$ be the set of vulnerable states of the system (i.e. a state where a security breach has been detected). The security breach detected by that state is placed on the root of the attack tree. The goal is to collect all sequences of transformations (attacker actions only) that led to that state. Let \mathcal{A} be the set of labels corresponding to the attacker basic attacks. We explore the attack graph from the target state (S_v^i) backward to the start state. Each time we encounter a state with more than one incoming edges we place a [OR] node in the attack tree (meaning that there is more than one way to reach that state) and each time we encounter a state with only one incoming edge, we place an [AND] node in the attack tree. Finally each time we encounter a state with

more than one outgoing edges, we place a sub-tree and check if we already computed that sub-tree. Sub-trees are attack trees that are present more than once inside a single attack tree. Algorithm 1 gives the details of the operated transformation. In this algorithm, the function $card(L)$ returns the cardinality of a set L , the function $InputEdges(S)$ returns the input edges of the state $S \in V_{LTS}$, the function $Pred(S)$ return the predecessors of a state $S \in V_{LTS}$ Figure 3.10 gives examples of these transformations.

Remark 4. Note that the tree generation algorithm does not necessarily output the normal form of the tree. We can nevertheless use logical reduction techniques to output the normal form of the attack tree. In our implementation we used the python SymPy library [69] to simplify the attack tree to its disjunctive normal form.

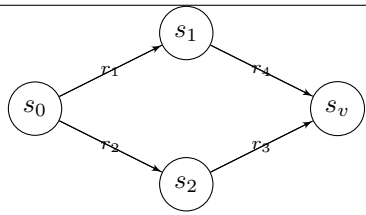
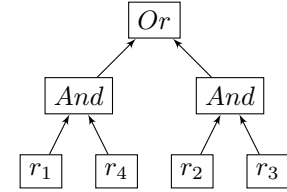
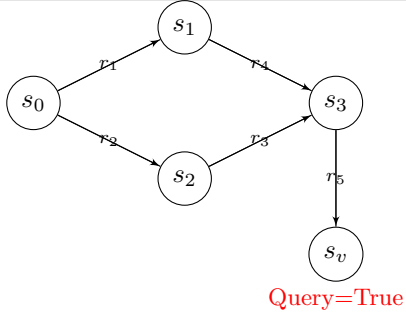
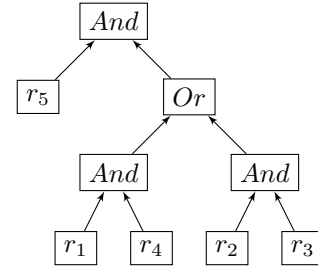
Attack Graph	Attack Tree
	
$p_1 = r_1 \wedge r_4$ $p_2 = r_2 \wedge r_3$	$Obj = p_1 \vee p_2$
	
$p_1 = r_1 \wedge r_4 \wedge r_5$ $p_2 = r_2 \wedge r_3 \wedge r_5$	$Obj = p_1 \vee p_2 = r_5 \wedge ((r_1 \wedge r_4) \vee (r_2 \wedge r_3))$

Figure 3.10: Examples of Attack graph to Attack tree transformation

3.6.3 Case-study generation of attacks

Using the example introduced in section 3.3 (Figure 3.3), we make a the query (cf. Figure 3.9) to detect vulnerable states where the attacker can force the system into a state where the screen displays a *modified* speed. The results produced by the methodology is shown in figure 3.11.

3. RISK ANALYSIS AND ATTACK TREE GENERATION

```

1 ConstructAttackTree ( $L$ );
   Input : L-List of states of the LTS  $\{S_v^1, S_v^2, \dots, S_v^n\}$ 
   Output: Attack - Tree
2 if  $\text{card}(L) > 1$  then
3    $n = \text{card}(L)$ ;
4   for  $i = 1$  to  $n$  do
5      $T_i = \text{ConstructAttackTree}(\{S_v^i\})$ ;
6   end
7   Return  $OR(T_1, T_2, \dots, T_n)$ ;
8 else
9   if  $\text{card}(L) == 0$  then
10    Return Null;
11  else
12     $I = \{I_1, I_2, \dots, I_d\} = \text{InputEdges}(L[1])$ ;
13    if  $\text{card}(I) == 0$  then
14      %comment :  $L[1] = S_0$  this is the start state
15      Return True;
16    else
17      if  $\text{card}(I) == 1$  then
18        if  $\text{label}(I) \in \mathcal{A}$  then
19          Return  $And(\text{label}(I), \text{ConstructAttackTree}(\{\text{Pred}(I)\}))$ 
20        else
21          Return  $\text{ConstructAttackTree}(\{\text{Pred}(I)\})$ 
22        end
23      else
24        for  $i = 1$  to  $d$  do
25          if  $\text{label}(I_i) \in \mathcal{A}$  then
26             $T_i = And(\text{label}(I_i), \text{ConstructAttackTree}(\{\text{Pred}(I_i)\}))$ ;
27          else
28             $T_i = \text{ConstructAttackTree}(\{\text{Pred}(I_i)\})$ ;
29          end
30        end
31        Return  $OR(T_1, T_2, \dots, T_d)$ 
32      end
33    end
34  end
35 end

```

Algorithm 1: Attack tree construction algorithm for constructing the attack tree starting from a list of vulnerable states

The attack tree shows that the analysis of the modeled architecture identified three attacks.

- Attack-1: where the attacker connects to the OBD port, eavesdrop on the CAN bus to

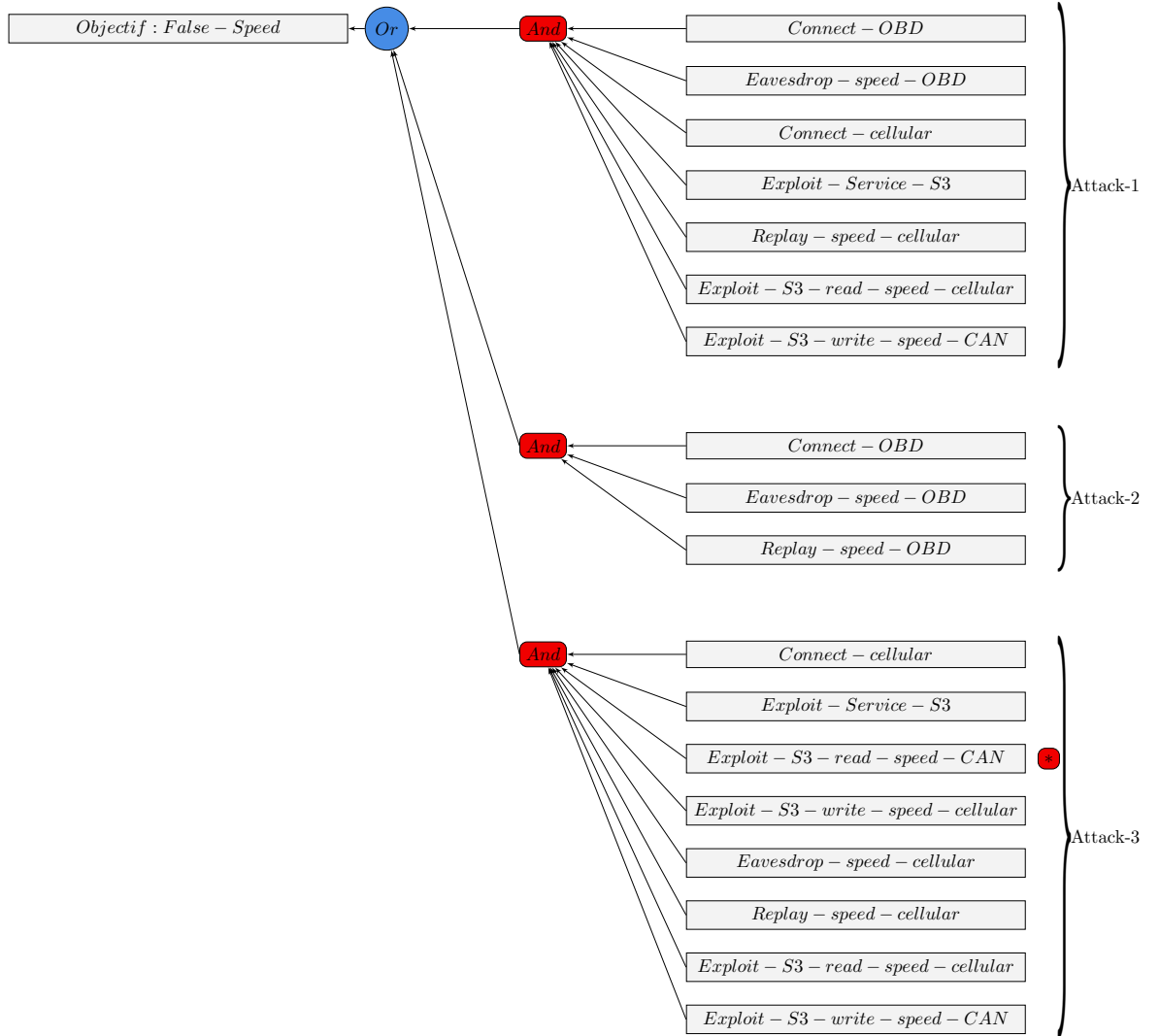


Figure 3.11: Attack tree automatically produced from the input model Figure 3.3

dump the Speed frame, then connects to the vehicle from the cellular network, exploits an exposed vulnerable service (S3), that has a "write" access on the CAN-controller then replays the data.

- Attack-2: where the attacker connects to the OBD port, eavesdrop on the CAN bus to dump the speed frame then replays it from the OBD port on the same CAN bus.

- **Attack-3:** where the attacker only operates from the cellular, she connects to the cellular, exploits the exposed service, and then uses that service to eavesdrop on the CAN hardware to dump and then replays the speed frame.

3.7 Security analysis and Countermeasure

3.7.1 Security analysis

As mentioned in the beginning of the chapter, attack trees do not only serve to represent security scenarios that contribute to the realization of a threat in a graphical way. They can also be used to quantify such scenarios with respect to a given parameter (called an attribute). Examples of attributes include the likelihood that the attacker's goal is satisfied, minimal time or cost of an attack. In this section, we perform a risk analysis of the attack objective based on the identified attacks and according to the EVITA risk analysis methodology [52], presented in table 2.3. During this step, the goal is to assign attributes to the leaf nodes (basic attacks) and then propagate it in a bottom-up fashion in order to compute an overall estimation of the attribute for the root node of the tree (i.e. the threat). The attribute propagation process described by Schneier in [114] is based on the definition of two functions (f_{And} and f_{Or}) one for the conjunction composition, the other for a disjunctive composition of nodes. These functions are used to compute (for each node type) the attribute value based on the attribute values of its child nodes. Mauw and Oostdijk [93] showed that if we want equivalent attack trees to yield the same estimated attribute for the overall threat, the algebraic structure $(D_{attribute}, f_{And}, f_{Or})$ should define a semiring, where $D_{attribute}$ is the attribute domain, f_{And} and f_{Or} are the two functions defined to evaluate respectively the conjunctive and disjunctive compositions. This result was generalized by Kordy et al. [77] to any semantics and attribute that satisfy a notion of compatibility. In the EVITA approach the estimated attribute is the attack potential (inverse of the attack probability or likelihood) estimated with a scoring system that includes multiple attack aspects that are: the *time*, the *expertise*, the *knowledge*, the *window of opportunity* and the *equipment* needed for the attack. Henniger et al. [52] used the $f_{OR} = Max$ of the attack probabilities attribute to propagate it through an *OR* node and the $f_{AND} = Min$ of the attack probabilities attribute to propagate it through an *AND* node.

Remark 5. *Note that it is easy to verify that the algebraic structure (\mathbb{R}, Min, Max) defines a semiring, which makes the estimated attack potential defined by the EVITA approach yields the same estimate for equivalent attack trees. When reasoning about the attack potential (inverse of attack probability), the operators are inverted i.e, the algebraic structure used to propagate the attack potential is the semiring defined (\mathbb{R}, Max, Min) .*

We identify a set of eleven basic attacks for which we estimate the attack potential, presented in the table 3.1. This table can be constructed from the beginning, before the generation of the attacks. It is independent of the architecture and can be considered as an input as well.

Table 3.1: Rating of the basic attacks

Basic Attacks		Attack potential				
		Time	Expertise	System knowledge	Window of opportunity	Equipment
a	Connect-OBd	\leq 1-day	Layman	Public	Moderate	Specialized
b	Connect-cellular	\leq 3-months	Multiple experts	Sensitive	Unlimited	Multiple bespoke
c	Eavesdrop-data-OBd	\leq 1-day	Expert	Public	Unlimited	Specialized
d	Eavesdrop-data-cellular	\leq 1-day	Layman	Restricted	Unlimited	Multiple bespoke
e	Replay-data-OBd	\leq 1-day	Expert	Restricted	Moderate	Specialized
f	Replay-data-cellular	\leq 1-day	Layman	Restricted	Unlimited	Multiple bespoke
g	Exploit-Service-S3	\leq 3-months	Multiple experts	Sensitive	Unlimited	Bespoke
h	Exploit-S3-read-data-CAN	\leq 1-day	Layman	Public	Unlimited	Standard
i	Exploit-S3-write-data-CAN	\leq 1-day	Layman	Public	Unlimited	Standard
j	Exploit-S3-read-data-cellular	\leq 1-day	Layman	Public	Unlimited	Standard
k	Exploit-S3-write-data-cellular	\leq 1-day	Layman	Public	Unlimited	Standard

3. RISK ANALYSIS AND ATTACK TREE GENERATION

We can now apply the estimate to the attack tree in order to estimate the attack potential of the identifier attacks. This estimate is presented in table 3.2. Of the three attacks, two (Attack-1 and Attack-3) were shown to be very unlikely attacks (with rating "beyond high" ($P=1$)). Those attacks require access to the vehicle from a cellular attack vector and exploitation of a vulnerable service (S3) which were considered to be complex steps and require special equipment. The other attack (Attack-2) requires only direct physical access which makes it easy to perform, but at the same time the window of opportunity of direct physical access is limited which makes the overall attack scenario probability rather "moderate" ($P=3$). The overall threat probability computed as the maximum of the attack scenarios probabilities is then "moderate". The impact of the threat is estimated independently from the attack scenarios and attack steps. In this particular case, we can consider that the threat can generate "Light or moderate injuries" (safety=1), with no specific privacy issues (Privacy=0), can generate "Low-level loss" (Financial=1) and has an impact on performance that the driver can perceive (Operational =2). The Impact vector (also called severity class in EVITA terminology) is $Impact = [1, 0, 1, 3]$. Additionally, the threat cannot be influenced by a human response (i.e., the controllability score $C = 4$). The combination of these ratings (c.f table 2.4) yields the Risk level presented in table 3.2.

Table 3.2: Estimates of attack potential of the identified attacks

Attack-steps	Aspects of Attack Potential					Attack Potential (Sum)	Attack Probability	Attack Scenario Probability	Threat Probability	Threat Impact	Associated Risk			
	Time	Expertise	System knowledge	Window of opportunity	Equipment									
Attack-1	a	0	0	4	4	8	5	Beyond High	Moderate (P=3)	[1,0,1,2]	[4,0,4,5]			
	c	0	6	0	0	4	10					4		
	b	10	8	0	9	34	1							
	g	10	8	0	7	32	1							
	f	0	0	0	9	12	4							
	j	0	0	0	0	0	5							
Attack-2	i	0	0	0	0	0	5	Moderate						
	a	0	0	4	4	8	5							
	c	0	6	0	0	4	10					4		
	e	0	6	3	4	17	3							
	b	10	8	7	0	9	34					1	Beyond High	
	g	10	8	7	0	7	32					1		
Attack-3	h	0	0	0	0	0	5	Beyond High						
	k	0	0	0	0	0	1							
	d	0	0	3	0	9	12					4		
	f	0	0	3	0	9	12					4		
	j	0	0	0	0	0	5							
	i	0	0	0	0	0	0					5		

3. RISK ANALYSIS AND ATTACK TREE GENERATION

Remark 6. Note that the obtained attacks are relative to an attacker model that has virtually no knowledge of the system (in terms of data) and is only able to connect to network interfaces. We can add to the attacker capability to connect to hardware components, in this case, we will be able to see that for instance, the attacker is also able to tamper with memory content and sensors.

Remark 7. We draw the reader's attention to the fact that there should have been four attacks. There is still a combination of basic attacks that was not returned by the attack tree generation algorithm. In fact, the fourth attack is where the attack can connect-cellular, exploit-Service-S3, Exploit-S3-read-speed-CAN, Exploit-S3-write-speed-cellular, Eavesdrop-speed-cellular, Connect-OBD, Replay-speed-OBD. This is mainly a technical issue, as the transformation rules associated to "Eavesdrop-speed-OBD" and "Eavesdrop-speed-cellular" is actually the same ("Eavesdrop-data-comm") that copies data from the communication medium to the attacker knowledge set. The labeled transition system returned by the GROOVE does not include the type of the node on which the rule is being instantiated.

The expected attack tree would thus be (with basic attack notations from table 3.1:

$$AT_{expected} = (a \wedge c \wedge b \wedge g \wedge f \wedge j \wedge k) \vee (a \wedge c \wedge e) \vee (b \wedge g \wedge h \wedge k \wedge c \wedge f \wedge j \wedge i) \vee (b \wedge g \wedge h \wedge k \wedge c \wedge a \wedge e)$$

We can see that when $c = d$, the simplification procedure would output the obtained attack tree:

$$AT_{obtained} = (a \wedge b \wedge c \wedge f \wedge g \wedge j \wedge k) \vee (a \wedge c \wedge e) \vee (b \wedge c \wedge f \wedge g \wedge h \wedge i \wedge j \wedge k)$$

We backtracked the traces generated by the state space exploration in order to understand that the basic attack "Eavesdrop-data-comm" (noted c) in the first term refers to an "Eavesdrop-speed-OBD" and that the basic attack "Eavesdrop-data-comm" (noted c) in the third term refers to an "Eavesdrop-speed-cellular"

3.7.2 Countermeasures:

In this section, we focus on attack-3 where the attacker only operated from the cellular network attack vector. This attack seems important as it does not require any physical access to the vehicle. Analyzing the steps of the attack, if we can prevent at least one of the basic attacks from happening, we can prevent the whole attack from happening as it is an [AND] node. It seems possible to either deploy a patch for the vulnerable service or to revoke the read access right of the service to the CAN controller. As a short term solution, one can opt for the second choice.

$$Read(S_3, HW-CAN-3) = False$$

Re-running the attack generation process with the modified architecture produces the attack graph presented in Figure 3.12 We can see that the newly generated attack tree contains only two attack options for the attacker since the service does not have a read access right to the CAN controller, the third attack scenario is no longer valid and hence not reported in the attack tree. The overall impact of the objective is affected by the architectural changes made, which were able to block an attack path.

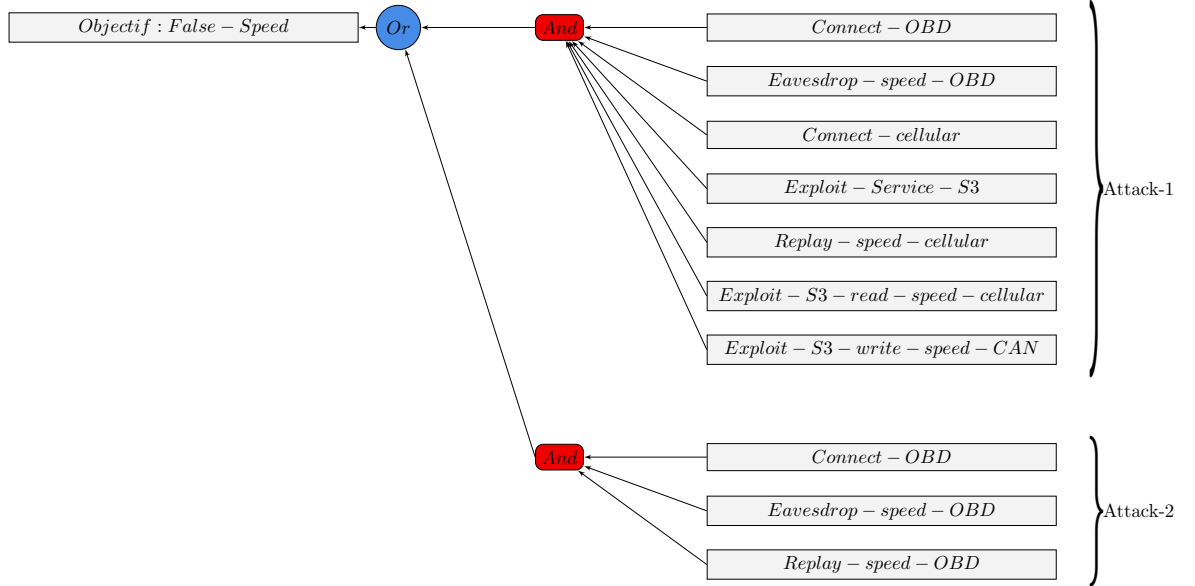


Figure 3.12: Attack tree automatically produced from Architecture-1

3.8 Conclusion

This chapter presents a modeling methodology using graph transformation system able to produce an attack tree based on vehicle architecture and an attacker model. This attack tree synthesizes all possible attack paths with respect to the input model. Figure 3.13 sketches the main steps of the overall approach.

The attack tree then serves as the basis for further analysis. Impact quantification and sensitivity analysis can be conducted given such attack tree whose goal is to improve the overall security of an automotive architecture during the design phase. The described methodology has nevertheless certain limitations due to required input data. In fact, these input data are:

- A structural and behavioral model of the services.
- A structural and behavioral model for hardware components.
- A behavioral model for the attacker as a set of basic attacks.

From the car manufacturer point of view, the fact that the modeling methodology requires architectural information of service nodes can be considered as a limitation as some software

3. RISK ANALYSIS AND ATTACK TREE GENERATION

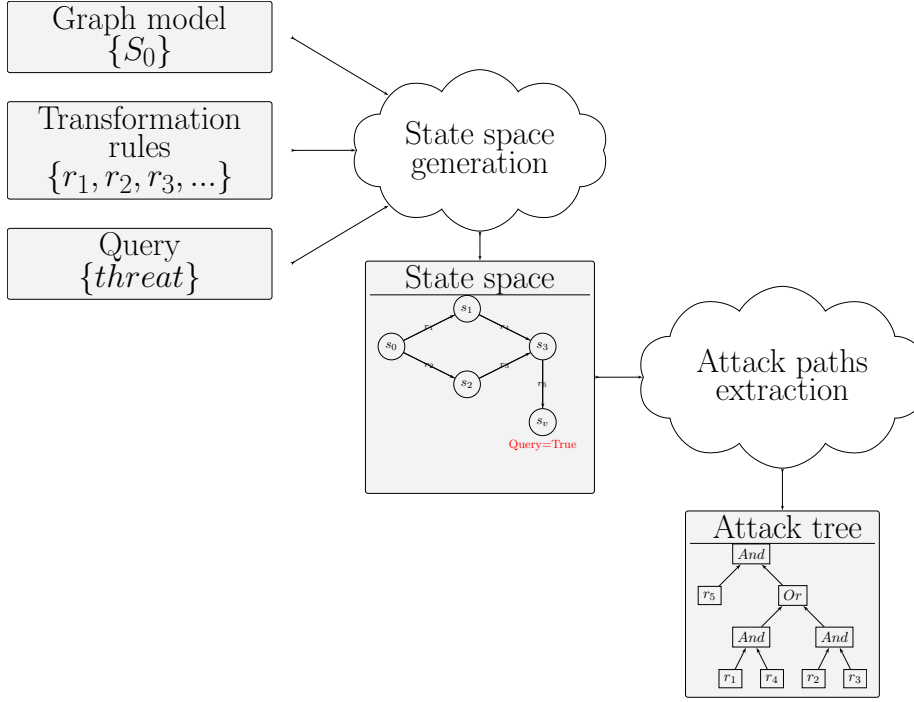


Figure 3.13: The overall approach to attack tree generation

architecture and implementation tasks are outsourced to other companies. However, certain architectural approximations based on functional blocks can still be used.

Furthermore, with regards to the attack potential analysis methodology, which is not automated, the formal model can incorporate further attributes and predicates in order to automate the assignment of attack potential ratings. For instance, in order to automatically rate the window of opportunity, we started to incorporate a *range-of-access* attribute to communication and hardware nodes, in order to estimate the expertise and time needed, we can think about adding complexity attributes to the vulnerabilities. Similarly, in order to estimate the needed equipment, we can add an attribute to communication and hardware nodes that specify in keywords the types of equipment needed by the attacker to connected to them. As a result, the estimation process can be improved and can be more precise. Another potential improvement would be to find a solution to dissociate, for a single attack, the attack preparation step (mainly reverse-engineering: connecting to the CAN-bus and eavesdropping on the speed frame or building an exploit for a vulnerable service), and the active deployment step where the attacker only launches the attack payload. This separation can also improve the attack potential estimation as the first step can be performed only once and later shared with less experimented attacker models.

Finally, from the presented use-case, we can notice that the access to the CAN-bus (i.e., gaining write and read access to it) is an essential step for all three attacks, without which none of the attacks would be possible. As a result, disabling this kind of access can effectively reduce the overall risk. In fact, in general, as in-vehicle communication buses orchestrate most of the communication inside the vehicle, virtually all the functionalities will at some

point process data that has been communicated through those buses. The security of these communication mediums constitutes a vital block of every security architecture towards safer cars. Thus in the next chapters, we will focus on developing mechanisms to secure in-vehicle networks.

3. RISK ANALYSIS AND ATTACK TREE GENERATION

Identifier Randomization: an Efficient Protection against CAN-bus Attacks

In this chapter, we investigate one of the prominent family of security protection mechanisms over the Controller Area Network, namely the identifier protection mechanisms. Results presented in this chapter has been scientifically valued by an article published as a book chapter in *Cyber-Physical Systems Security* [72].

Contents

4.1	Introduction	77
4.2	General formalism of ID-based protection	78
4.3	Evaluation metrics	80
4.4	Proposed solutions	81
4.5	Comparison	93
4.6	Conclusion	96

4.1 Introduction

In Chapter 2, we introduced state of the art protection mechanisms that can be applied to protect the Controller Area Network against possible attacks. One of these mechanisms is the *identifier protection* and can be used for an attacker model that has direct physical access to the CAN bus. Proposed solutions that use this concept, like the work of Han et al. [48, 49], or the work of Humayed et al. [56], prove that there are various ways of setting up protection mechanisms based on the identifier field.

Chapter contributions. In this chapter, we make the following contributions,

- First, we define the general formalism of the identifier based protection mechanisms, in particular, those based on a randomization strategy.

4. IDENTIFIER RANDOMIZATION: AN EFFICIENT PROTECTION AGAINST CAN-BUS ATTACKS

- Then, we introduce clear information-theoretic metrics in order to evaluate the efficiency of the applied protection from a security point of view.
- Third, we propose identifier randomization procedures to protect the CAN network from reverse-engineering, replay and injection attacks, both at software and hardware level. Amongst the proposed methods, some of them are proved to achieve optimal protection.
- Finally, we evaluate and compare the proposed methods and state of the art methods, based on the proposed metrics.

In what follows, section 4.2 introduces the general formalism of the identifier protection mechanism. Section 4.3 defines the evaluation metrics that measure the level of protection against known attacks. In section 4.4, we propose novel randomization strategies both at software and hardware level, to enhance the security of the CAN protocol. In section 4.5 we compare the randomization strategies with the state-of-the-art solutions according to the proposed evaluation metrics. Finally section 6 concludes the chapter.

4.2 General formalism of ID-based protection

This section identifies the main characteristics and constraints of the randomization function that has to be used for protection.

The way the CAN protocol is used today by car manufacturers is the following: each information that needs to be communicated from one ECU to the others is sent in a CAN frame. Each frame has a fixed identifier which is known to the sender and the receivers. This situation allows for a limited number of message identifiers to be used permanently throughout the vehicle life-cycle. The set of identifiers used can also be known to the attacker as it is communicated in clear on the CAN bus. They are fixed during the design phase of the vehicle and respond to priority criteria imposed by safety requirements. The priority level defines the criticality of the information and allows the CAN protocol to arbitrate between concurrent messages. Most of these messages have an update frequency (that also depends on the criticality of the information) that defines the probability of occurrence of the frame over the CAN network. Figure 4.1 illustrates a CAN communication network and the histogram of the identifiers that appear.

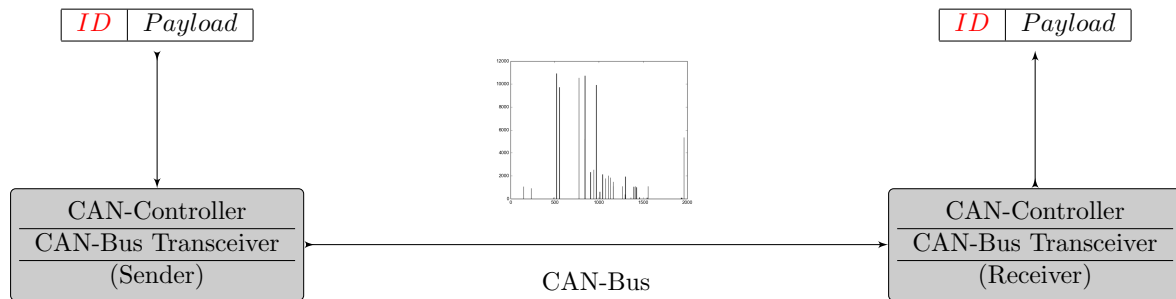


Figure 4.1: Controller Area Network with original identifier distribution

The fact that the same information is always sent over the same frame identifier enables the attacker to reverse the protocol and forge frames that can be accepted by the vehicle ECUs. The attacker first starts with a reverse-engineering step during which the goal is to identify the messages identifiers being used. Then builds an attack by injecting, or replaying, one or multiple CAN frames.

In order to protect the CAN network from such attacks, the idea is to change the message identifiers regularly. Optimally, every time the ECU needs to send information. This should be done in a way that the receiving ECUs can recover the original identifier and does not allow the attacker to reverse the protocol, or inject messages that can be accepted by other ECUs. To do so, an identifier randomization function F is added in such a way that it takes the original identifier ID and substitutes it with another identifier ID_r that changes at every occurrence m of a new frame on the CAN bus. The index m is the value of a message counter that has to be kept synchronized between the sender and the receivers. The function F can be written as in (4.1).

$$ID_r = F(ID, m) \quad (4.1)$$

At the receiver side, the ID is recovered by using the inverse function of F , F^{-1} , and the value m of the internal counter of the ECU as given by (4.2).

$$ID = F^{-1}(ID_r, m) \quad (4.2)$$

The randomization function F has to satisfy certain conditions in order to be useful in this context:

- First, F has to be injective and efficiently computable in order for the receiver to recover the original identifier rapidly. Figure 4.2 shows how this function could be integrated. We can see in this figure the expected histogram of randomized identifier ID_r that needs to be more spread compared to the one in Figure 4.1.
- Second, and for safety reasons, the function F has to be priority-preserving. This means that the priority of message identifiers ID_1 and ID_2 has to be the same as their randomized versions $F(ID_1)$ and $F(ID_2)$, respectively. This boils down to the condition expressed in (4.3).

$$ID_1 < ID_2 \Rightarrow F(ID_1, m) < F(ID_2, m) \quad (4.3)$$

- Third, the priority condition have to be preserved over time. Indeed, the message can go through a transmission buffer before being physically sent to the bus. Consequently, the state of every ECU counter m can be different from the real number of transactions counter on the physical layer. In order to be consistent, the randomization function has to guarantee that the identifiers keep their priority even if the transaction counter m is different. This is expressed by the following constraint (4.4).

$$ID_1 < ID_2, m_1 \neq m_2 \Rightarrow F(ID_1, m_1) < F(ID_2, m_2) \quad (4.4)$$

- Fourth, the output of the randomization function F has to be unpredictable. An attacker that has some information about previous outputs or identifiers should not be able to

4. IDENTIFIER RANDOMIZATION: AN EFFICIENT PROTECTION AGAINST CAN-BUS ATTACKS

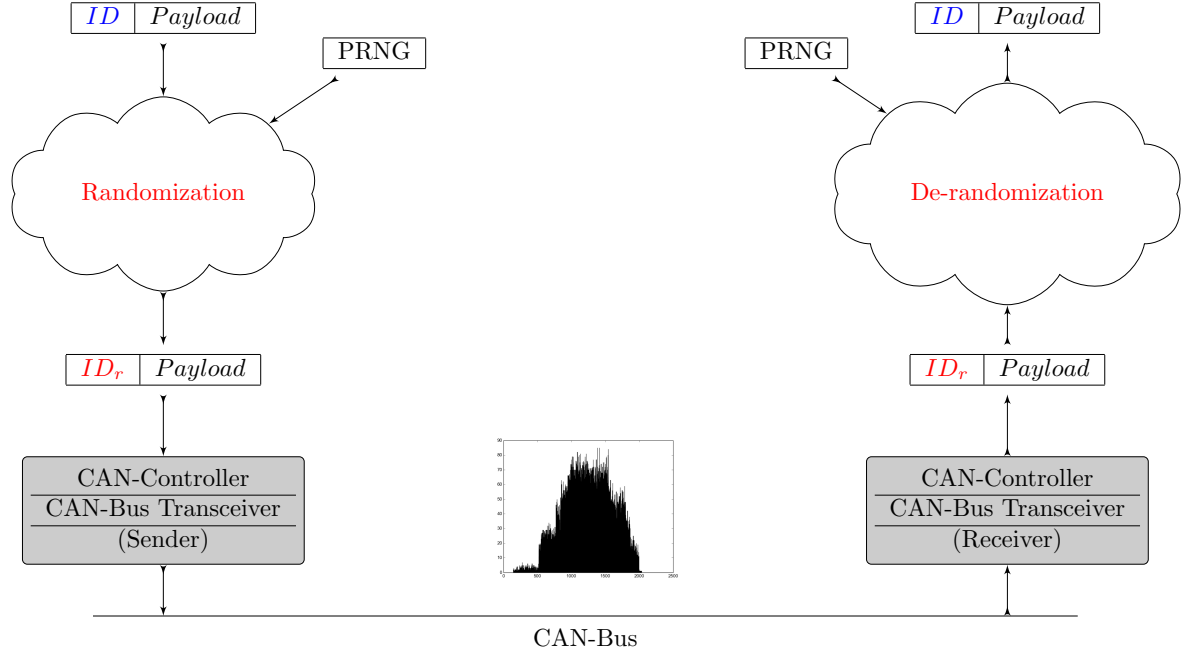


Figure 4.2: CAN-ID randomization principle

predict with high probability the randomized identifier. We achieve this goal by choosing a randomization function based on a cryptographically secure pseudo-random number generator.

4.3 Evaluation metrics

Many randomization functions can meet the previous constraints. In order to compare these functions between them, we need to define security metrics that measure their ability to protect against reverse-engineering and replay/injection attacks. These metrics are based on information theory, which links them to *optimal attacks*, that is attacks which maximize the likelihood of success [35].

4.3.1 Reverse-engineering attack

In the presence of a randomization scheme, the attacker knows that each original identifier has multiple substitute identifiers. The reverse-engineering challenge is then to be able to determine for each original identifier the set of substitute identifiers that it could be randomized into. A randomization scheme is perfect if the resulting randomized identifiers are identically distributed over the set of identifiers. From an information theory point of view, the capacity of the attacker to perform this task is related to the entropy of the resulting randomized identifiers. The more the identifiers look random, the harder it is for the attacker to reverse the protocol. Thus we use the entropy as security metrics to evaluate the protection level of

the randomization function against reverse-engineering.

$$H(id_r) = \sum_{x \in [0, 2^n - 1]} P(id_r = x) \times \log_2 \left(\frac{1}{P(id_r = x)} \right). \quad (4.5)$$

4.3.2 Replay and injection attacks

In order for the attacker to successfully inject a message on the CAN bus with the presence of a randomization function, he/she needs to “predict” the next randomized identifier to be sent. If the attacker successfully conducts a reverse-engineering attack, he/she should be able to predict the next original identifier to be sent. Knowing the original identifier, the attacker has to predict its randomized version. Since we suggested to combine the randomization function with a cryptographically secure pseudo-random number generator that has a uniform distribution, we suppose that the prediction capability of the attacker is not better than a simple “guess”. Thus, the conditional entropy of the randomized identifier knowing the original identifier can be used as a metrics to evaluate the protection level of the randomization function against replay and injection attacks.

$$H(id_r | id_o) = \sum_{x \in [0, 2^n - 1]} P(id_r = x | id_o) \times \log_2 \left(\frac{1}{P(id_r = x | id_o)} \right). \quad (4.6)$$

4.4 Proposed solutions

4.4.1 The IA-CAN Approach

In [48, 49] Han et al. developed a method for CAN protection called Identity Anonymized CAN (IA-CAN). More precisely, they used identifier and data randomization. Their approach is to mix a part of the identifier (LSB part) and a part of the payload with a random variable generated at sender and receiver sides. In this chapter, we focus only on the randomization of the CAN identifier. This is motivated by the fact that if the attacker successfully injects an identifier that gets passed through the CAN filter, even if the rest of the payload is not correct, it will nevertheless exhaust the receiver ECU.

If we disregard the payload part of the anonymization in the IA-CAN approach, we can conclude that the randomization function applied to the identifiers can be expressed with equation- 4.7. We refer the reader to the original paper [48, 49] for further details.

$$\begin{array}{llll} f_r : & [0, 2^a - 1] & \times & [2^a, 2^n - 1] \rightarrow [0, 2^n - 1] \\ & r & & id \rightarrow id_{MSB(n-a)} + id_{LSB(a)} \oplus r \end{array} \quad (4.7)$$

Where:

- n is the number of bits of the identifier ($n = 11$ for standard CAN, $n = 29$ for extended CAN)
- a is the number of bits that will be used for randomization ($a < n$)

4. IDENTIFIER RANDOMIZATION: AN EFFICIENT PROTECTION AGAINST CAN-BUS ATTACKS

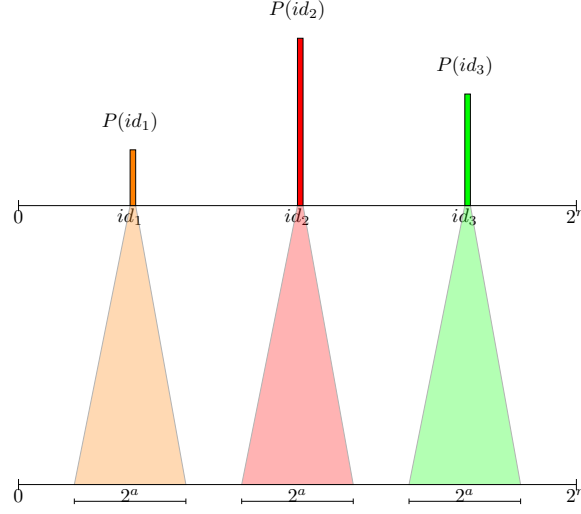


Figure 4.3: Illustration of the IA-CAN identifier transformation approach

- r is a random variable in $[0, 2^a - 1]$ generated at both sender and receiver sides.
- id is the original identifier of the message.
- $id_{MSB(\alpha)}$ is the identifier α Most Significant Bit.
- $id_{LSB(\alpha)}$ is the identifier α Least Significant Bit.

We assume that the random number r is uniformly distributed over the randomization interval $[0, 2^a - 1]$. Figure 4.3 shows the principle of the transformation applied to the identifiers. Naturally, the choice of the variable a is bounded by the total number of original identifiers N , and the minimum of inter-space between all identifiers (equation- 4.8).

$$1 \leq a \leq \left\lfloor \log_2(\text{Min}_{i,j \in [1,N]} |id_i - id_j|) \right\rfloor \quad (4.8)$$

In order to maximize the protection level, directly linked to the randomness of the identifiers, we need to choose the maximum possible a . For better security performance, this choice translates to equation-4.9.

$$a = \left\lfloor \log_2(\text{Min}_{i,j \in [1,N]} |id_i - id_j|) \right\rfloor \quad (4.9)$$

Particular case:

A particular case arises when the identifiers inter-space is constant between all original identifiers. The constant is then $\frac{2^n}{N}$. The upper bound of a is then expressed with equation-4.10.

$$\left\lfloor \log_2(\text{Min}_{i,j \in [1,N]} |id_i - id_j|) \right\rfloor = n - \lceil \log_2(N) \rceil \quad (4.10)$$

To measure the efficiency of this randomization function against reverse-engineering, we compute the entropy of the randomized identifiers. This entropy is expressed in equation- 4.11.

$$H_{IA-CAN}(id_r) = H(id_o) + a \quad (4.11)$$

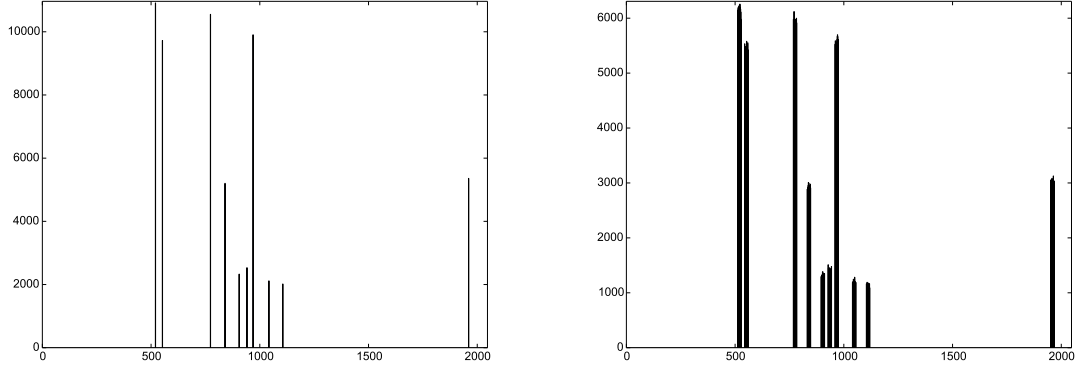


Figure 4.4: IA-CAN transformation: Original (Left) Randomized (Right)

Moreover, to quantify the level of protection against frame injection and replay attacks, we compute the conditional entropy of the randomized identifiers knowing the original identifiers. The results is reported in equation- 4.12.

$$H_{IA-CAN}(id_r|id_o) = a = \left\lfloor \log_2(\text{Min}_{i,j \in [1,N]} |id_i - id_j|) \right\rfloor \quad (4.12)$$

In case the identifiers inter-space is constant, this conditional entropy reaches a maximum expressed in equation- 4.13.

$$H_{IA-CAN}(id_r|id_o) = a = n - \lceil \log_2(N) \rceil \quad (4.13)$$

Proof of equations (4.11) and (4.12) is presented in the Appendix.

Practical analysis:

To test this approach, we made a real acquisition on a vehicle CAN-bus, that we used with this randomization procedure to assess its efficiency. Figure 4.4 shows the identifier histograms before and after randomization. On this particular example, the randomization was done over $a = 4$ bits which means that for each identifier we allocated $2^a = 2^4 = 16$ substitute identifiers. The computed entropy of the original distribution is $H(id_o) = 3.05$. After randomization, the computed entropy of the randomized identifiers is $H_{IA-CAN}(id_r) = 7.05$. The computed conditional entropy is $H_{IA-CAN}(id_r|id_o) = 4$. We can observe from the randomized identifier distribution of figure 4.4 that the attacker can still distinguish frequencies of the messages. It is also clear from equation (4.11) that the entropy of randomized identifiers depends on the entropy of the original identifiers. Using this information the attacker can deduce the next original identifier to be sent, and try to inject a frame within the observed randomization interval.

4.4.2 Equal Intervals

The first observation that we can make concerning the IA-CAN approach is that there is still room for amelioration in terms of entropy and conditional entropy. In fact, the randomization of IA-CAN is done only on the a least significant bits of the identifier, which makes the added

4. IDENTIFIER RANDOMIZATION: AN EFFICIENT PROTECTION AGAINST CAN-BUS ATTACKS

entropy bounded by a which is also bounded by $\log_2(\text{Min}_{i,j \in [1,N]} |id_i - id_j|)$

A possible improvement is first to create a mapping function that assigns to each original identifier a substitute identifier such that the set of substitute identifiers satisfies the equidistance condition, mentioned in the previous section, that maximizes random space. A second improvement is to change the randomization function from an XOR function to an arithmetic addition in order to increase the randomness and thus the entropy. These improvements yield a randomization procedure that we denote *Equal intervals* in this thesis.

Suppose we have N identifiers $\{id_1, id_2, \dots, id_N\}$ such that $id_1 < id_2 < \dots < id_N$. The identifiers are ordered from the most priority (id_1) to the least priority (id_N). We can partition the identifier space $[0, 2^n - 1]$ over N intervals $I_i = [inf_i, sup_i]$ such that:

- ✓ $inf_1 = 0, sup_N = 2^n - 1$
- ✓ For each $i \in [1, N - 1] : inf_{i+1} = sup_i + 1$
- ✓ For each $i \in [1, N - 1] : sup_i - inf_i = const = \frac{2^n}{N}$

Thus, we define an identifier mapping function *Map* in equation- 4.14.

$$\begin{array}{ccc} Map : & [0, 2^{n-1} - 1] & \rightarrow [0, 2^n] \\ & id_i & \rightarrow inf_i \end{array} \quad (4.14)$$

The *Map* function is designed to redefine the distribution of the identifier (by assigning a substitute identifier to the original one) in such a way that the new identifiers maximize the identifier inter-space. At the same time, the *Map* function is priority preserving : $Map(id_1) < Map(id_2) < \dots < Map(id_N)$.

Given this new distribution of substitute identifiers, in each interval $I_i = [inf_i, sup_i]$ we have only one identifier $Map(id_i)$. All the interval can be exploited to randomize that identifier. Thus the randomization function of equation- 4.15.

$$\begin{array}{ccc} f_r : & [0, 2^{n-1} - 1] & \rightarrow [0, 2^n] \\ & id_i & \rightarrow Map(id_i) + r_{[0, sup_i - inf_i]} \end{array} \quad (4.15)$$

Figure 4.5 shows the transformation applied to the identifiers.

In order to compare this proposed solution to the previous one, we compute the security metrics to assess the level of protection against reverse-engineering and injection and replay attacks:

Entropy:

$$H_{EI}(id_r) = H(id_o) + n - \log_2(N) \quad (4.16)$$

Conditional entropy:

$$H_{EI}(id_r | id_o) = n - \log_2(N) \quad (4.17)$$

Proof of equations (4.16) and (4.17) are presented in the Appendix.

In section 4.5 we show that based on theoretical analysis of the proposed metrics, this randomization function is more secure than the state-of-the-art solution.

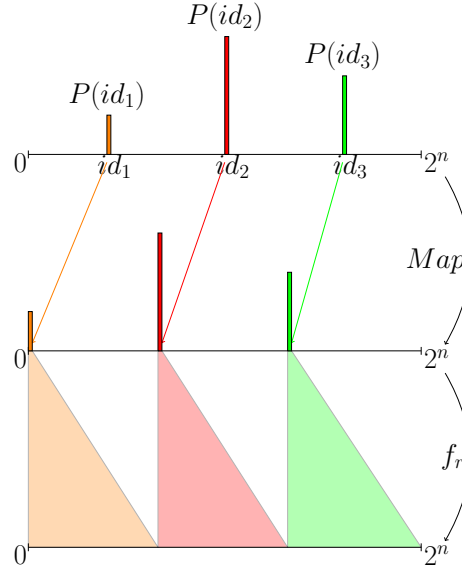


Figure 4.5: Illustration of the Equal intervals identifier transformation

Practical analysis:

The randomization function is applied to the same identifier distribution used in the previous section. Figure 4.6 shows the identifier histograms before and after randomization. We can see that compared to the IA-CAN approach, the equal intervals randomization function (4.15) exploits all the available identifier space. This is mainly due to the *Map* function that re-defines an identifier mapping over all the identifier space. The computed entropy of this particular example is $H_{EI}(id_r) = 10,72$. Compared to the IA-CAN randomization function ($H_{IA-CAN} = 7.05$), the equal intervals randomization function generates more entropy. The conditional entropy of the randomized identifier knowing the original identifier is $H_{EI}(id_r|id_o) = 7.67$. We can also observe that compared to IA-CAN ($H_{IA-CAN}(id_r|id_o) = 4$) it has better conditional entropy. In section 4.5, we formally prove that it is always the case.

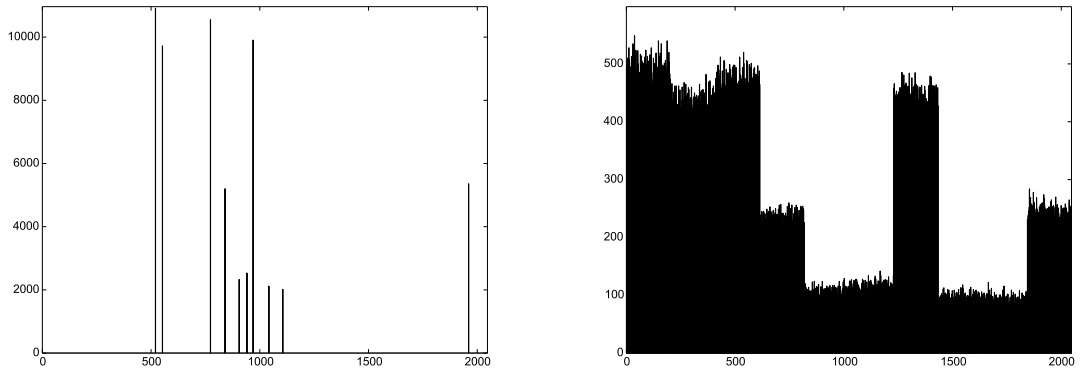


Figure 4.6: Equal intervals transformation: Original (Left) and Randomized (Right)

4. IDENTIFIER RANDOMIZATION: AN EFFICIENT PROTECTION AGAINST CAN-BUS ATTACKS

Nevertheless, the attacker can still identify clusters of randomized identifiers that can guide him in the reverse-engineering process even if the probability of a successful injection is slightly smaller than the previous solution.

4.4.3 Frequency Intervals

The previous methods are not secure enough against reversing the original identifiers. Indeed, given the previous histograms of randomized identifiers, even visually the attacker can identify clusters of identifiers. In fact, since most of these messages are periodic frames, the probability of occurrence of each identifier is impacted by the sending period and so is all of its randomized versions. In this section, we design a new randomization function whose aim is to overcome this limitation. The goal of this function is to make the randomized identifier distribution [histogram] as uniform as possible to improve the entropy of the randomized identifiers and still preserve the priority order. In order to do that, a flattening of the peaks has to be done. We choose the randomization interval of each identifier to be proportional to its frequency of appearance on the CAN bus. Thus an identifier that has a high frequency (small period) will appear more frequently on the CAN bus; this identifier will be assigned a large interval of randomization. Similarly, an identifier that has a small frequency (large period) of appearance on the CAN bus, will appear less frequently, and thus will be assigned a small interval of randomization. In order for this strategy to be possible, we also need a priority preserving mapping function that assigns substitute identifiers to the original identifier. Then we apply the randomization to the substitute identifiers in their respective randomization intervals.

Suppose we have N identifiers $\{id_1, id_2, \dots, id_N\}$ such that $id_1 < id_2 < \dots < id_N$, respectively with a sending frequencies of f_1, f_2, \dots, f_N . The identifiers are ordered from the most priority (id_1) to the least priority (id_N). We can partition the identifier space $[0, 2^n - 1]$ over N intervals $I_i = [inf_i, sup_i]$ such that:

- ✓ $inf_1 = 0, sup_N = 2^n - 1$
- ✓ For each $i \in [1, N - 1] : inf_{i+1} = sup_i + 1$
- ✓ For each $i \in [1, N - 1] : sup_i - inf_i = \frac{2^n \times f_i}{\sum_{j=1}^N f_j} = P(id_i) \times 2^n$

Where $P(id_i)$ is the probability of the identifier id_i to appear on the CAN bus.

We define an identifier mapping function Map (equation- 4.18) that assigns substitute identifiers to the original ones such that the inter-identifier space is proportional to the frequency of the smaller identifier.

$$\begin{aligned} Map : \quad [0, 2^{n-1} - 1] &\rightarrow [0, 2^n] \\ id_i &\rightarrow inf_i \end{aligned} \tag{4.18}$$

The randomization function then assigns a randomized identifier to the substitute identifier. Each identifier is randomized in an interval proportional to its frequency. The randomization

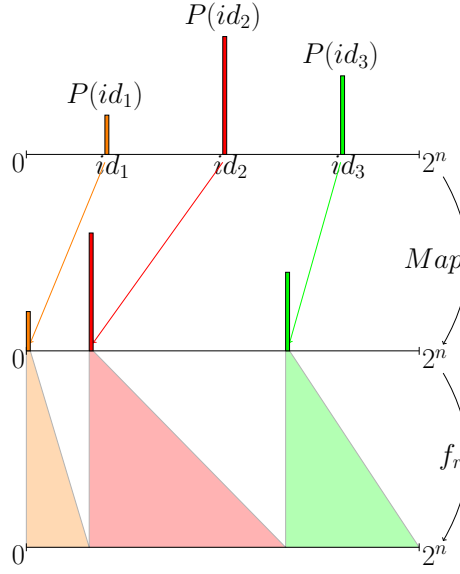


Figure 4.7: Illustration of the Frequency Intervals identifier transformation

function thus described is given in equation- 4.19. Figure 4.7 shows the transformation applied to the identifiers.

$$\begin{aligned} f_r : [0, 2^{n-1} - 1] &\rightarrow [0, 2^n] \\ id_i &\rightarrow Map(id_i) + r_{[0, sup_i - inf_i]} \end{aligned} \quad (4.19)$$

In order to compare this proposed solution to the previous ones, we compute the security metrics to assess the level of protection against reverse-engineering and injection and replay attacks:

Entropy:

$$H_{FI}(id_r) = n \quad (4.20)$$

Conditional entropy:

$$H_{FI}(id_r|id_o) = n - H(id_o) \quad (4.21)$$

A first observation is that in terms of theoretical entropy this solution reaches the maximum entropy which equals to n . Another interesting result is that it gives an enhancement of the conditional entropy as it is shown in section 4.5. From a theoretical analysis, it is shown in Appendix 6.2.2 that the Frequency intervals randomization strategy maximizes the conditional entropy when the mapping is static (i.e. does not change over time). We will see in the next section that a dynamic mapping can increase even more the conditional entropy.

Practical analysis:

To test this randomization strategy, we apply it to the identifier distribution used for the previous functions. Figure 4.8 shows the identifier histograms before and after randomization. The computed entropy for this example is $H_{FI}(id_r) = 10.99$. The computed conditional entropy is $H_{FI}(id_r|id_o) = 7.94$. It is clear from the histogram and the computed entropy that the randomized identifier distribution is more uniform than the previous solutions. A uniform distribution of identifiers is a perfect protection against reverse-engineering as it is harder for the attacker to distinguish clusters of identifiers. We can also observe that there is an

4. IDENTIFIER RANDOMIZATION: AN EFFICIENT PROTECTION AGAINST CAN-BUS ATTACKS

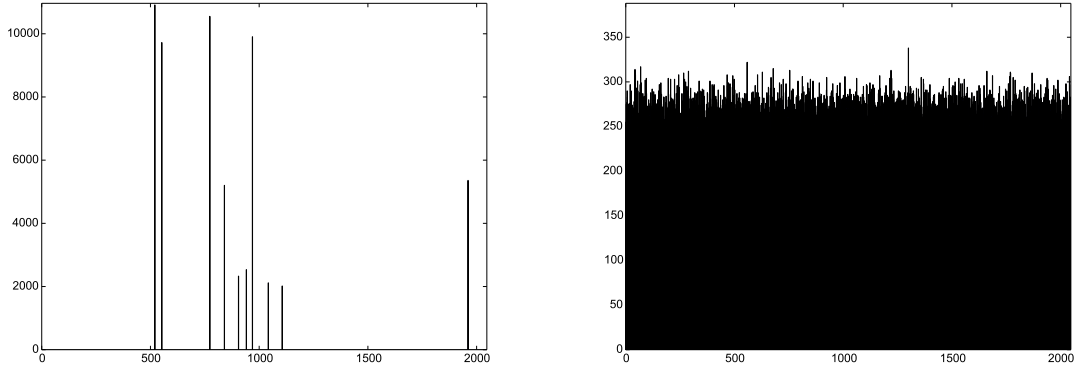


Figure 4.8: Frequency intervals transformation: Original (Left) and Randomized (Right)

enhancement in terms of conditional entropy compared to the previous solutions. That is to say, this solution has better security performance against injection and replay attacks.

4.4.4 Dynamic Intervals

In the previous section, we found the optimal randomization solution that can be implemented with a static partition of the identifier space. This partitioning is designed to maximize the entropy and conditional entropy for a maximum security level against reverse-engineering and injection and replay attacks at the same time. In this section, the goal is to keep the same level of entropy (maximum) but explore new ways to improve the conditional entropy. We prove that this can be possible if we no longer follow a static partitioning of the identifier space.

A practical observation of the CAN bus behavior shows that there is a strong dependency between consecutive identifiers. In other words, the order in which the identifiers appear on the CAN bus is not entirely random. Suppose we have a set of used identifiers $\{id_1, id_2, \dots, id_N\}$, at an instant t where id_i appears on the CAN bus, there is a handful of identifiers that can appear right after it on the bus. Probabilistically, the majority of identifiers will have zero probability to appear right after id_i . To theoretically consolidate this observation, we can argue that two identifiers with the same period that are not sent right after each other at the start of the system, will probably never be seen right after each other as long as the system is up and running. This observation involves that using a fixed identifier mapping after that identifier id_i has been sent, an essential part of the allocated space for identifiers will not be used. Hence, if the mapping is changed dynamically after every sending of id_i , and according to the dependency between identifiers, we can exploit more space for randomization, thus increasing the conditional entropy.

To construct such a randomization function, we model the dependency between consecutive identifiers with a Markov chain. The associated Markov matrix can be built to give the probabilities $p_{i,j} = P(id_j^{t+1}/id_i^t)$ of receiving an identifier id_j at iteration $t + 1$ knowing that

we received the identifier id_i at iteration t .

$$M = \begin{pmatrix} . & id_1^{t+1} & id_2^{t+1} & \dots & id_j^{t+1} & \dots & id_n^{t+1} \\ id_1^t & p(id_1^{t+1}/id_1^t) & p(id_2^{t+1}/id_1^t) & . & . & . & p(id_n^{t+1}/id_1^t) \\ id_2^t & p(id_1^{t+1}/id_2^t) & p(id_2^{t+1}/id_2^t) & . & . & . & p(id_n^{t+1}/id_2^t) \\ id_3^t & . & . & . & . & . & . \\ \vdots & . & . & . & . & . & . \\ id_i^t & . & . & . & p(id_j^{t+1}/id_i^t) & . & . \\ \vdots & . & . & . & . & . & . \\ id_n^t & . & . & . & . & . & . \end{pmatrix} \quad (4.22)$$

$p(id_j^{t+1}/id_i^t)$: is the probability of receiving the identifier id_j at iteration $t + 1$ knowing that at iteration t we received the identifier id_i . Thus, we have equation- 4.23.

$$\sum_{id_j^{t+1}} p(id_j^{t+1}/id_i^t) = \sum_{j \in [1, N]} p_{i,j} = 1 \quad (4.23)$$

Each time an identifier id_i is received, immediately after, there is $p(id_j^{t+1}/id_i^t)$ probability to receive id_j . With this in mind, the idea is to opt for the Frequency Intervals strategy to randomize the upcoming identifiers since it is the optimal strategy that guarantees the maximal entropy. The interval partition is updated according to Frequency intervals strategy that depends on the received identifier at instant t and the probabilities in the Markov transition matrix. It yields a dynamic identifier mapping function defined in equation- 4.24.

$$\begin{aligned} Map^{t+1} : [0, 2^n - 1] &\rightarrow [0, 2^n - 1] \\ id_{i+1} &\rightarrow id_i + 2^n \times p_{k,i} \end{aligned} \quad (4.24)$$

The Map function has to be updated every received identifier according to the Frequency Interval strategy. At an instant $t + 1$, knowing that the previous sent identifier is id_k , identifier id_i have an assigned randomization interval of I_i of width $W(I_i) = 2^n \times p_{k,i}$. The resulting randomization function is expressed in equation- 4.25.

$$\begin{aligned} f_r^{t+1} : [0, 2^n - 1] &\rightarrow [0, 2^n - 1] \\ id_i &\rightarrow Map^{t+1}(id_i) + r_{[0, 2^n \times p_{k,i}]} \end{aligned} \quad (4.25)$$

Illustrative example:

As an example, consider the following sequence of identifiers appearing on the CAN bus: $[id_2, id_3, id_1, id_2, id_3, id_1, id_2, id_3, id_2, id_1, id_2, id_3, id_2, id_1, id_2]$. After analyzing the sequence, the following transition matrix can be established:

$$M = \begin{pmatrix} . & id_1^{t+1} & id_2^{t+1} & id_3^{t+1} \\ id_1^t & 0 & 1 & 0 \\ id_2^t & \frac{1}{3} & 0 & \frac{2}{3} \\ id_3^t & \frac{1}{2} & \frac{1}{2} & 0 \end{pmatrix} \quad (4.26)$$

This transition matrix is used to define new mapping upon reception of a new identifier. Figure 4.9 shows the transformation applied to the identifiers after reception of id_2 , then id_3 .

4. IDENTIFIER RANDOMIZATION: AN EFFICIENT PROTECTION AGAINST CAN-BUS ATTACKS

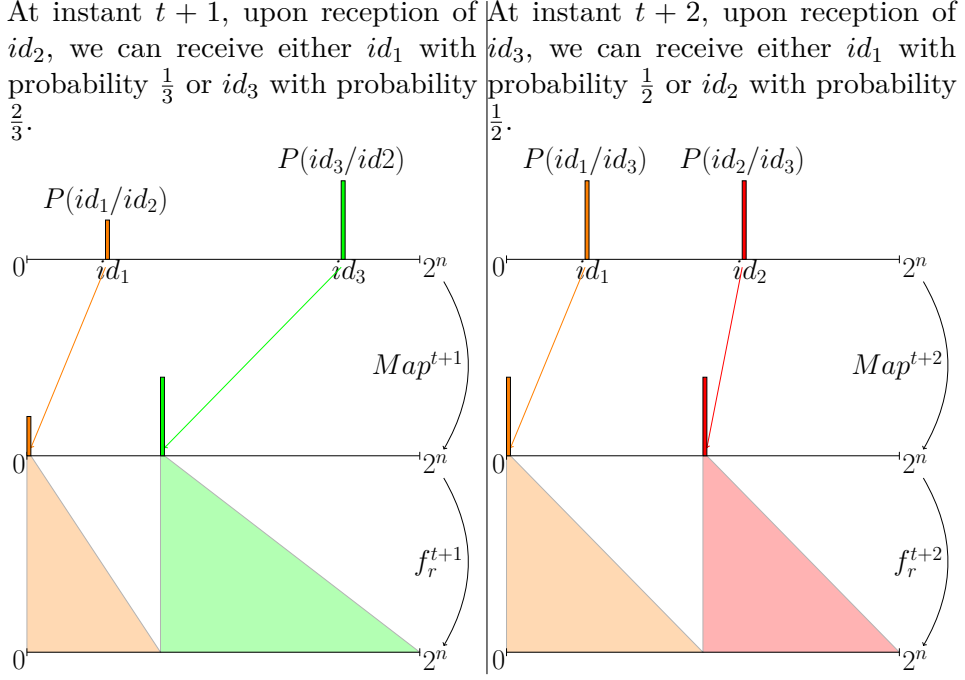


Figure 4.9: Illustration of the Dynamic intervals identifier transformation at $t + 1$ (Left) and $t + 2$ (Right)

The security metrics obtained with the Dynamic Intervals strategy is the following:

Entropy:

$$H_{DI}(id_r) = n \quad (4.27)$$

Conditional entropy:

$$H_{DI}(id_r^{t+1}|id_o^{t+1}) = \sum_{x \in [0, 2^n]} \sum_{id_j^{t+1}} \sum_{id_i^t} \frac{1}{W(I_{i,j})} P(id_i) \log_2 \left(\frac{1}{\sum_{id_k^t} \frac{1}{W(I_{k,j})} P(id_k)} \right) \quad (4.28)$$

Practical analysis:

To test this randomization strategy, we apply it to the identifier distribution used for the previous functions. Figure 4.10 gives the resulting randomized distribution.

The computed entropy for this example is $H_{DI}(id_r) = 11$. The computed conditional entropy is $H_{DI}(id_r|id_o) = 10.24$. It is clear from the histogram and the computed entropy that the randomized identifier distribution is as uniform as the Frequency-intervals randomization strategy. Moreover, this method provides a significant enhancement in terms of conditional entropy, compared to the previous solutions.

4.4.5 Arithmetic Masking

All of the above-proposed solutions can be applied at the software level (Layer 3). In this section, we consider a hardware solution which can involve some change in the CAN controllers.

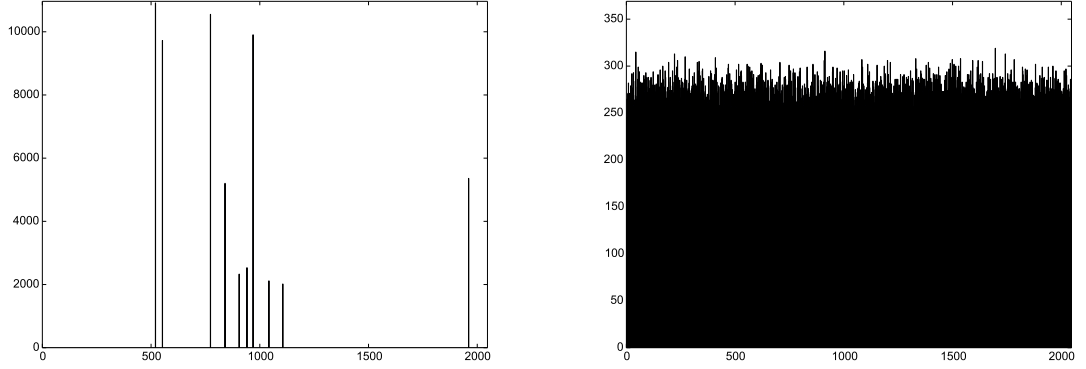


Figure 4.10: Dynamic intervals transformation: Original (Left) and Randomized (Right)

The payoff of this choice is to eliminate the third constraint imposed on section 4.2 that states that the randomization function has to preserve priority over time. Here we consider that the new hardware at the physical layer does not have any frame buffer. Hence all the CAN controllers can share the same random variable in a consistent manner. The internal changes of this random variable could be done by a Pseudo Random Number Generator (PRNG) which is initialized identically in every CAN controller at start-up.

The hardware randomization proposal is based on Arithmetic Masking, meaning that the random variable is added arithmetically to the original identifier. The operations are the following :

- First a mapping function is defined. It assigns new substitute identifiers to the original identifiers.
- Then the randomization is performed by adding the random variable to the substitute identifier.
- The random variable is such that it is shared with all CAN controllers and the randomized identifier does not exceed 2^{11} . This allows preserving the priority between identifiers.

Suppose there are N identifiers $id_1 < id_2 < \dots < id_N$, with a sending frequencies of f_1, f_2, \dots, f_N . A substitute and random identifier is assigned for each original identifier. The identifier mapping function is defined with equation- 4.29:

$$\begin{aligned} Map : [0, 2^{n-1} - 1] &\rightarrow [0, 2^n] \\ id_i &\rightarrow i - 1 \end{aligned} \quad (4.29)$$

The mapping function substitutes the original identifiers with the N first lowest identifiers. The rest of the interval $[N, 2^n]$ is used for randomization. Hence the randomization function expressed in equation- 4.30.

$$\begin{aligned} f_r : [0, 2^n - N] &\times [0, 2^{n-1} - 1] \rightarrow [0, 2^n] \\ r &id_i \rightarrow Map(id_i) + r \end{aligned} \quad (4.30)$$

4. IDENTIFIER RANDOMIZATION: AN EFFICIENT PROTECTION AGAINST CAN-BUS ATTACKS

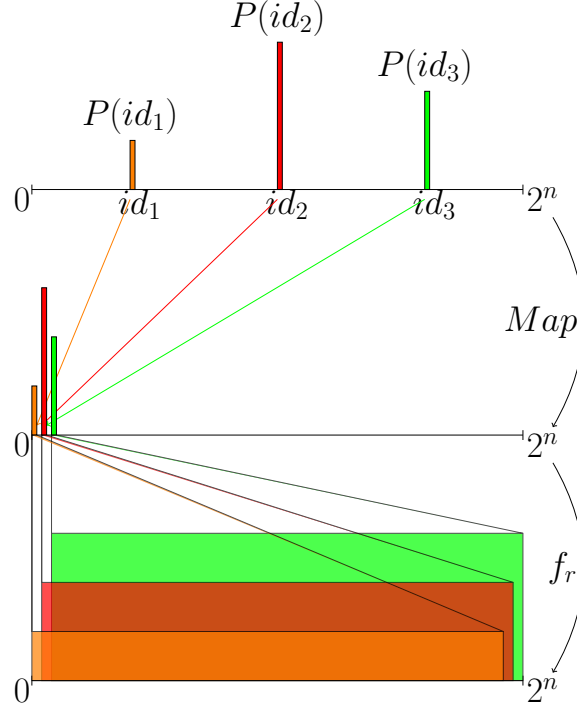


Figure 4.11: Illustration of the Arithmetic masking identifier transformation

Figure 4.11 shows the transformation applied to the identifiers.

The security metrics applied to the Arithmetic Masking solution give the following results:
Entropy:

$$\begin{aligned}
 H_{AM}(id_r) = & \log_2(2^n - N + 1) + \frac{1}{2^n - N + 1} \sum_{x \in [0, N-2]} \sum_{i=0}^x P(id_i) \times \log_2\left(\frac{1}{\sum_{i=0}^x P(id_i)}\right) \\
 & + \sum_{i=x+1}^{N-1} P(id_i) \times \log_2\left(\frac{1}{\sum_{i=x+1}^{N-1} P(id_i)}\right)
 \end{aligned} \tag{4.31}$$

Conditional entropy:

$$H_{AM}(id_r | id_o) = \log_2(2^n - N + 1) \tag{4.32}$$

Practical analysis:

To test this randomization strategy, we apply it to the identifier distribution used for the previous functions. The computed entropy for this example is $H_{AM}(id_r) = 10.99$. The computed conditional entropy is $H_{AM}(id_r | id_o) = 10.99$. The histogram and the computed entropy show that the randomized identifier distribution is approximately uniform. Moreover, we observe a significant enhancement in terms of conditional entropy compared to the previous solutions.

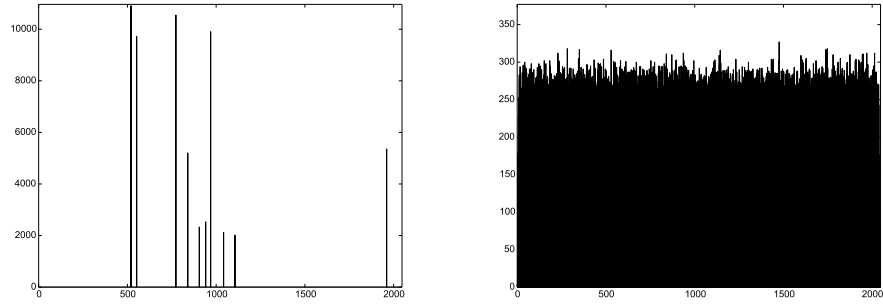


Figure 4.12: Arithmetic masking transformation: Original (Left) and Randomized (Right)

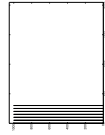
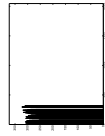
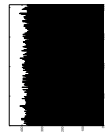

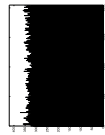
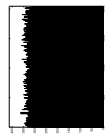
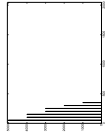
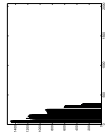




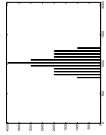

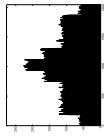
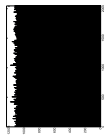
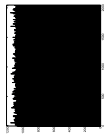
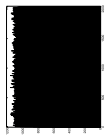



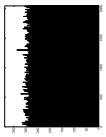


4.5 Comparison

In the previous section, we introduced the state-of-the-art solution for CAN identifier randomization, and we proposed solutions both at software and hardware layers. These solutions were tested on a real identifier trace captured from a real vehicle. In this section, we compare the proposed solutions applied to multiple identifier distributions based on the proposed security metrics.

Four reference identifier distributions are considered. Table 4.1 summarizes the obtained results.

4. IDENTIFIER RANDOMIZATION: AN EFFICIENT PROTECTION AGAINST CAN-BUS ATTACKS

Table 4.1: Comparison between different randomization strategies

Reference distribution	IA-CAN	Equal intervals	Frequency intervals	Dynamic intervals	Arithmetic masking
					
$H(id_o) = 2.80$	$H(id_r) = 7.80$	$H(id_r) = 10.9997$	$H(id_r) = 10.99$	$H(id_r) = 10.99$	$H(id_r) = 10.9948$
					
$H(id_o) = 2.68$	$H(id_r) = 7.68$	$H(id_r) = 10.86$	$H(id_r) = 10.99$	$H(id_r) = 11$	$H(id_r) = 10.99$
					
$H(id_o) = 3.35$	$H(id_r) = 8.35$	$H(id_r) = 10.88$	$H(id_r) = 10.99$	$H(id_r) = 11$	$H(id_r) = 10.99$
					
$H(id_o) = 3.05$	$H(id_r) = 7.05$	$H(id_r) = 10.72$	$H(id_r) = 10.99$	$H(id_r) = 11$	$H(id_r) = 10.99$

The visual inspection of the histograms indicates that Frequency-intervals and Dynamic-intervals randomization strategies have more uniform distribution than Equal-intervals and IA-CAN. Hence, these solutions should better protection against reverse-engineering attack, at first glance. This observation can be theoretically proven. By comparing the closed-form expressions of the respective entropies, we can establish the following:

$$H_{IA-CAN}(id_r) \leq H_{EI}(id_r) \leq H_{FI}(id_r) = H_{DI}(id_r) \quad (4.33)$$

Proof.

$$H(id_o) \leq \log_2(N)$$

And we can establish from equations (4.8) and (4.10) that:

$$\begin{aligned} a &\leq n - \lceil \log_2(N) \rceil \leq n - \log_2(N) \\ \Rightarrow H(id_o) + a &\leq H(id_o) + n - \log_2(N) \\ \Rightarrow H_{IA-CAN}(id_r) &\leq H_{EI}(id_r) \end{aligned}$$

Since

$$H_{DI}(id_r) = H_{FI}(id_r) = n$$

Then:

$$H_{IA-CAN}(id_r) \leq H_{EI}(id_r) \leq H_{FI}(id_r) = H_{DI}(id_r)$$

□

It is clear that compared to Arithmetic Masking, Dynamic Intervals and Frequency Intervals have better performance in terms of entropy, involving high robustness against reverse engineering. Comparing the Arithmetic Masking to IA-CAN and Equal Intervals is not trivial. This is mainly because established entropy expressions depend on the entropy of the original identifier distribution. If we consider that the original identifiers have equal probabilities (example of the first distribution), the Equal Intervals solution has better entropy ($H_{EI}(id_r) = 10.9997$, $H_{AM}(id_r) = 10.9948$). Theoretically, the entropy of Equal intervals for this example is maximal. On the other side, concerning the second distribution, we can observe that the Arithmetic Masking performs better.

To compare the protection level against replay and injection attacks, the conditional entropy metric is used. Based on the closed form expressions established in the previous sections, we draw the curve showing the evolution of the conditional entropy as a function of the total number of identifiers. Figure 4.13 show the results. From this graph, we can conclude that all the proposed solutions outperform the IA-CAN strategy. Second, it appears that the hardware-based solution, namely Arithmetic Masking is the best against replay and injection attacks. However, As discussed previously, the Arithmetic Masking needs to be implemented in the CAN controller between the physical and data link layer, which makes it not easy to deploy. At the software level, the Frequency Intervals strategy performs the best, both against replay, injection attacks and reverse-engineering.

4. IDENTIFIER RANDOMIZATION: AN EFFICIENT PROTECTION AGAINST CAN-BUS ATTACKS

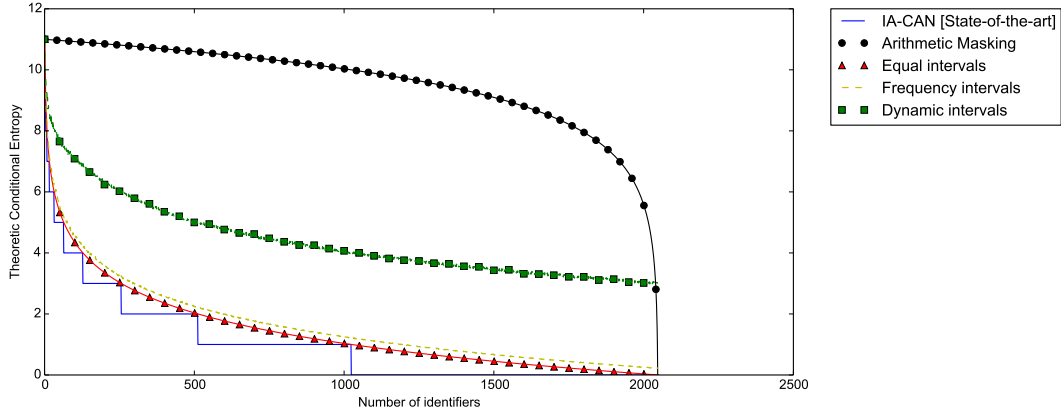


Figure 4.13: Conditional entropy $H(id_r|id_o) = f(N)$

4.6 Conclusion

For an attacker model that has direct physical access to the in-vehicle communication networks, it appears that one of the most efficient class of protection is based on the randomization of the CAN identifiers. Starting from the existing Identity-Anonymized CAN (IA-CAN), three significant enhancements based on randomization have been proposed: with Equal Intervals, Frequency Intervals, and Dynamic Intervals. In case it is possible to change the CAN hardware, randomization based on Arithmetic Masking has also been introduced. The security assessment has been carried out by using security metrics coming from the information theory: entropy (for the reverse engineering attack) and conditional entropy (for the replay and injection attacks). It has been shown that the enhanced protections provide a significant gain compared to the IA-CAN approach. The entropy obtained from the new randomization solutions is very near the optimum (11 bits), thus presenting high robustness against Reverse Engineering Attacks. The conditional entropy is better achieved with the Arithmetic Masking and the Dynamic Intervals. This last solution has the interest not to modify the hardware of the CAN interface. Overall, the proposed solutions are much better than the existing IA-CAN, as proven by the resulting security gain formally expressed by means of information theory metrics. In the next chapter, we investigate a protection solution against an attacker that has indirect and remote access to one of the legitimate ECUs inside the vehicle cyber-physical architecture.

On-board Intrusion Detection and Prevention system

In this chapter, we introduce a novel intrusion detection technique over the in-vehicle network. Results presented in this chapter has been scientifically valued by an article published in the proceedings of the *Workshop on information security theory and practice* [73].

Contents

5.1	Introduction	97
5.2	Machine learning algorithms	99
5.3	Principle and problem formulation	100
5.4	Validation metrics	105
5.5	Supervised learning algorithms	108
5.6	Data collection and feature engineering	116
5.7	Experimental validation and discussion	119
5.8	Evaluation against attacks	131
5.9	Alerts handling	136
5.10	Conclusion and discussion	138

5.1 Introduction

In Chapter 2, we introduced in-vehicle intrusion detection and prevention systems (IDS/IDPS) as one of the prominent protection mechanisms that can be applied to protect the connected vehicle. The advantage of these systems with regards to other protection mechanisms, namely payload protection and identifier protection, is that they can protect not only against an attacker that has physical access to the CAN bus but also against an attacker that has indirect access and remote access to a legitimate ECU.

Recall from Section 2.5.3 and Figure 2.9 that an attacker that has direct physical access to in-vehicle communication buses can inject and replay messages to other ECUs which

5. ON-BOARD INTRUSION DETECTION AND PREVENTION SYSTEM

can potentially be detected by *Rule-based* intrusion detection mechanisms that monitor the syntax and periodicity of the messages. On the other hand, an attacker that has indirect or remote access to the in-vehicle communication bus through one of the legitimate ECUs can inject malicious content directly inside the payload without disrupting the defined protocol. Although this type of access is technically more challenging for an attacker as it involves taking control over a legitimate ECU leveraging hardware or software vulnerability, it nevertheless represents a higher risk as the accessibility and range of the attack vector is more important. Detection systems able to counter these kinds of attacks have to be able to detect bad behavior inside the payload. They are called *deep-packet-inspection*.

Section 2.5.3.3 already introduced the state of the art intrusion detection techniques used for the CAN network. Rule-based detection mechanisms are very effective as they allow only for compliant packets to be accepted. Nevertheless, they are adapted to an attacker model that has direct physical access to the communication bus. Intrusion detection mechanisms that are adapted to an attacker model with indirect and remote access to a legitimate ECU include outlier detection techniques (based a statistical measure like entropy or hamming distance) and classification techniques that use machine learning algorithms to recognize attacked frames. Outlier detection mechanisms generally rely on a statistical measure and do not allow to know the precise CAN frames that carry the attack payload. In fact, they report misbehavior detected on a relatively large time window (like entropy) Classification-based intrusion detection mechanisms address this particular limitation and are able to point out the frame with anomalous content. In fact, they are trained with examples of attacked frames and normal frames. This is also a limitation because in order to produce this kind of data one need to select and perform multiple attacks on the vehicle. Thus it is challenging to generate the data for a large range of attacks. Besides, the intrusion detection system learns only to recognize performed attacks included in the training set.

Chapter contributions. This chapter introduces a novel intrusion detection system developed in order to monitor vehicle state from information collected on internal buses. We tackle the problem of deep packet inspection of in-vehicle networks from a practical viewpoint. In order to overcome the limitations of state of the art methods, the problem is formulated using supervised machine learning techniques, in a way that allows learning the normal behavior of the system in terms of message payload content. The principle is to learn how to predict the next state of the vehicle based on information and sensor values sent over communication buses. The normal behavior is then used as a reference to detect deviations. Bad behavior and bad payload content is flagged with outlier detection techniques.

The method thus described can be adopted not only as an intrusion detection mechanism but also as an *online monitoring failure detection* and a *Sensor rationality check* safety mechanisms as described by the “Road vehicles – Functional safety” standard ISO-26262 [63]. We validate in practice the model with real CAN traces collected from drive tests. We show that the approach is able to learn the nominal behavior with high accuracy and low false positives, for three different driving behaviors separately. Then we show that it is also able to learn a unified nominal behavior with high accuracy and low false positives, that can accommodate different driving behaviors. Finally, we run an attack campaign in order to test

the robustness of the detection rules and demonstrate its ability to predict attacks with a low false negative rate.

The remainder of the chapter is structured as follows. Section 5.2 gives some background on machine learning techniques, the problem formulation and a presentation of the algorithms used during evaluation and validation. Section 5.3 presents the principle and theoretical foundation underlying the intrusion detection techniques using supervised learning algorithms as well as an introduction to the specific learning algorithms used in the remainder of the chapter for evaluation. Section 5.6 gives details about data collection and feature engineering. Section 5.7 presents practical validation results on real CAN traces. Section 6 concludes.

5.2 Machine learning algorithms

This section introduces state of the art CAN intrusion detection methods, and their limitations. Then it introduces machine learning techniques and how they can be applied to intrusion detection applications.

5.2.1 Learning strategy

In practice there are multiple application domains where machine learning algorithms excelled in prediction tasks. They are generally used to study correlation/dependencies between different inputs (also called features) and used to approximate an output function and/or to discover interesting data structures.

Machine learning algorithms can be divided into two main categories depending on the learning strategy (Figure 5.1):

- *Supervised learning*: a machine learning algorithm is said to be using supervised learning strategy when the training set includes both the input data and the output data of the algorithm. In that sense, the machine algorithm is training to learn a mapping function by minimizing a pre-defined cost function. The trained algorithm is then tested on some other examples that were not included in the training set. It is said to be generalizing well if the performance of the trained algorithm on the test set is comparable to its performance on the training set.
- *Un-supervised learning*: a machine learning algorithm is said to be using unsupervised learning strategy if the training only includes the input data but not the expected output. In that sense, the machine learning algorithm is trying to discover interesting data structures.

5.2.2 Parametric and non-parametric models

In machine learning, an important step is choosing a model for the data. In general, we can distinguish between two types of models:

5. ON-BOARD INTRUSION DETECTION AND PREVENTION SYSTEM

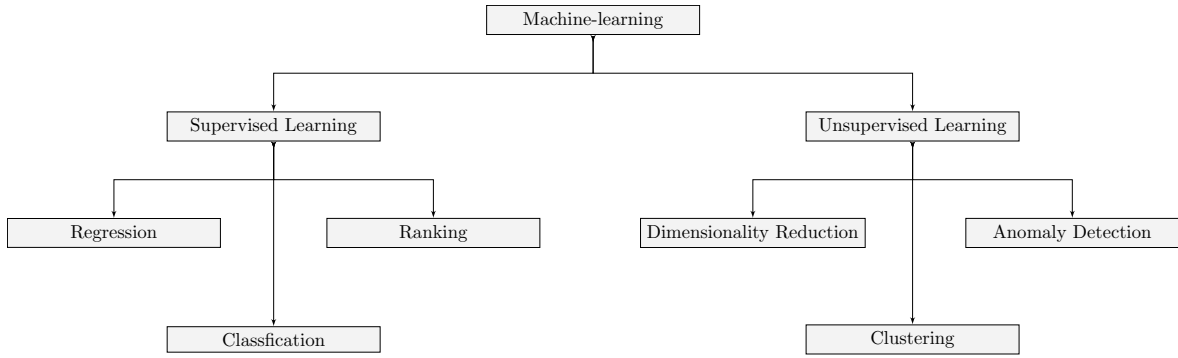


Figure 5.1: Machine learning taxonomy

- Models where some hypothesis about the data distribution is made resulting in a fixed number of parameters. These are called *parametric* models.
- Models where the number of parameters grows with the amount of training data. These are called *non-parametric* models.

Using a parametric model with a fixed number of parameters can significantly simplify the learning process, but can also limit what can be learned. Thus parametric models have the advantage of being faster and simpler to use, but the disadvantage of making strong assumptions on data distributions which can sometimes result in bad approximations and a poor fit of the data. On the other hand, non-parametric models are more flexible in the sense that they do not impose strong assumptions, but often consume a lot of computational and/or memory resources for large datasets.

5.3 Principle and problem formulation

In order to overcome the limitations identified in state of the art *deep packet inspection* techniques, the goal of this section is to present and formulate the problem in a way that allows learning a nominal behavioral model of the frames of in-vehicle communication buses.

As explained Chapter 2, the vehicle cyber-physical system architecture is composed of multiple ECUs, holding sensors and actuators and sharing communication buses used to send sensor information and actuator commands. In this chapter, we refer to them as *signals*.

5.3.1 Signal types

In general we can identify two types of signals (Figure 5.2 gives examples of these signal types):

1. *Real-valued signal*: is a piece of information that can take multiple values generally sent over more than one data byte. This type encompasses in general multiple sensors measurements like vehicle engine rotational speed, vehicle acceleration, engine torque ... An example of a real-valued signal is the vehicle speed of the vehicle (Figure 5.2a). It is sent over 2 bytes of payload information. The received value is then an integer between

0 and 65535. A multiplication by 0.01 is necessary to recover the actual measurement of the sensor to make speed range in $[0, 655.35]$ km/h .

2. *Categorical signals*: is a piece of information that can take a finite and relatively small set of values (also called categories or classes). It is generally sent over a small number of bits. This type encompasses in general multiple sensors states and commands like open/closed door command, gear-box position sensor ... An example of a categorical signal is the brake lights command signal (Figure 5.2b). It is sent over 1 bit of payload data. The received value is binary information (0/1) indicating whether to activate the brake lights (1) or not (0).

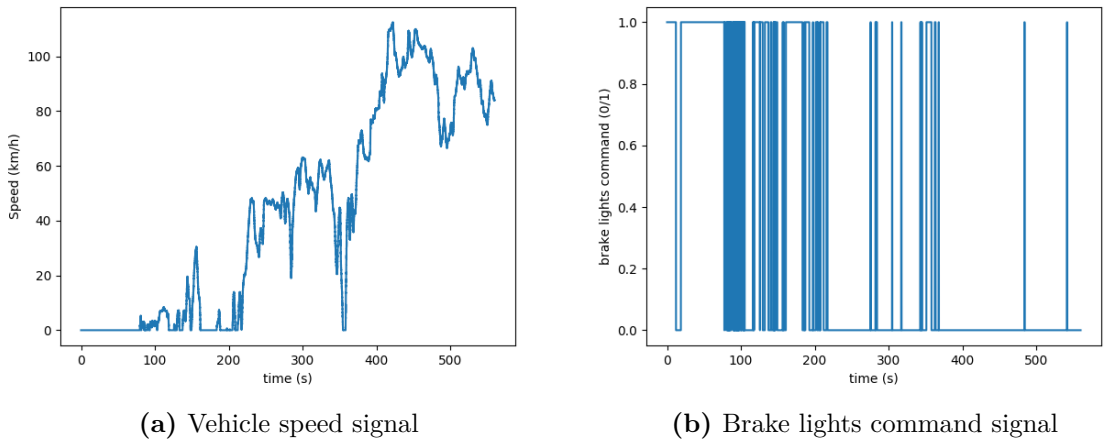


Figure 5.2: Example of real-valued and categorical signals

5.3.2 Intrusion detection principle

The ECUs use the sensed information in order to elaborate and take decisions that ultimately produce commands issued for the actuators. Produced commands can in return have an effect on sensed information. Safety-critical attacks are intentional signal manipulations whose goal is to cause undesired physical effects. When signals are manipulated before being handed to other ECUs and actuators, the security property of integrity and authentication is violated. If there is no canonical security solution guaranteeing that the communicated information from a sensor or a command from an ECU is authentic, dependencies between signals may help the defender to detect impossible or implausible sensor/command information. It becomes then possible to build an approach to detect when communicated information is being maliciously manipulated.

The set of signals composed of sensor information and actuator commands define a particular state of the vehicle. There are certain dependencies and correlations between these signals in the sense that they obey to causal relationships and/or are tied together to the overall state of the vehicle. To grasp the concept of dependency, consider the following simple examples: if the brake-lights-commands is activated, it usually means that the driver has

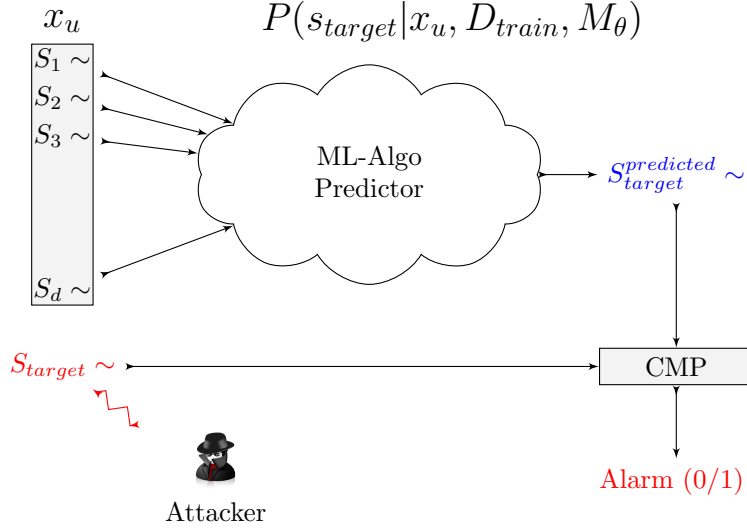


Figure 5.3: Prediction principle

activated the brake pedal which in return means that if the vehicle is moving, it must be decelerating. Similarly, if the gear-box position is on “Reverse” or the first speed, then its a high indication that the vehicle speed is relatively small, but if it is on the fifth position, the speed should be relatively high. Using these simple examples, we can already predict values or confidence intervals of some signals depending on other signals and vehicle state.

However, we can find dependencies that are clear and intuitive, but also others that the human mind cannot quite predict. This is where it is highly interesting to use machine learning techniques. First because of their ability to catch linear as well as non-linear and complex dependencies. Second, because they are able to build a model simply from a set of data inputs. Thus the idea is to train machine learning algorithms to exploit these dependencies in order to build a signal predictor ($S_{target}^{predicted}$ in Figure 5.3) for a target sensitive signal (S_{target} in Figure 5.3) solely based on other signals ($\{S_1, S_2, \dots, S_d\}$ in Figure 5.3). The principle is to break down the payload information into signals according to the manufacturer proprietary protocol and then to train a machine learning algorithm (“ML-Algo Predictor“ in Figure 5.3) to predict the next signal value. We can then compare the predicted signal and the received signal. Under the assumption that the predictor is *accurate enough*, we assume the following as a security metric: if the difference between the prediction and the received value is *large enough*, then, with a high probability, the signal is being maliciously manipulated. This means that the vehicle is being attacked and that the predicted signal is the potential cause of the attack. Figure 5.3 shows the principle of the proposed intrusion detection mechanisms.

The advantage compared to previous work is that during the training process the proposed approach does not need data representing attacked and non-attacked states in order to learn to recognize attacks. It needs only data representing the normal functioning of the vehicle in order to build a nominal behavior. Deviations from this nominal behavior during monitoring phase will then be considered as attacks.

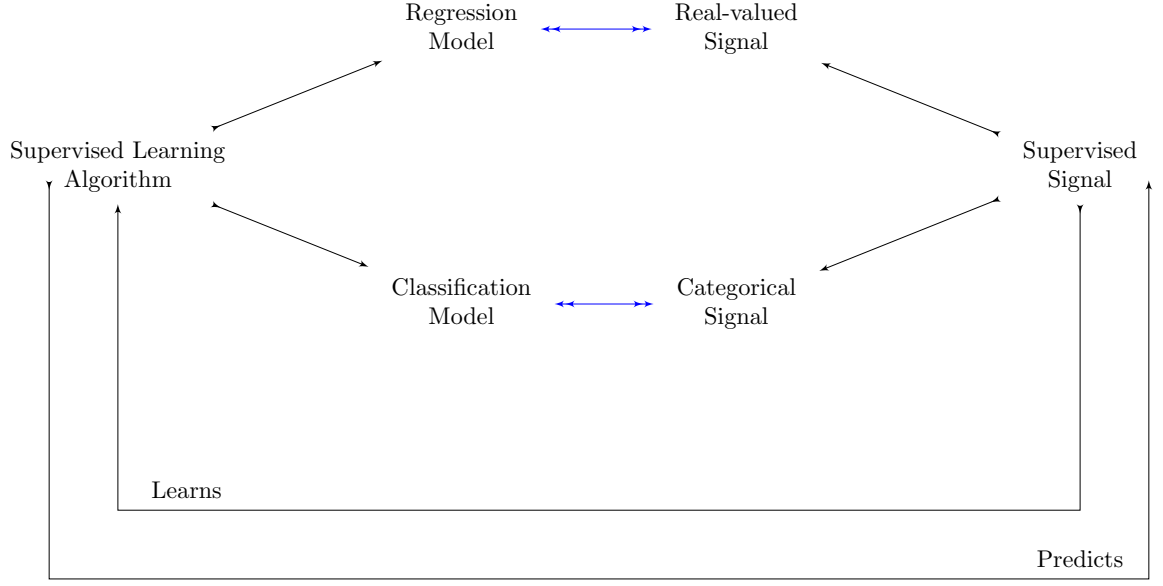


Figure 5.4: Model choice depending on the target signal type.

5.3.3 Mathematical formulation

In what follows we formulate our problem as a supervised machine learning problem. Let $\mathcal{D} = \{(x_i, y_i)\}_{i \in [1, N]}$ be the set of input-output pairs. Here \mathcal{D} is the collected Data set, and N is the number of observed examples. Each training input $(x_i)_{i \in [1, N]}$ is a d -dimensional vector of components representing signal values/states $(s_i^{(1)}, s_i^{(2)}, \dots, s_i^{(d)})$. These are called features and are stored in an $(N \times d)$ matrix X . The output $(y_i)_{i \in [1, N]}$ is stored in a 1-dimensional vector y and represents the target signal that we want to predict. It can be either real-valued (in this case we will talk about *regression*) or a categorical value (in which case we will talk about *classification*), depending on the signal type (Figure 5.4).

The object of supervised machine learning is to assume the existence of some unknown function $\langle f \rangle$ that maps the inputs to the outputs, as in (5.1).

$$f(x) = y, \quad \forall (x, y) \in \mathcal{D} \quad (5.1)$$

During the learning process, the goal is to estimate the function $\langle f \rangle$ given a labeled training set and then to make predictions on unseen data x_u using the estimated function. We denote by $\langle \hat{f} \rangle$ the estimate of the mapping function $\langle f \rangle$.

Predicting the output y_u given the input vector x_u can be established with (5.2).

$$y_u = f(x_u) \quad (5.2)$$

Since the mapping function $\langle f \rangle$ is not known, and we have an estimated mapping function $\langle \hat{f} \rangle$, an estimated output prediction \hat{y}_u with (5.3).

$$\hat{y}_u = \hat{f}(x_u) \quad (5.3)$$

We denote the probability distribution over possible labels, given the input vector x_u and the training data set \mathcal{D}_{train} by $P(y|x_u, \mathcal{D}_{train})$. This probability is conditional on the (unseen) input vector x_u and the training set \mathcal{D}_{train} . Approximating the mapping function $\langle f \rangle$ using a machine learning algorithm assumes the use of a machine learning model M_θ , where M denotes the model, and *optionally* $\langle \theta \rangle$ denotes the parameters of the model. Thus, the probability distribution over possible labels becomes also conditioned by the chosen model, $P(y = \hat{y}|x_u, \mathcal{D}_{train}, M_\theta)$.

5.3.4 Real-valued signal

Machine learning models that are adapted to predicting real-valued output are called regression models. When using regression models, the estimated mapping function used for the prediction introduces a residual error ε between the predictions and the ground truth (5.4), (5.5).

$$y_u = \hat{y}_u + \varepsilon. \quad (5.4)$$

$$f(x_u) = \hat{f}(x_u) + \varepsilon. \quad (5.5)$$

We make the hypothesis that the residual error term ε has a Gaussian normal distribution (5.6).

$$\varepsilon \sim \mathcal{N}(\mu, \sigma^2). \quad (5.6)$$

More explicitly we will assume that the probability distribution over possible labels is as follows (5.7):

$$P(y|x_u, \mathcal{D}_{train}, M_\theta) = \mathcal{N}(\mu_\theta(x_u), \sigma^2). \quad (5.7)$$

In this context, building an estimate $\langle \hat{f} \rangle$ of the mapping function $\langle f \rangle$ boils down to estimating the model parameters $\langle \theta \rangle$. Given a set of training data, model parameters $\langle \theta \rangle$ are estimated using the maximum likelihood estimator (MLE). MLE maximizes the the probability of training data set \mathcal{D}_{train} given the model parameters $\langle \theta \rangle$ (5.8).

$$P(\mathcal{D}_{train}|\theta) = \prod_{i=1}^N P(y_i|x_i, \theta) \quad (5.8)$$

It is equivalent to finding the estimated model parameters $\hat{\theta}$ that minimizes the negative log-likelihood which is the sum of residual errors $\sum_{i=1}^N (y_i - \hat{y}_i)^2 = \sum_{i=1}^N \varepsilon_i^2$ (5.9).

$$\hat{\theta} = \underset{\theta}{\text{Argmin}} \sum_{i=1}^N (y_i - \hat{f}_\theta(x_i))^2. \quad (5.9)$$

Once optimal parameters $\hat{\theta}$ are estimated, the prediction model outputs a predicted signal estimation $\hat{y}_u = \hat{f}_{\hat{\theta}}(x_u)$ for an unseen input vector x_u . The received signal value y_u is then compared to the estimated signal value. An alert is raised if the two signals are *not similar* i.e the difference is greater then a pre-defined threshold t_p as in(5.10).

$$Alert = 1 \iff |\hat{y}_u - y_u| \geq t_p. \quad (5.10)$$

5.3.5 Categorical signal

Machine learning models that are adapted to predicting categorical output are called classification models. When using classification models, where the output is one out of C classes, we model the probability over possible labels with a categorical distribution. Let $y_{ij} = I(y_i = j)$ be the one-hot encoding of y_i . This probability is given by (5.11).

$$P(y|x_u, \mathcal{D}_{train}, M_\theta) = \prod_{j=1}^C \mu_{\theta,j}(x_u)^{I(y=j)} \quad (5.11)$$

Similarly, in order to estimate the classification model parameters $\langle \theta \rangle$, we use the maximum likelihood estimator that maximizes the probability of training data set \mathcal{D}_{train} given the model parameters $\langle \theta \rangle$ (5.12)

$$P(\mathcal{D}_{train}|\theta) = \prod_{i=1}^N P(y_i|x_i, \theta) = \prod_{i=1}^N \prod_{j=1}^C \mu_{\theta,j}(x_i)^{I(y_i=j)} \quad (5.12)$$

This is equivalent to minimizing the negative log-likelihood which is the cross entropy function (5.13)

$$\hat{\theta} = \underset{\theta}{\text{Argmin}} - \sum_{i=1}^N \sum_{j=1}^C y_{ij} \log(\mu_{\theta,j}(x_i)). \quad (5.13)$$

Once we have the optimal model parameters $\hat{\theta}$, for each unseen input vector x_u , we make a prediction in favor of the class where the probability distribution is the highest (5.14)

$$y_u = \underset{j \in [1, C]}{\text{Argmax}} (\mu_{\hat{\theta},j}(x_u)) \quad (5.14)$$

The received signal value y_u is then compared to the estimated signal value. An alert is raised if the two signals are *not similar*:

$$\text{Alert} = 1 \iff \hat{y}_u \neq y_u. \quad (5.15)$$

5.4 Validation metrics

During the training process, the training data set is used to estimate model parameters as explained in the previous section. Nevertheless, the *optimal* model parameters are computed with respect to the training set. In order to see if the built model *generalizes well*, the goal is to test it on a set of unseen data. Let $\mathcal{D}_{test} = \{(x_{u_i}, y_{u_i})\}_{i \in [1, N_{test}]}$ be the set of *unseen* data used to test. Hereafter we introduce the metrics used in order to validate the nominal model of a monitored signal $\langle y \rangle$.

5.4.1 Regression metrics for real-valued signals:

Measuring how well a regression machine learning algorithm fits the training data is called the *regression accuracy* of the algorithm (denoted as Acc^{reg}). For regression problems, it is generally measured using the coefficient of determination R^2 . The R^2 coefficient of

5. ON-BOARD INTRUSION DETECTION AND PREVENTION SYSTEM

determination is a statistical measure of how well the regression predictions approximate the observed target values. The closer it is to 1, the more accurate the prediction is. A coefficient of determination of 1 indicates that the regression predictions perfectly fit the data. Values of R^2 outside the range $[0, 1]$ can occur for instance when the wrong model was chosen, and the model fits the data worse than a horizontal hyper-plane. We can express the prediction accuracy with (5.16)

Definition 5.1. The accuracy of a regression learning algorithm is defined by (5.16).

$$\begin{aligned}
 Acc^{reg} &= R^2 \\
 &= 1 - \frac{\sum(\hat{y}_i - y_i)^2}{\sum(y_i - \mu_y)^2} \\
 &= 1 - \frac{\sigma_\varepsilon^2 + \mu_\varepsilon^2}{\sigma_y^2}
 \end{aligned} \tag{5.16}$$

Where:

- y_i : is the i^{th} ground truth target signal input,
- \hat{y}_i : is the estimated target signal given by $\hat{f}(x_i)$,
- μ_y : is the mean of the ground truth signal y ,
- $\sum(\hat{y}_i - y_i)^2$: is the residual sum of squares between the ground truth target signal y and the estimated signal \hat{y} .
- $\sum(y_i - \mu_y)^2$: is the total sum of squares,
- σ_ε^2 : is the standard deviation of the error term ε ,
- μ_ε^2 : is the mean of the error term,
- σ_y^2 : is the standard deviation of ground truth signal y .

The accuracy will serve in order to compare between different machine learning models. That is to say in order to pick and choose the best model adapted to the data. Intuitively, comparing the quality of the predictors can be based on the mean and variance of the prediction error ε . Ideally, the error has to be centered around zero (unbiased predictor) with the smallest possible variance. In fact, the more precise the prediction, the smaller should be the variance of the prediction error. The advantage of the coefficient of determination is that it can accommodate the effect of both mean and variance of the error term (eq: 5.16).

Defining an intrusion detection system based on the predictor of a real-valued target signal needs to set-up an *acceptable deviation* of the prediction that can be tolerated. The need arises from the fact that it is basically impossible to make an exact prediction $< \hat{y} >$ on the target signal $< y >$. Nevertheless, we might consider that the prediction is good enough if it is close enough to the target signal. It can be interpreted as a confidence interval on the error term $\varepsilon = (y - \hat{y})$ (Figure 5.5). Beyond this *acceptable deviation*, the received signal can be

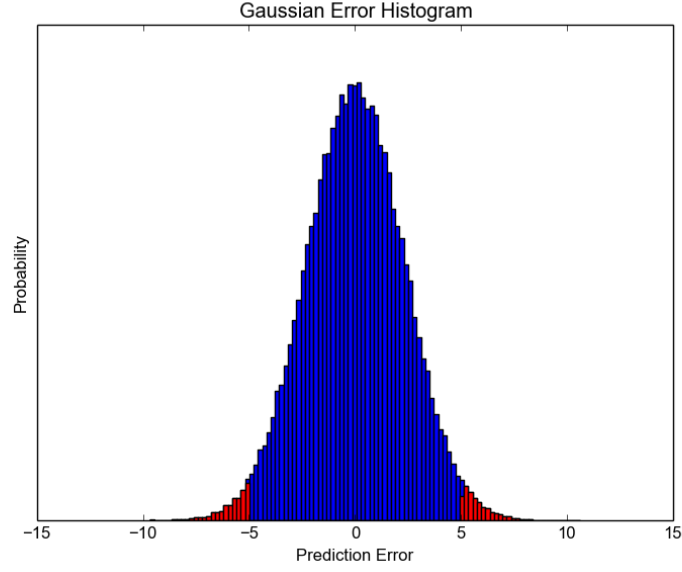


Figure 5.5: Gaussian shaped prediction error

considered way off compared to the prediction, and an alarm should be raised. This *acceptable deviation* or detection threshold t_p for the predictor defines the false positives statistically generated by the predictor (red bars in Figure 5.5). More formally we can define the *false prediction*, as in (5.17).

Definition 5.2. A received signal y_i is compared to the corresponding prediction value \hat{y}_i generated by the predictor P . We say that \hat{y}_i is a false prediction of the signal y_i if the absolute value of the error term $|\varepsilon_i| = |y_i - \hat{y}_i|$ is greater than a predefined detection threshold t_p .

$$FP_{t_p}(y_i, \hat{y}_i) = \begin{cases} 1 & \text{if } |y_i - \hat{y}_i| \geq t_p, \\ 0 & \text{if } |y_i - \hat{y}_i| < t_p \end{cases} \quad (5.17)$$

Tweaking this parameter t_p helps increase/decrease the false positives probability of the intrusion detection rule that will be defined based on this predictor. The new accuracy measure with respect to t_p is given by (5.18).

$$Acc_{t_p}^{reg} = P(|\varepsilon| < t_p) \quad (5.18)$$

Computing the accuracy $Acc_{t_p}^{reg}$ gives the false positive rate statistically generated by the detection.

5.4.2 Classification metrics for categorical signals:

When the target signal is a categorical signal ($y \in [1, C]$), we formulated the problem as a classification problem, and we can build a predictor using a classification machine learning algorithm. Measuring how well a classification machine learning algorithm fits the training

5. ON-BOARD INTRUSION DETECTION AND PREVENTION SYSTEM

data is called the *classification accuracy* of the algorithm (denoted as Acc^{cls}). The default accuracy metrics used in machine learning classification tasks is the correct classification ratio.

Definition 5.3. The classification ratio of a classification machine learning algorithm is defined by (5.19).

$$Acc^{cls} = \frac{\# \text{ correct predictions}}{\# \text{ use-cases}} \quad (5.19)$$

Where:

- $\#$ correct predictions : is the the number of correctly predicted signals $\hat{y}_i = y_i$.
- $\#$ use-cases : is the number of test vectors included in the test set \mathcal{D}_{test} .

Unlike regression, for classification, it is straightforward to define a false prediction which in this case is simply a misclassification. More formally we can define the misclassification function as the following:

Definition 5.4. A received signal y_i is compared to the corresponding prediction value \hat{y}_i generated by the predictor P . We say that \hat{y}_i is a false prediction of the signal y_i if the predicted and the ground truth signal are not of the same signal class (5.20).

$$MC(y_i, \hat{y}_i) = \begin{cases} 1 & \text{if } class(y_i) \neq class(\hat{y}_i), \\ 0 & \text{if } class(y_i) = class(\hat{y}_i). \end{cases} \quad (5.20)$$

5.5 Supervised learning algorithms

In this section, we introduce the supervised machine learning algorithms that were tested in our work.

5.5.1 K-Nearest Neighbor

Supervised neighbors-based learning is a non-parametric machine learning technique. This algorithm does not have a training step. It has only a prediction step (i.e. online monitoring). The principle that underlies this technique is to make predictions of new observations based on a number of training inputs closest in *distance* to the observation for some predefined distance measure. In principle, the number of training inputs can be predefined (in this case we talk about k-nearest neighbor learning) or vary based on the local density of points (in which case we talk about radius-based neighbor learning). In addition, the distance can be any metric measure, which opens the door to an infinity of possibilities although the Euclidean distance is the most common choice and will be used in our experiments. Neighbors-based methods are not memory efficient methods since they simply “remember” all of the training data (that can be transformed into a fast indexing structure). Nevertheless, they have been successful in a number of classification and regression tasks.

More formally, given training vectors $x_i \in R^n$, $i = 1, \dots, l$, and a label vector $y \in R^l$. Let $dist()$ be the selected distance function on R^n . As mentioned before, it is common to choose the Euclidean distance, but depending on the task other distance functions may be more adequate.

During prediction, given an unseen input vector x_u , let $\{(x_{\sigma(1)}, y_{\sigma(1)}), \dots, (x_{\sigma(n)}, y_{\sigma(n)})\}$ be a reordering of the training data such that condition (5.21) is satisfied for some permutation function σ .

$$\text{dist}(x_{\sigma(1)} - x_u) \leq \dots \leq \text{dist}(x_{\sigma(n)} - x_u) \quad (5.21)$$

5.5.1.1 KNN for regression

Neighbors-based regression can be used in cases where the data labels are continuous rather than discrete variables. The label assigned to an unseen input vector is computed based the mean of the labels of its nearest neighbors. We can identify two different neighbors regressors based on how to define the neighborhood of an unseen input:

- k-nearest neighbors, where the prediction is based on the k nearest neighbors of each unseen input, where k is an integer value specified by the user. In this case, the label is computed with the following (5.22).

$$y_u = \frac{1}{k} \sum_{j=1}^k y_{\sigma(j)} \quad (5.22)$$

- radius nearest neighbors, where the prediction is based on the neighbors within a fixed radius r of the unseen input, where r is a floating-point value specified by the user and is used to compute the set of neighbors training vectors $Neighbors(x_u) = \{x_{\sigma(j)}, \text{dist}(x_{\sigma(j)} - x_u) \leq r\}$. In this case, the label is computed with the following (5.23).

$$y_u = \frac{1}{\text{card}(Neighbors(x_u))} \sum_{j=1}^{\text{card}(Neighbors(x_u))} y_{\sigma(j)} \quad (5.23)$$

5.5.1.2 KNN for classification

Neighbors-based classification can be used in cases where the target variable is discrete. Classification of an unseen vector x_u is computed from a simple majority vote of the nearest neighbors of each point: an unseen input vector is assigned the data class which has the most representatives within the nearest neighbors of the vector. Similar to regression, we can identify two different neighbors classifiers based on how to define the neighborhood of an unseen vector:

- k-nearest neighbors classification, where the prediction is based on the k nearest neighbors of each unseen point, where k is an integer value specified by the user. In this case, the label is computed with the following (5.24).

$$y_u = \underset{c}{\text{Argmax}} \left(\frac{\text{card}\{y_{\sigma(j)} = c, j \leq k\}}{k} \right) \quad (5.24)$$

- radius nearest neighbors classification, where the prediction is based on the neighbors within a fixed radius r of the unseen input vector, where r is a floating-point value specified by the user and is used to compute the set of neighbors training vectors

5. ON-BOARD INTRUSION DETECTION AND PREVENTION SYSTEM

$Neighbors(x_u) = \{x_{\sigma(j)}, dist(x_{\sigma(j)} - x_u) \leq r\}$ In this case the label is computed with the following (5.25).

$$y_u = \underset{c}{\text{Argmax}} \left(\frac{\text{card}\{y_{\sigma(j)} = c, j \leq \text{card}(Neighbors(x_u))\}}{\text{card}(Neighbors(x_u))} \right) \quad (5.25)$$

Remark 8. Note that for both classification and regression, we introduced the KNN with uniform weights. It can be advantageous, under some circumstances, to weight vectors such that nearby points contribute more to the regression/classification than faraway points. This may have a positive impact on the overall accuracy of the learning algorithm although the weights will constitute hyper-parameters that will have to be tweaked.

Remark 9. In this thesis, we used k-nearest neighbor learning technique although the approach would be the same for radius-based KNN.

Remark 10. Note that the goal is to build a detection rule that captures the nominal behavior of a target signal with regards to the other input signals. The goal is also to build a rule that can be implemented in an embedded system. The KNN algorithm needs to have access to the training set \mathcal{D}_{train} at the prediction time (i.e. online monitoring) which makes it not adapted to be implemented in an embedded system as it will need significant memory resources in order to store the training set. Nevertheless, we used the KNN algorithm in order to have a reference accuracy to which we can compare other algorithms, as it gives very precise local approximations.

5.5.2 Decision tree

Decision Trees are a non-parametric supervised learning method used for classification and regression tasks. During the learning phase, the algorithm generates a set of *if-then-else* decision rules through a recursive space partitioning procedure inferred from data features. These rules are organized in a tree-like structure and then used to predict the value of a target variable. This tree is then used during the prediction phase in order to assign a label for an unseen input vector. The Tree is composed of interior nodes and leaf nodes. Each interior node corresponds to one of the data features. Edges to children nodes represent each of the possible values of the feature values. Each leaf node represents a value of the target variable given the values of the features of the input vector from the root node to the leaf node. Figure 5.6 illustrates an example of a decision tree.

Constructing a decision tree (during training phase) consists in recursively partitioning the training data space (until requested tree depth is reached) such that the samples with the same labels are grouped. Decision tree algorithms usually work in a top-down fashion, by choosing a feature at each step that best splits the set of input data.

Given training vectors $x_i \in R^n$, $i = 1, \dots, l$, and a label vector $y \in R^l$. Let the set of data samples at node m be represented by D_m . The goal is to split the data D_m into two subsets D_m^{left} and D_m^{right} by building a split rule $\Phi_m = (j, t_m)$ such as (5.26).

$$\begin{cases} D_m^{left}(\Phi_m) = (x, y) | x_j \leq t_m \\ D_m^{right}(\Phi_m) = D_m \setminus D_m^{left}(\Phi_m) \end{cases} \quad (5.26)$$

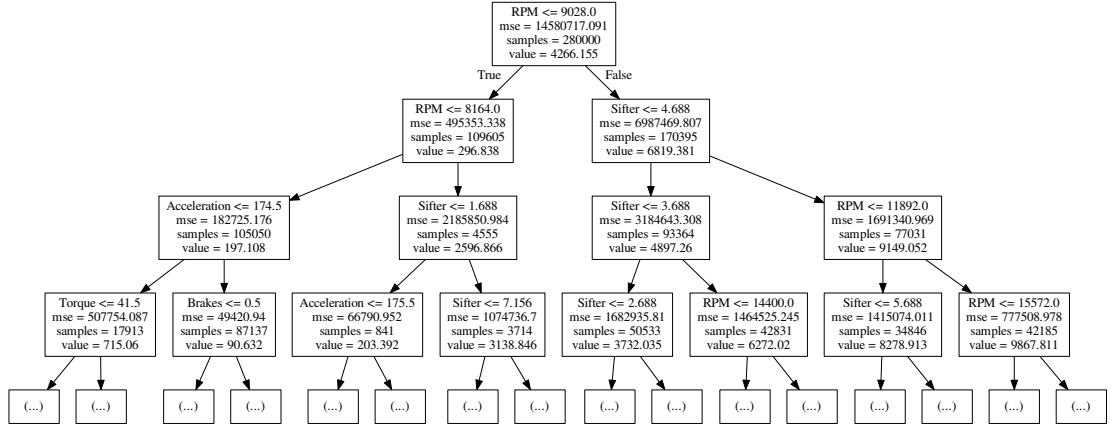


Figure 5.6: Example of a decision tree

For each node, we then compute an impurity function $H(D_m)$ that indicates *how pure or impure* the subset D_m is. It measures the homogeneity of the target variable within the subsets. The choice of the impurity function H depends on the type of target variable (Regression or classification) and will be detailed in section 5.5.2.1 and section 5.5.2.2. Finding the optimal split rule Φ_m consists in maximizing the information gain $Gain$ after the split ((5.27)).

$$Gain(D_m, \Phi_m) = H(D_m) - \left(\frac{N_m^{left}}{N_m} H(D_m^{left}(\Phi_m)) + \frac{N_m^{right}}{N_m} H(D_m^{right}(\Phi_m)) \right) \quad (5.27)$$

Or minimizing The function G (5.28).

$$G(D_m, \Phi_m) = \frac{N_m^{left}}{N_m} H(D_m^{left}(\Phi_m)) + \frac{N_m^{right}}{N_m} H(D_m^{right}(\Phi_m)) \quad (5.28)$$

The optimal split rule is then given by (5.29).

$$\Phi_m^* = \underset{\Phi_m}{\operatorname{Argmin}}(G(D_m, \Phi_m)) \quad (5.29)$$

Thus, building a decision tree from a set of training data \mathcal{D}_{train} consists in finding the best split for subsets $D_m^{left}(\Phi_m^*)$ and $D_m^{right}(\Phi_m^*)$ in a recursive fashion until the requested tree depth is reached, the sets are *pure* or the size of the set is $N_m = 1$ or $N_m < N_{min}$.

5.5.2.1 Regression Trees

Decision trees where the target variable y can take continuous values (typically real numbers) are called regression trees. In this case, a common criteria to minimize used for determining the best split rule is the Mean-Squared-Error, which minimizes the L2 error using mean values at leaf nodes. Let D_m be the set of N_m observations at node m . We denote by c_m the mean of the target value computed over the input vectors of node m (5.30).

$$c_m = \frac{1}{N_m} \sum_{(x_i, y_i) \in D_m} y_i \quad (5.30)$$

The impurity function used is the mean squared error (5.31).

$$H(D_m) = \frac{1}{N_m} \sum_{(x_i, y_i) \in D_m} (y_i - c_m)^2 \quad (5.31)$$

5.5.2.2 Classification Trees

Tree models where the target variable y can take a discrete set of values are called classification trees. In these tree structures, leaves represent class labels. When dealing with a classification task with K classes, the target variable takes values in $[1, K]$. Let D_m be the set of N_m observations at node m . We denote $p_{m,k}$ the proportion of class k observations in node m (5.32).

$$p_{m,k} = \frac{1}{N_m} \sum_{x_i \in D_m} I(y_i = k) \quad (5.32)$$

A common impurity function used is the cross-entropy defined as follows (5.33).

$$H(D_m) = - \sum_k p_{m,k} \log(p_{m,k}) \quad (5.33)$$

Remark 11. Note that D_m is a subset of the training data set D , that can be the result of a previous data partitioning $D_{\text{parent}(m)}^{\text{left/right}}$ at parent node level for some optimal partitioning rule $\Phi_{\text{parent}(m)}^*$.

Remark 12. The decision tree algorithm is very efficient in making predictions (i.e. during monitoring) once the tree structure has been constructed. In fact, given the decision rules structured in a tree structure, it can quickly go through the tree in order to assign a label to the unseen input vector.

5.5.3 Artificial Neural Network

Unlike previously introduced machine learning algorithms, Artificial Neural Network is a parametric supervised learning technique inspired by the biological neural network. These types of algorithms are usually modeled as an interconnected group of artificial neurons with a graph where nodes represent neurons and arcs represent connections between neurons.

The mathematical model of the neuron is an activation function also called perceptron. Each perceptron has multiple inputs and one output. The output is a non-linear function of a weighted sum of the inputs. A special sub-class of ANN is the feed-forward neural networks. In this sub-class, connections between the nodes do not form a cycle. Under this assumption, the information moves in only one direction, *forward*, from the input nodes, through the hidden nodes (if any) and to the output nodes. Figure 5.7 is an example of a feed-forward neural network. The leftmost layer, known as the input layer, consists of a set of neurons $\{s_i | s_1, s_2, \dots, s_m\}$ representing the input features. Each neuron in the hidden layer transforms the values from the previous layer with a weighted linear summation $w_1 s_1 + w_2 s_2 + \dots + w_m s_m$, followed by a non-linear activation function $g(\cdot) : R \rightarrow R$. The output layer receives the values from the last hidden layer and transforms them into output values.

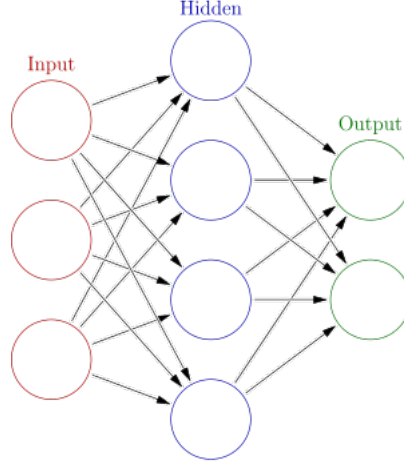


Figure 5.7: Illustration of a feed-forward artificial neural network structure with an input layer, one hidden layer, and an output layer.

Multi-layer Perceptron (MLP) is a feed-forward neural network with at least one hidden layer of perceptrons. Given training vectors $x_i \in \mathbf{R}^n$, $i = 1, \dots, l$, and a label vector $y \in \mathbf{R}^l$, it can learn to approximate a non-linear mapping function $f(x_i) \rightarrow y_i$ for either classification or regression task.

Multiple non-linear activation functions were used in different applications of neural networks. The most popular ones are:

- Logistic activation function

$$g(x) = \frac{1}{1 + e^{-x}} \quad (5.34)$$

- Rectified linear unit (Relu):

$$g(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \quad (5.35)$$

- Hyperbolic tangent (tanh):

$$g(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (5.36)$$

Given a set of training vectors $\mathcal{D}_{train} = \{(x_1, y_1), (x_2, y_2), \dots, (x_l, y_l)\}$ where $x_i \in \mathbf{R}^n$ and $y_i \in \mathbf{R}$, a one hidden layer one hidden neuron MLP (Figure 5.8) learns the function (5.37).

$$f(x_i) = g(W_1^T x_i + b_1) = y_i \quad (5.37)$$

Where $W_1 \in \mathbf{R}^n$ and $b_1 \in \mathbf{R}$ are model parameters. W_1 represent the weights of the input layer, and b_1 represent a bias term added to the hidden layer. $g : \mathbf{R} \rightarrow \mathbf{R}$ is the activation function, typically one of the functions introduced previously in this section.

Putting all together, suppose we have a complex MLP structure (Figure 5.9) with n input feature, and k hidden layers. Each hidden layer j has p_j ($j \in [1, k]$) neurons and computes a

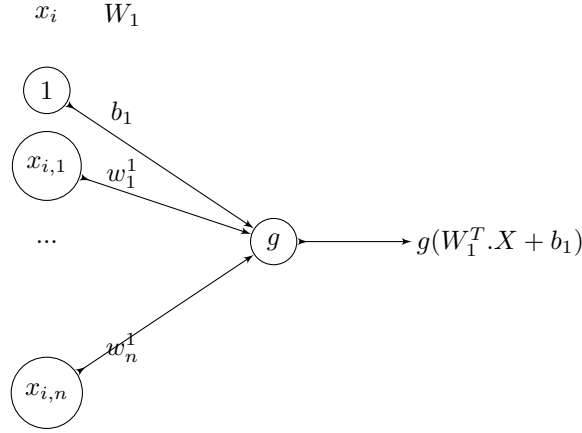


Figure 5.8: One hidden neuron

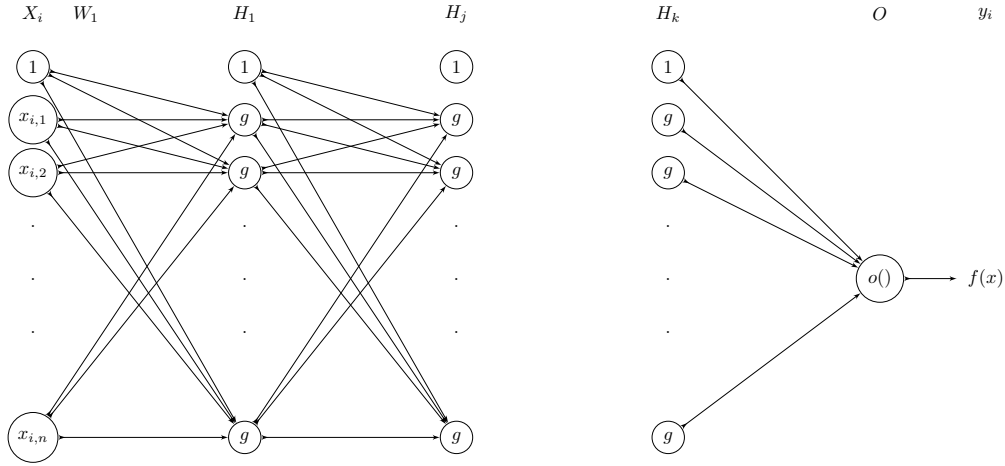


Figure 5.9: Multilayer perceptron structure

vector of values $H_j = [h_{j,1}, \dots, h_{j,p_j}]^T$. Values of layer (j) are computed based on values of layer $(j-1)$. We denote W_j the matrix of weights between layer $(j-1)$ and layer (j) . W_j is a $(p_{j-1} \text{ by } p_j)$ matrix, and we denote B_j the vector of bias terms ($B_j = [b_{j,1}, \dots, b_{j,p_{j-1}}]$). The following set of equations holds (5.38)

$$\begin{cases} H_1 = g(B_1 + (W_1)^T X) \\ H_j = g(B_j + (W_j)^T H_{j-1}) & \text{for } 2 \leq j \leq k \\ f(x) = o(B_k + W_k^T H_k) \end{cases} \quad (5.38)$$

The MLP structure with the non-linear activation function as perceptron and the weights yields a set of mathematical functions. The hypothesis when using a particular MLP structure is that the *ground-truth* mapping function will be approximated in the set of functions that can be represented by the MLP structure.

5.5.3.1 MLP for Regression

Multi-layer Perceptron can be used for regression when the target variable y can take continuous values (typically real numbers). In this case, the activation function of the output layer is the identity function. Moreover, it is customary to make the hypothesis that the error function follows a Gaussian distribution. Which leads us (as explained in section 5.4) to define the squared error as the error function to be minimized (5.39) in order to find the optimal weights.

$$\begin{aligned}\hat{W} &= \underset{W}{\operatorname{Argmin}} \left(\sum_{i=1}^N (y_i - \hat{y}_i)^2 \right) \\ &= \underset{W}{\operatorname{Argmin}} \left(\sum_{i=1}^N \varepsilon_i^2 \right)\end{aligned}\tag{5.39}$$

$$\text{where: } \hat{y}_i = f_W(x_i)$$

Starting from initial random weights, multi-layer perceptron (MLP) minimizes the error function by repeatedly updating these weights. After computing the error, a backward pass propagates it from the output layer to the previous layers, providing each weight parameter with an update value meant to decrease the error.

5.5.3.2 MLP for Classification

Multi-layer Perceptron can be used for classification tasks when the target variable y takes discrete/categorical values. For instance, the target variable y can take binary values in $\{0, 1\}$. In this case the output $f_W(x)$ is passed through the logistic function $g(z) = 1/(1 + e^{-z})$ that outputs values between zero and one. These values are interpreted as probabilities for the corresponding class. A threshold (set to 0.5 for instance), would assign samples of outputs larger or equal 0.5 to the positive class (1), and the rest to the negative class (0). On the other hand, if the target variable represents more than two classes, typically in $[1, K]$, with $K \geq 3$, then the output $f_W(x)$ should be a vector $[z_1, \dots, z_K]$ of size K , where each component represents one of the K classes of the target variable. In order to make a prediction in favor of one of the classes, it is passed through the Softmax function which is expressed as (5.40).

$$\operatorname{Softmax}(f_W(x))_i = \frac{\exp(z_i)}{\sum_{l=1}^K \exp(z_l)}\tag{5.40}$$

The Softmax function transforms the input vector into a probability vector which components sum up to one (5.41). The i^{th} component represents the probability that the input vector x results in a target variable y in class $i \in [1, K]$. The predicted class is then the one with the highest probability.

$$\sum_{i=1}^K \operatorname{Softmax}(f_W(x))_i = 1\tag{5.41}$$

5. ON-BOARD INTRUSION DETECTION AND PREVENTION SYSTEM

The error function for classification is the Cross-Entropy, which in the binary case is given by (5.42).

$$\hat{W} = \underset{W}{\text{Argmin}}(-y \ln \hat{y} - (1 - y) \ln (1 - \hat{y})) \quad (5.42)$$

Remark 13. *Note that the activation functions introduced previously and that constitute the perceptron are non-convex functions. It follows that MLP with hidden layers has a non-convex error function defined as the difference between the desired output of the MLP structure and the actual output. This holds both for regression and classification MLP structures. The optimization process uses the learning procedure called back-propagation [111] with a minimization algorithm such as stochastic gradient descent (SGD). This is a limitation, as during the minimization process SGD can run into a local minimum. However, in practice gradient descent usually works reasonably well. For this reason, MLP approximation requires multiple runs with random initialization. The advantage nevertheless is that the MLP structure based on the non-linear functions as defined previously can learn non-linear dependencies.*

5.6 Data collection and feature engineering

5.6.1 Experimental set-up

In order to provide training vectors, the best way is to collect data directly from a real vehicle. For this purpose, we prepared a CAN acquisition device. The device is composed of a Raspberry-Pi¹ with additional CAN-Bus hardware module running a Linux kernel with *SocketCAN* [75] drivers. The CAN hardware communicates with the Raspberry-Pi over SPI² communications. We made sure that the Raspberry-Pi has enough storage space in order to save the logs locally.

We equipped a vehicle with the acquisition device connected directly to different CAN buses in order to have direct access to *all* sensor information, although not all of them will be used during the training phase. Figure 5.10 gives an overview of the prepared experimental set-up.

Remark 14. *Note that one of the issues that may arise with regards to our set-up is that receiving data with a Raspberry-Pi through SPI may lead to issues in receiving the total data transmitted over the CAN network if the band usage of the CAN network is too high. In our case, the throughput of the CAN network configuration allows 500 kbit/s. The CAN Hardware uses the MCP2515 [8] Stand-Alone CAN Controller With High-speed SPI Interface (configured at 10 MHz). The raspberry-Pi board also implements a high-speed SPI interface, which means that the SPI interface in our case allows much more data throughput than the CAN interface.*

5.6.2 Data collection

Data collection procedure is quite complicated. It is essential to have data representing all situations of the vehicle in order to be able to capture the nominal behavior in all those use-cases. We identified two factors that could potentially influence the collected data.

¹Raspberry-Pi: <https://www.raspberrypi.org/>

²SPI: The Serial Peripheral Interface is a synchronous Master/Slave serial communication interface specification used for short distance communication, primarily in embedded systems.

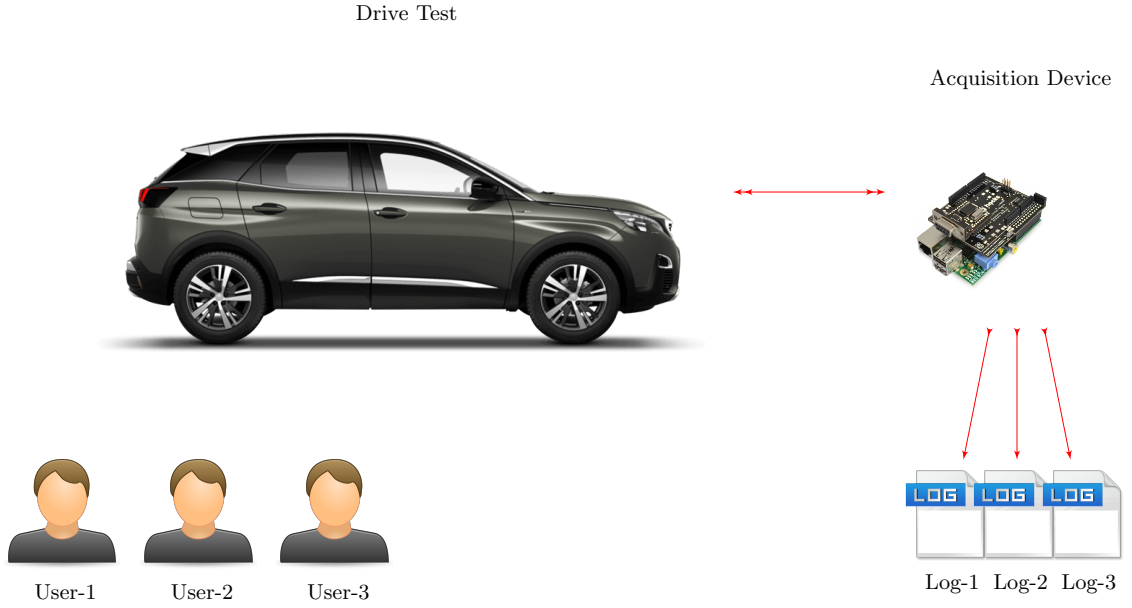


Figure 5.10: Illustration of the data acquisition platform

1. *The Driving circuit:* this includes for instance if the vehicle is on a highway, city circuit, parking lot, This nature of the circuit can put the vehicle into multiple different legitimate states that need to be captured.
2. *The driving behavior:* this can include different driving styles and different states and moods of the driver that naturally influence the input sensors and commands triggered by the driver.

We collected CAN traces from one vehicle for three different drivers, driving in different circuits for about 90 minutes each. Circuits consisted of multiple driving conditions including city driving, vehicle parking, highway driving, During those data collections, drivers were asked to drive normally but also to perform rare but legitimate scenarios like activating cruise control, activating lane keep assist, activating emergency breaking, For safety reasons, no attacks were performed during data collections step.

The total amount of data collected is about 10^6 CAN messages for each driver. Moreover, when receiving the CAN messages, for each received frame, we log the reception time-stamp, identifier, and payload content. In order to verify the integrity of the collected data and make sure that messages were not lost, we run some integrity verification tests. Integrity verification tests are designed to make sure that the manufacturer protocol with regards to message syntax (mainly Identifiers and DLCs) are respected as well as the periodicity of the messages is also respected.

5. ON-BOARD INTRUSION DETECTION AND PREVENTION SYSTEM

5.6.3 Feature engineering

After raw data acquisition, the second step consists in preparing the data for processing. In this step, the goal is to select and arrange the features in a form that would be useful during the training step. Each CAN identifier sent over the CAN bus has a payload that is composed of one or multiple signals. Recall that a signal is a piece of information (sensor value, ECU state, counter, checksum, ...) that can occupy one or multiple bits or bytes depending on the nature of the information. Extraction of signals requires the knowledge of the proprietary protocol of the car manufacturer. Signals included in the payload for safety reasons, like checksums, process counters, duplicated signals, are checked by safety functions and problems with those signals, if any, would be handled by appropriate safety mechanisms. Thus, they are not relevant for this task and therefore are not selected. Typically we are interested in physical sensor values like speed, acceleration, RPM, etc. ...

The set of those signals defines the state of the vehicle and constitutes the input features that are relevant for learning the normal behavior and evolution of the car states. The second selection criterion is the relevance with respect to the target signal. In fact, the dimensionality of the training vectors equals the number of selected signals. However, in general, machine learning algorithms do not work well with high dimensional inputs. Indeed, as input vectors dimensions grow, the performance deteriorates, due to the curse of dimensionality. As a result, we choose to select only a signal with a high correlation with the target signal. For instance, the engine oil temperature does not influence the vehicle speed, thus would not be selected when building a predictor for the speed signal. On the other hand, the acceleration of the vehicle is highly correlated to the speed of the vehicle. Thus it will be selected as an input to predict the speed. Using this selection criterion, we can guarantee that signals that can *explain the most* the target signal are used for prediction.

Signals are featured in the form of a matrix where columns represent signals and lines represent signal values evolution over time. For each received CAN message that holds selected signal, a new line is added to the matrix where all signals keep their previous values/states except the one that has just been received. Figure 5.11 gives more details about how to construct the features matrix.

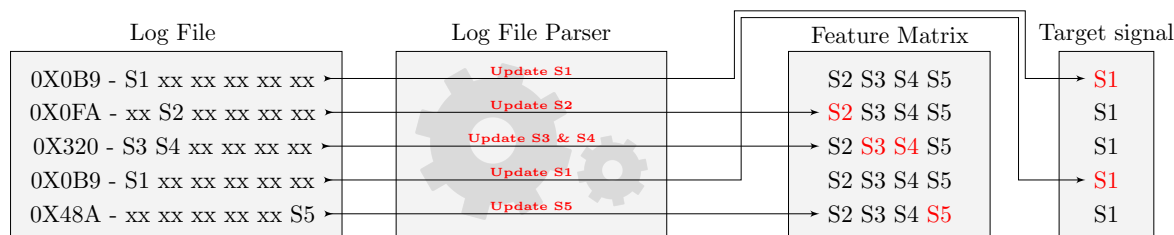


Figure 5.11: Parsing the log file and building the training data.

5.7 Experimental validation and discussion

In order to validate the approach, we conduct some experiments to predict two target signals, one of each type (categorical and real-valued), using five selected input signals. To this end, a total of six signals were extracted from the CAN logs. For each target signal, the remaining five were used as input features.

- Speed, is a real-valued signal sent from the Electronic Stability Program (ESP) and that is generated by an embedded speed sensor. Hereafter denoted $\langle S_{Speed} \rangle$.
- Acceleration, is a real-valued signal that is sent from the Electronic Stability Program (ESP) and generated by an acceleration sensor. Hereafter denoted $\langle S_{Acceleration} \rangle$.
- Engine rotational speed expressed in revolutions per minute (RPM), is a real-valued signal sent by the Engine Control Module (ECM). Hereafter denoted $\langle S_{RPM} \rangle$.
- Torque, is a real-valued signal sent by the Engine Control Module (ECM) that contains the engine torque. Hereafter denoted $\langle S_{Torque} \rangle$.
- Shifter position, is a categorical signal sent by the Electronic Shifter Module (ESM), that indicates the gear lever position. Hereafter denoted $\langle S_{Shifter} \rangle$.
- Brake lights command is a categorical signal that is sent from the Electronic Stability Program (ESP) module to control brake lights. Hereafter denoted $S_{\langle Brake-lights \rangle}$.

Experimental validation is conducted in two steps:

1. First we train and evaluate the detection rules using collected data and without performing any attacks. In terms of intrusion detection, this step gives us the True Negative rate. The True Negative rate is defined hereafter as the accuracy (Acc) of the supervised learning algorithm, which was formally introduced in the section 5.4. The False Positive rate is then derived from the accuracy and equals $(1 - Acc)$.
2. Then we conduct an attack campaign and measure how many of the performed attacks are detected. This step gives us the True Positive rate and the False Negative rate.

Table 5.1 defines the metrics that will be used in the sequel.

Table 5.1: Detection metrics

	Detected	Not-detected
No-attack	$FP = 1 - Acc$	$TN = Acc$
Attack	TP	FN

5.7.1 Predicting a real-valued signal

5.7.1.1 Speed signal

For regression problems, we chose to validate the approach we described in previous sections on a signal that is important from a safety standpoint. The speed information is sent by the

5. ON-BOARD INTRUSION DETECTION AND PREVENTION SYSTEM

Electronic Stability Program (ESP) over the CAN bus for the other ECUs to be used in other functions. Besides being displayed for user-information, it is used to compute the effort to be applied on brakes when emergency brakes are activated, to decide when to activate airbags in case of an accident, also to decide if the car doors should be open or closed, and whether or not to accept diagnostics commands and a lot of other safety-critical functions.

5.7.1.2 Capturing nominal behavior of the speed signal

We use the previously introduced supervised learning algorithms in order to capture the nominal behavior of the speed signal. We use five different input signals featured in an input vector as in (5.43).

$$(S_{Acceleration}, S_{RPM}, S_{Torque}, S_{Shifter}, S_{Brake-lights}) \quad (5.43)$$

In order to be able to predict the label of the speed signal $< S_{Speed} >$ of an unseen input vector, the problem is to estimate the probability expressed in (5.44).

$$P(S_{speed} | (S_{Acceleration}, S_{RPM}, S_{Torque}, S_{Shifter}, S_{Brake-lights})) \quad (5.44)$$

In the performed experiments, the goal is to compare between different machine learning algorithms, as each algorithm has a different way of capturing dependencies between input features and the target signal. We used a data set of 10^6 input vectors from each drive test. The data set was split into a training set and a test set of 0.7 and 0.3 size ratio respectively. All experiments are done with the **Scikit-learn** library [105].

In the first experiment, we train and evaluate detection rules for each driver separately. We used four types of machine learning algorithms: k -nearest neighbors (KNN), Decision Tree, Neural-Network with logistic perceptron and Neural-Network with rectified linear unit (Relu) perceptron.

Remark 15. *Note that for each type of machine learning algorithms, there are some tuning parameters to be set. For example, when using KNN, we have to choose the number of neighbors to be considered, for Decision Tree, we have to choose the depth of the tree, and for Neural-Network we have to choose the number of hidden layers and neurons in each layer. These parameters have an effect on the complexity of the prediction and give the algorithms more capability to capture more complex dependencies.*

Remark 16. *In order to set-up the learning algorithms parameters, we adopt the following approach: First, we start with the less complex configuration, and then we change the parameters progressively and compare the evolution of the accuracy. For instance, this consists in increasing the depth of a decision tree or in increasing the number of neurons and layers for neural networks and capture the accuracy then pick and choose the best configuration.*

KNN: As mentioned before, the KNN algorithm is used in this context in order to establish a reference accuracy to compare other algorithms as it is expected to give precise predictions based on local approximations. It cannot be implemented in an embedded system because of its memory needs. We tested different instances of the algorithm by progressively increasing

the number of neighbors to be used for the prediction. The goal is to find the optimal number of neighbors for which the accuracy is maximized. Figure 5.12 and Figure 5.13 report the evolution of the accuracy of the prediction model and the True Negative rate respectively. Figure 5.14 and Figure 5.15 report the evolution of the prediction error mean and standard deviation respectively.

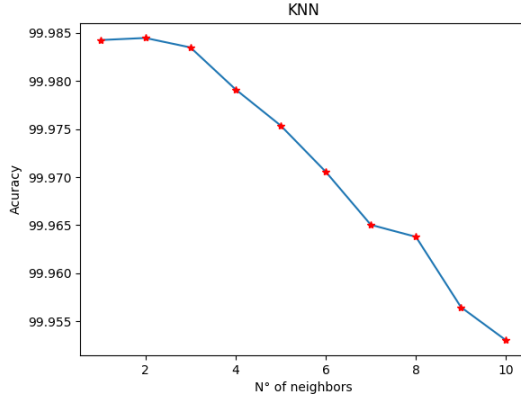


Figure 5.12: Accuracy ($Acc^{reg} \%$) of KNN algorithm as a function of the number of neighbors

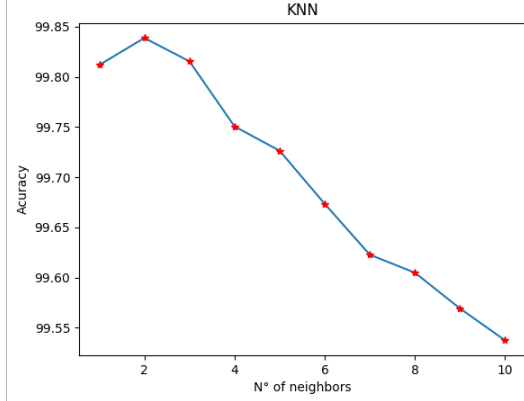


Figure 5.13: Accuracy ($Acc_{tp}^{reg} \%$) of KNN algorithm as a function of the number of neighbors

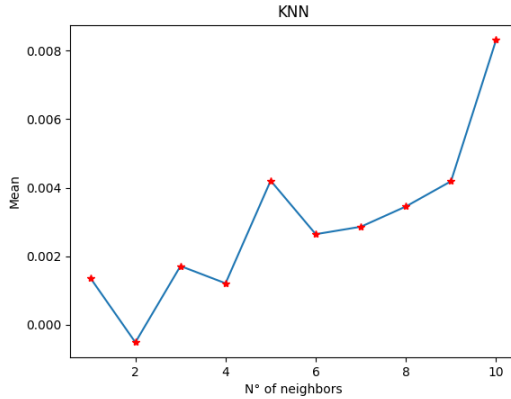


Figure 5.14: Mean of the prediction error μ_ϵ of KNN algorithm as a function of the number of neighbors

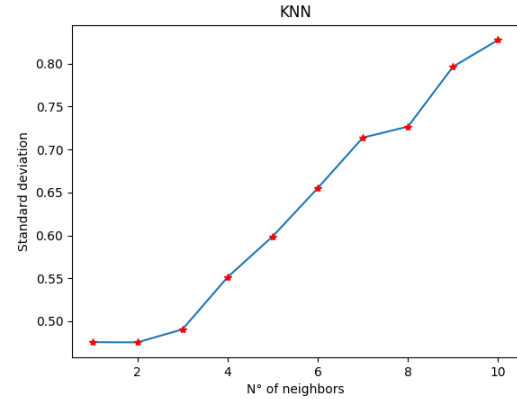


Figure 5.15: Standard deviation of the prediction error σ_ϵ of KNN algorithm as a function of the number of neighbors

These results show that as the number of considered neighbors increases, the accuracy (respectively the True Negative rate) decrease. The tendency is reflected in the mean and standard deviation evolution. As the number of neighbors increases, the predictor becomes biased and imprecise.

Linear Regression: Training a detection rule to monitor the speed signal S_{Speed} with linear regression is straight-forward. There are no hyper-parameters to change that can increase the algorithm capacity to capture more dependencies. The dependencies that can be captured

5. ON-BOARD INTRUSION DETECTION AND PREVENTION SYSTEM

with Linear-Regression are linear regressions. Results are given in Table 5.2 for the one driver. These results are averaged over ten runs each time with a random selection of training and test vectors.

Table 5.2: Linear Regression detection rule [results are averaged over 10 runs]

Algorithm	$Acc^{reg}[\%]$	$Acc_{t_p}^{reg}[\%]$	Error Mean μ_ϵ	Error Standard deviation σ_ϵ
Linear Regression	85.51	36.38	-0.0055	14.53

Results show that the linear regression completely fails at predicting with high accuracy the speed signal. Even though the statistical accuracy (Acc^{reg}) of the algorithm exceeds 85%, we can see that within the interval $[-5, 5] km/h$, the True positive rate is about 36% which is very low. This can be explained by the fact that the standard deviation is high (14 km/h). We conclude that the set of input features and the target signal do not have strong linear dependencies that allow to make a precise prediction on the target signal given the input vector.

Decision Tree: We evaluate the decision tree algorithm against data collected from the first driver. Gradually increasing the tree depth allows the tree structure to capture more data dependencies. The Accuracy (Acc^{reg}) of the prediction algorithms are represented in figure 5.16. The accuracy in the interval $[-5, 5] km/h$ is given in figure 5.17. The error mean $< \mu_\epsilon >$ and standard deviation $< \sigma_\epsilon >$ are given respectively in figure 5.18 and figure 5.19.

The first observation that we can make is that as the tree depth increases, the accuracy Acc^{reg} of the machine learning algorithm improves. With a tree depth of only 2 we already have 88% accuracy. It exceeds 99% accuracy for a tree depth of 10 and 99.9 for a tree depth of 40. The accuracy $Acc_{t_p}^{reg}$ at $[-5, 5] km/h$ which is the True Negative rate of the detection rule, also improves as the tree depth increases. It is nevertheless always below the accuracy of the machine learning algorithm. For instance, the accuracy of a tree with a tree depth of only 2 equals to 53%. This can be explained by the evolution of the mean and standard deviation of the prediction error. In fact, in figure 5.19 we can see that the standard deviation at the start is quite significant (12 km/h), this explains the low True Negative rate (high False Positive) as a high number of predictions lie outside the acceptable deviation interval $[-5, 5] km/h$. The standard deviation nevertheless converges to 0.4 km/h as the tree depth reaches 40. At the same time, the mean also stabilizes at around zero.

Neural Network with Logistic perceptron: Similar to the previous supervised learning algorithms, we use Neural-Network with logistic perceptron to capture dependencies between input features and evaluate how well it can predict the speed signal. However, setting up the structure of the neural network is not intuitive. We have to specify the number of hidden layers as well as the number of neurons in each layer. To the best of our knowledge, there are no specific guidelines to follow in order to define the structure. Our approach is purely empirical. We start with the simplest possible structure which is a neural network with one hidden layer that contains one neuron. Then at first, we start to progressively increase the

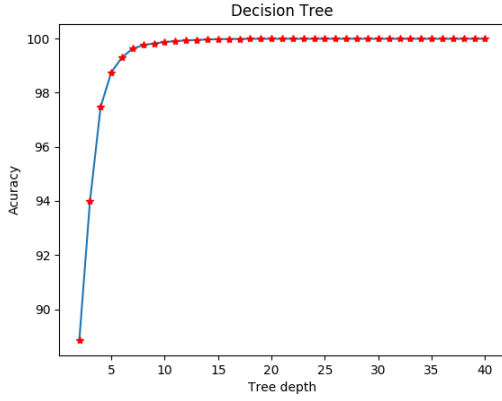


Figure 5.16: Accuracy (Acc^{reg} %) of decision tree algorithm as a function of the tree depth

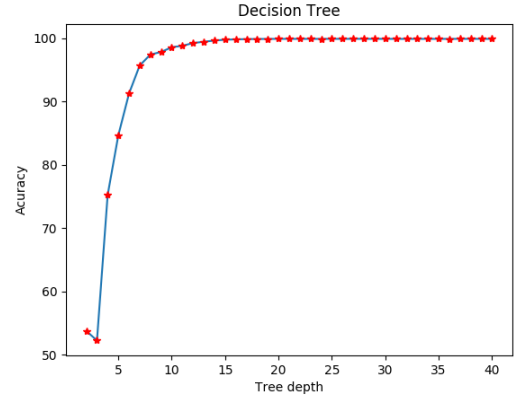


Figure 5.17: Accuracy ($Acc_{t_p}^{reg}$ %) of decision tree algorithm as a function of the tree depth ($tp = \pm 5$ km/h): True Negative.

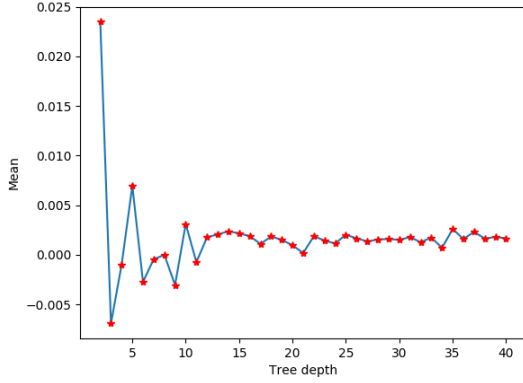


Figure 5.18: Mean of the prediction error μ_ϵ of decision tree algorithm as a function of the tree depth

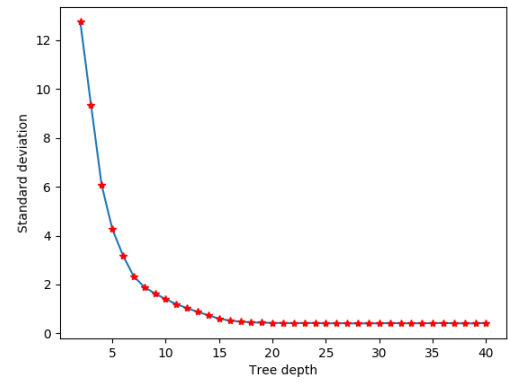


Figure 5.19: Standard deviation of the prediction error σ_ϵ of decision tree algorithm as a function of the tree depth

number of neurons in the layer until we reach optimal accuracy. Then we proceed with the same strategy in defining the number of layers.

Remark 17. Recall that the error function to be minimized by during the training of a neural network structure is non-convex. This means that the optimization algorithm may run into a local minimum. In order for the optimization process to come to an end at some point, we need to set-up a tolerance parameter against which we check the relative error improvement. If the prediction error does not improve more than the tolerance, then the convergence is considered to be reached, and the training stops. In our experiments we set-up this parameter to 10^{-4} .

Figure 5.20 and figure 5.21 show the accuracy of the learning algorithm and the true negative rate evolution as the number of neurons in the first layer increase respectively. These results show that as the number of neurons in the first layer increases, the accuracy (respectively the True Negative rate) increases. The accuracy reaches 99%, and the true

5. ON-BOARD INTRUSION DETECTION AND PREVENTION SYSTEM

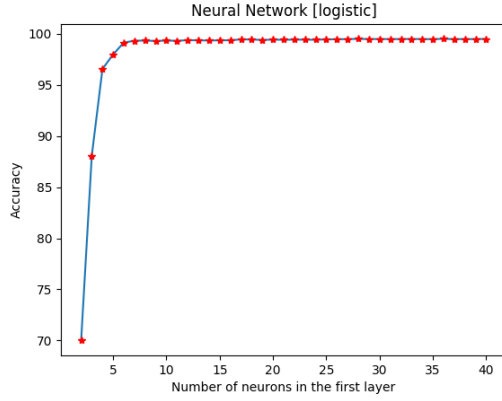


Figure 5.20: Accuracy (Acc^{reg} %) of Neural-Network with logistic perceptron algorithm as a function of the number of neurons in the first layer.

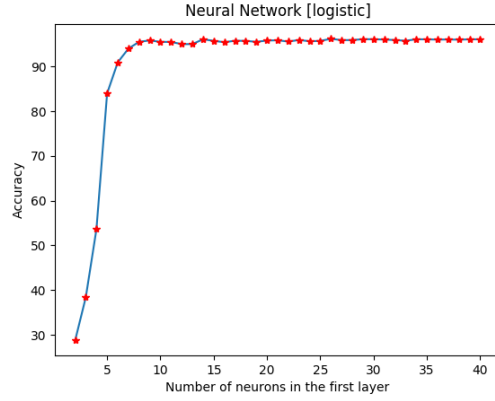


Figure 5.21: Accuracy ($Acc_{t_p}^{reg}$ %) of Neural-Network with logistic perceptron algorithm as a function of the number of neurons in the first layer.

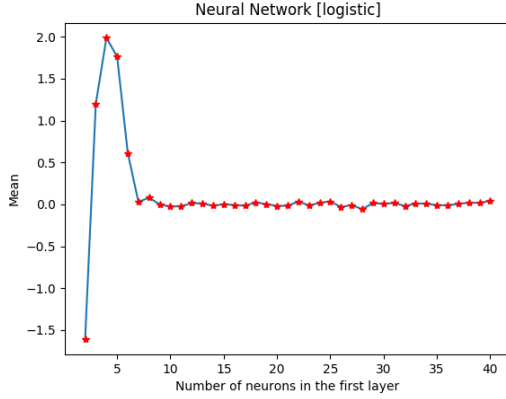


Figure 5.22: Mean of the prediction error μ_ε of Neural-Network with logistic perceptron algorithm as a function of the number of neurons in the first layer.

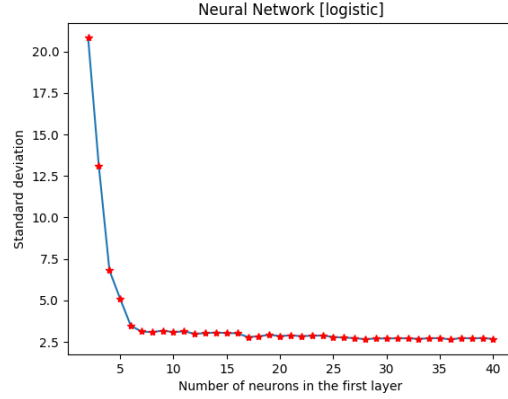


Figure 5.23: Standard deviation of the prediction error σ_ε of Neural-Network with logistic perceptron algorithm as a function of the number of neurons in the first layer.

positive rate reaches 95%. These accuracies stabilize at nearly ten neurons. The tendency is reflected in the mean and standard deviation evolution represented respectively in figure 5.22 and figure 5.23. The mean error stabilizes around 0 km/h , and the standard deviation around 2.5 km/h . Thus, if the structure holds more than ten neurons in the first layer, the complexity of the algorithm is increased, but the accuracy does not get better.

Neural Network with Relu perceptron: In this test, we use Neural-Networks with rectified linear units (Relu) perceptron to capture dependencies between input features and evaluate how well it can predict the speed signal. Similarly, the goal is to see how the accuracy evolves when the structure (number of layers and neurons) is modified.

Figure 5.24 and figure 5.25 show the accuracy of the learning algorithm and the true

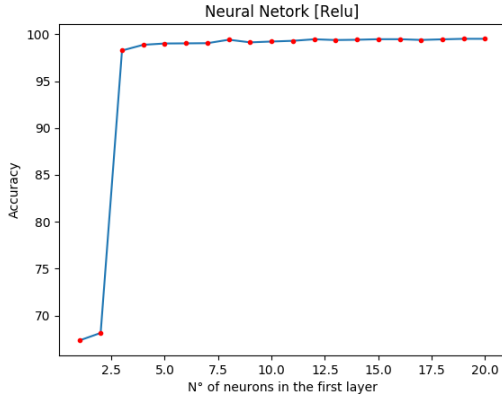


Figure 5.24: Accuracy (Acc^{reg} %) of Neural-Network with Relu perceptron algorithm as a function of the number of neurons in the first layer.

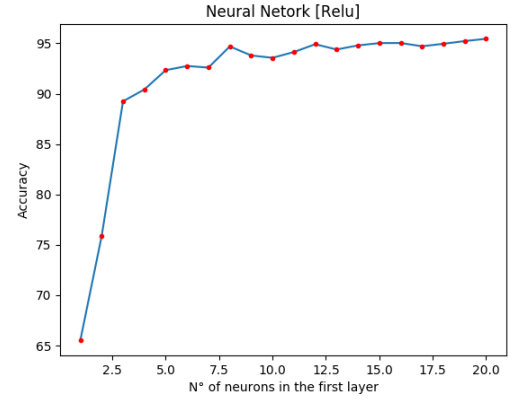


Figure 5.25: Accuracy ($Acc_{t_p}^{reg}$ %) of Neural-Network with Relu perceptron algorithm as a function of the number of neurons in the first layer.

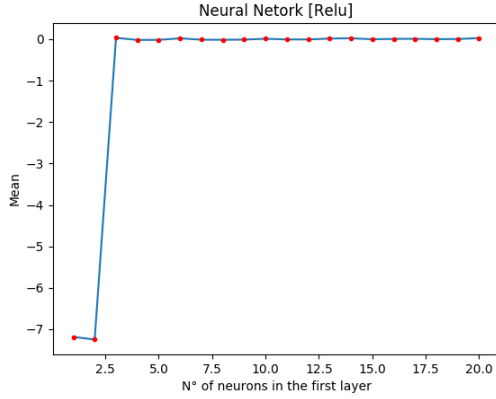


Figure 5.26: Mean of the prediction error μ_ϵ of Neural-Network with Relu perceptron algorithm as a function of the number of neurons in the first layer.

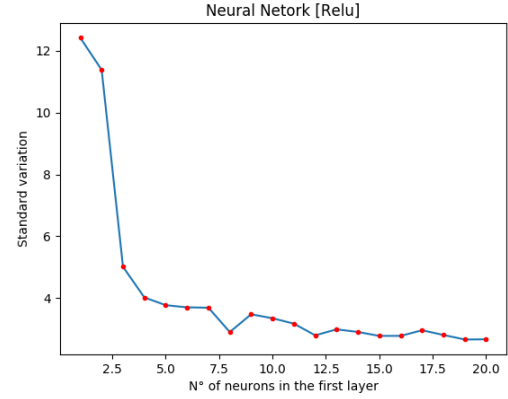


Figure 5.27: Standard deviation of the prediction error σ_ϵ of Neural-Network with Relu perceptron algorithm as a function of the number of neurons in the first layer.

negative rate evolution as the number of neurons in the first layer increase respectively. Similar to the logistic perceptron structure, these results show that as the number of neurons in the first layer increases, the accuracy (respectively the True Negative rate) increases. The accuracy reaches 99.5% and the true positive reaches 95.5%. The accuracy of the learning algorithm starts stabilizing at nearly three neurons, but the true positive rate continues to grow and stabilizes at nearly ten neurons. It can be explained by the fact that the standard deviation (figure 5.27) stabilizes also at nearly 10 neurons by reaching 2.6 km/h while the mean (figure 5.26) stabilizes at 3 neurons around 0 km/h . It seems that similar to logistic perceptron if the structure holds more than ten neurons in the first layer, the complexity of the algorithm is increased, but the accuracy does not get better.

Remark 18. *The effect of using multiple layers in the neural network structure is reported*

5. ON-BOARD INTRUSION DETECTION AND PREVENTION SYSTEM

in table 5.3.

Overall results for three drivers: Table 5.3 reports accuracy and true negative rate results of the different tested algorithm for three different drivers. The goal of testing the learning algorithms on different data sets captured from different drivers is to establish if they have similar performances or if the performances depend on the driving behavior.

Table 5.3: Prediction accuracy of detection rules for $tp = \pm 5 \text{ km/h}$ trained and tested with data captures from three different drive tests

ML-Algorithm	Tuning	Driver 1		Driver 2		Driver 3	
		Acc^{reg}	$Acc_{t_p}^{reg}$	Acc^{reg}	$Acc_{t_p}^{reg}$	Acc^{reg}	$Acc_{t_p}^{reg}$
KNN regression	$k = 1$	99.97	99.66	99.97	99.77	99.66	99.22
KNN regression	$k = 2$	99.97	99.78	99.97	99.82	99.71	99.40
KNN Reg	$k = 3$	99.97	99.76	99.97	99.78	99.71	99.40
Linear Reg	Null	85.51	36.38	83.47	22.61	74.42	59.89
Decision Tree	depth = 10	99.71	98.19	99.67	98.39	98.58	96.17
Decision Tree	depth = 20	99.97	99.89	99.97	99.93	99.67	99.29
Decision Tree	depth = 40	99.97	99.92	99.97	99.96	99.77	99.59
NN (Logistic)	1 Layer, 30 neurons	98.97	94.74	98.22	88.52	75.67	84.94
NN (Logistic)	1 Layer, 35 neurons	98.96	94.90	98.35	88.95	80.38	83.02
NN (Logistic)	1 Layer, 40 neurons	99.01	94.76	98.62	88.66	82.10	84.97
NN (Logistic)	1 Layer, 80 neurons	99.15	94.74	99.07	92.58	97.54	94.20
NN (Relu)	1 Layer, 10 neurons	99.31	92.82	99.11	87.91	97.26	92.52
NN (Relu)	1 Layer, 20 neurons	99.25	92.44	99.35	93.58	97.32	92.55
NN (Relu)	1 Layer, 40 neurons	99.36	93.75	99.29	92.42	97.61	93.65
NN (Relu)	5 Layer, 10 neurons	99.53	95.19	99.46	94.52	97.67	94.11
NN (Relu)	10 Layers, 10 neurons	99.55	95.36	99.55	95.37	98.37	95.90

First, we recall that the results of KNN are merely provided as a baseline. In fact, using KNN is advantageous as it gives a very precise local approximation for dense and uniform distribution of the training set. It is nevertheless not useful in the context of embedded systems as it needs all the training data in memory in order to make a prediction. Second, we note that each algorithm performs approximately similarly on the three drivers. Third, we confirm the previous observation that for a given algorithm, we note that as we increase the complexity (tuning parameters) of the learning algorithm, the accuracy improves for all the drivers. The rule becomes progressively able to capture more dependencies. As a result, it becomes necessary to take into consideration the added complexity compared to the gain in accuracy. For the decision tree algorithm, changing the tree depth from 20 to 40 does not improve the accuracy significantly. Similarly increasing the number of neurons in the Logistic-Neural-Network up to 80 neurons, and increasing the number of layers in the Relu-Neural-Network up to 10 layers does not have a significant effect on the accuracy for all three drivers. We conclude that as the complexity of the algorithm increases, its ability to capture more dependencies also increases, but reaches a certain limit beyond which it is no longer advantageous to increase the complexity. Overall, and for all three drivers, we can establish that the best results were reported for the decision tree algorithm tuned with a

depth parameter equals 40.

5.7.2 Predicting a categorical signal

5.7.2.1 Brake lights command signal

For classification problem, we choose to validate the approach on the *brake-lights-command* categorical signals. This signal is sent from the Electronic Stability Program (ESP) module to control brake lights (actuator). It is encoded on only 1 bit and sent over a periodic frame to the actuator. When the driver is pushing the brake pedal, the ESP module sends the *brake-lights-command* = 1 command. Otherwise it send *brake-lights-command* = 0. On reception, the actuator knows if yes or not it should activate the brake lights.

Statistically, the occurrence of the commands throughout a circuit is not balanced, the reason behind that the brakes are activated occasionally. However, in general, the probability depends on the circuit itself. For instance, on a highway, it is less likely to activate the brakes. In city driving this probability increases. In order for the accuracy metric to make sense, test data should be balanced, i.e., the number of test vectors should be roughly the same for each class. Otherwise, the results can be misinterpreted and do not reflect real performances of the used algorithms [107].

5.7.2.2 Capturing nominal behavior of the brake-lights-command signal

We use the previously introduced supervised learning algorithms in order to capture the nominal behavior of the *brake-lights-command* signal. We use five different input signals featured in an input vector as in (5.45).

$$(S_{Speed}, S_{Acceleration}, S_{RPM}, S_{Torque}, S_{Shifter}) \quad (5.45)$$

The goal is to estimate the label distribution of the *brake-lights-command* signal $< S_{Brake-lights} >$ given the input vector, i.e., the problem is to estimate the probability expressed in (5.46).

$$P(S_{Brake-lights} | (S_{speed}, S_{Acceleration}, S_{RPM}, S_{Torque}, S_{Shifter})) \quad (5.46)$$

In the performed experiments, the goal is to compare different classification machine learning algorithms. We used a data set of 10^6 input vectors from each drive test that we balance first. The resulting data set was split into a training set and a test set of 0.7 and 0.3 size ratio respectively. All experiments are done with the `scikit-learn` library [105].

KNN: We test the KNN algorithm with multiple values of the number of neighbors considered (k) increased progressively in order to establish a reference accuracy. The goal is to find the optimal number of neighbors for which the accuracy is maximized. Figure 5.28 reports the evolution of the accuracy of the prediction model that also corresponds to the True negative rate. These results show that as the number of considered neighbors increases, the accuracy decrease. Even though the difference between using one neighbor and ten neighbors is only 0.04%, the tendency is clear, and we establish that there is no need to increase the number of neighbors.

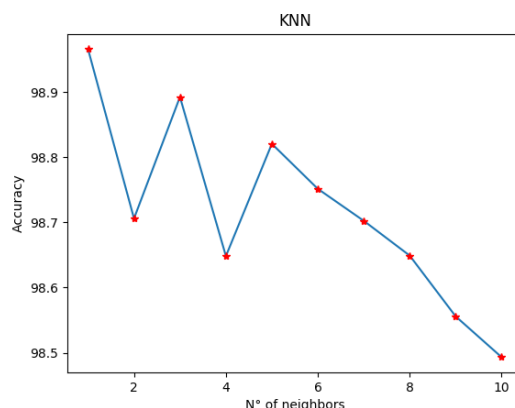


Figure 5.28: Accuracy Acc of the KNN classification algorithm as a function of the number of neighbors

Logistic regression: When testing the logistic regression algorithm, there are no hyper-parameters to optimize. Table 5.4 reports the resulting accuracy of the logistic regression algorithm tested on the first driver. Given an accuracy of only 93%, we can establish that this algorithm is not adapted to predicting the brake-lights-command signal from the given input.

Table 5.4: Logistic Regression detection rule

Algorithm	$Acc[\%]$
Logistic Regression	93.68

Decision Tree: We use the classification Decision Tree algorithm in order to predict the brake-lights-command signal. Similar to the previous test, we evaluate the evolution of the prediction accuracy as the tree depth increases. Figure 5.29 show the obtained results. We can see that as the allowed tree depth increases, the accuracy improves. The algorithm becomes progressively able to capture more dependencies between input data that makes it able to predict better the target signal. This accuracy reaches a limit of 99.3%. Starting from a tree depth of about 20, the accuracy does not seem to improve anymore. In this case, only the complexity of the tree increases but not the performance. Nevertheless, as it is showed later in Table 5.5 may depend on the driver's behavior.

Neural network with logistic perceptron: Similar to regression tests, we use the classification Neural-Networks with logistic perceptron to predict the brakes-lights-command signal. For the first test, we start with only one layer, and we increase the number of neurons in the first layer progressively and evaluate the evolution of the prediction accuracy on the first driver. Figure 5.30 shows that the accuracy improves as the number of neurons in the first layer increase. It seems to reach a limit between 96% and 96.5% accuracy for a number of neurons of 10 neurons. When increasing the number of neurons beyond 10 this accuracy does not improve any more. Note that the curve is wiggly. It is due to the fact that the final result

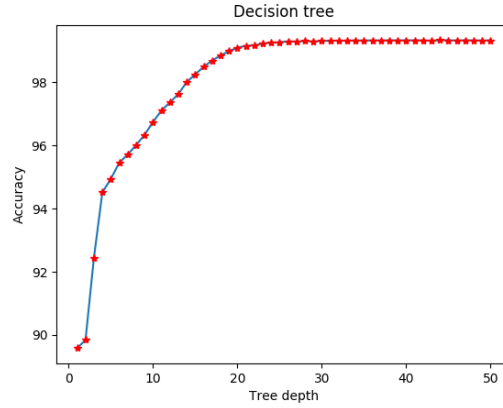


Figure 5.29: Accuracy Acc of the decision tree classification algorithm as a function of the tree depth

for each test depends on the start parameters (weights) of the neurons structure and the fact that the minimization solver can run into local minima. Note also that for only one neuron we get the same result obtained in logistic regression (i.e. 93%). In fact, a neural network with one layer one neuron is equivalent to the logistic regression algorithm.

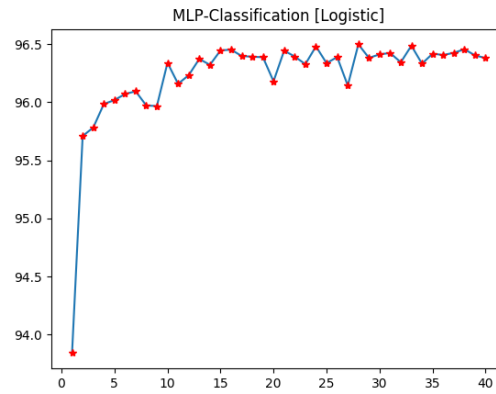


Figure 5.30: Accuracy Acc of the Neural network with logistic perceptron classification algorithm as a function of the number of neurons in the first layer

Neural Network with Relu perceptron: Performance results of the classification neural network with rectified linear unit (Relu) perceptron are reported in figure 5.31. The same observations also apply here. We note an improvement of the accuracy as the number of neurons increase. Starting from 10 neurons the accuracy reaches a limit between 96% and 96.5%.

Remark 19. *The effect of using multiple layers in the neural network structure is reported*

5. ON-BOARD INTRUSION DETECTION AND PREVENTION SYSTEM

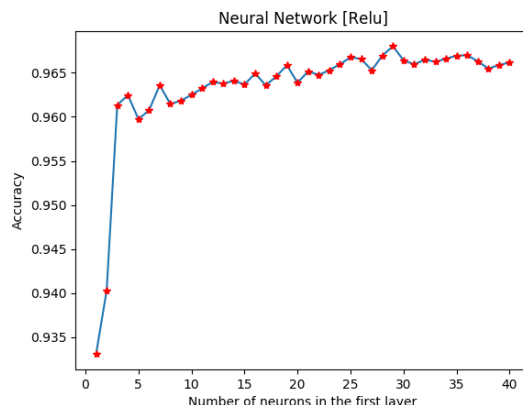


Figure 5.31: Accuracy Acc of the Neural network with Relu perceptron classification algorithm as a function of the number of neurons in the first layer

in table 5.5.

Overall results for three drivers: Results for the three drivers are summarized in Table 5.5. We notice that there are small differences in the accuracy for the same rule

Table 5.5: Prediction Accuracy of detection rules for the *brake-lights-command* signal

ML-Algorithm	Tuning	Driver 1 $Acc(\%)$	Driver 2 $Acc(\%)$	Driver 3 $Acc(\%)$
KNN classification	$k = 1$	98.96	98.45	97.27
KNN classification	$k = 2$	98.70	98.11	96.14
KNN classification	$k = 3$	98.89	98.34	97.22
Logistic Regression	Null	93.68	93.01	90.62
Decision Tree	depth = 10	96.72	95.80	94.65
Decision Tree	depth = 20	99.10	98.63	97.12
Decision Tree	depth = 40	99.36	99.00	97.77
Neural Net (Logistic)	1 Layer, 30 neurons	95.86	94.23	94.56
Neural Net (Logistic)	1 Layer, 35 neurons	95.82	94.11	94.48
Neural Net (Logistic)	1 Layer, 40 neurons	96.01	93.88	94.57
Neural Net (Logistic)	1 Layer, 80 neurons	95.97	94.15	94.55
Neural Net (Logistic)	5 Layer, 30 neurons	95.22	93.43	94.80
Neural Net (Relu)	1 Layer, 10 neurons	96.25	94.59	94.23
Neural Net (Relu)	1 Layer, 20 neurons	96.56	95.26	94.33
Neural Net (Relu)	1 Layer, 40 neurons	96.70	95.38	94.48
Neural Net (Relu)	5 Layer, 10 neurons	96.70	95.49	94.49
Neural Net (Relu)	10 Layers, 10 neurons	96.72	95.67	94.70

when comparing between different drivers. In fact, practically all the tested rules perform better on the first and second driver than on the third driver. An explanation of this result

might be that the third drive test contained singular use-cases that did not appear frequently enough. Thus the rules did not train well enough in order to recognize them. An easy solution to overcome this limitation is to collect more data for these specific use-cases. We also notice that the decision tree algorithm tuned with depth parameter equals to 40, reported the best performance for all three drivers.

5.7.3 Unification of detection rule

In the previous section, we reported results on the accuracy of the predictors trained and evaluated for each driver separately. The resulting detection rules could be influenced by the driving behavior of the driver. In this section, we investigate the possibility of building one single detection rule that can accommodate all three drivers. According to the previous results, the Decision Tree algorithm outperforms the rest of the algorithms for both predicted signals. Thus, we use the Decision Tree algorithm to build the detection rules in this section. In order to train the algorithm, we combine the data sets collected during the three drive tests, and we split the resulting data set into 0.7 and 0.3 ratio training set and test sets. We report results of the accuracy on the test set as well as on the three data sets separately for the speed signal in Table 5.6 and for *brake-lights-command* in Table 5.7.

Table 5.6: Prediction Accuracy of the unified detection rules for the *speed* (Decision Tree with Tree depth= 40)

ML-Algorithm	All		Driver 1		Driver 2		Driver 3	
	Acc^{reg}	$Acc_{t_p}^{reg}$	Acc^{reg}	$Acc_{t_p}^{reg}$	Acc^{reg}	$Acc_{t_p}^{reg}$	Acc^{reg}	$Acc_{t_p}^{reg}$
Decision Tree	99.95	99.66	99.97	99.77	99.98	99.77	99.76	99.43

Table 5.7: Prediction Accuracy of the unified detection rules for the *brake-lights-command* (Decision Tree with Tree depth= 40)

ML-Algorithm	All $Acc(\%)$	Driver 1 $Acc(\%)$	Driver 2 $Acc(\%)$	Driver 3 $Acc(\%)$
Decision Tree	98.16	99.37	98.16	97.97

Results show that, for both signals, the resulting detection rules have a high accuracy level on the combined data set as well as on data from each driver. This shows that it is possible to build a single detection rule that can accommodate the three drivers.

5.8 Evaluation against attacks

The robustness evaluation against attacks measures the detection capability of a given rule in case of an attack on the target signal. Thus we need to conduct an attack campaign and measure the **True Negative rate** defined as the ratio between the number of *alerts raised* and the number of attacked frames, and the **False Negative rate** defined as the ratio between the number of *alerts not raised* and the number of attacked frames, against each type of attack (Table 5.1). For safety reasons, this attack campaign should not be done on a real vehicle.

5.8.1 Simulation of attacks

In order to evaluate the effectiveness of the detection rule, we conduct a test campaign against simulated attacks. Since we claim that our model can detect attacker that has full control over one of the ECUs (Figure 2.9b), we assume an attacker aiming to cause a tangible impact on the vehicle and to hide real sensor values or commands. The simulated attacks consist in replacing the data content of the messages with an attacked content. Thus the attacker is showcasing a *Man-in-the-middle attack* between the signal generator (sensor) and the receiver ECU on which we install the intrusion detection system.

5.8.2 Attacks against real-valued signal

For the speed signal monitoring we perform three types of attacks:

- Random speed injection: in this attack, the attacker substitutes the real sensor value with a random value.
- Speed offset injection: in this attack, the attacker adds to the real speed sensor value an offset value.
- Speed Denial of service (signal drop): in this attack, the attacker interrupts the sending of the frame causing the speed signal to freeze at the last sent value.

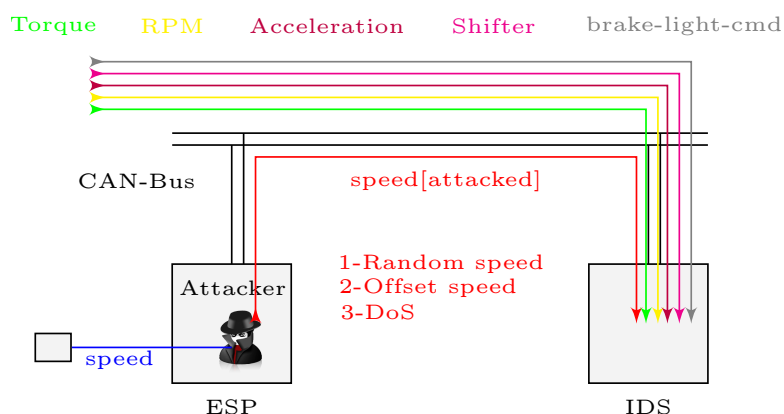


Figure 5.32: Illustration of *Man-in-the-middle attack* principle on the Speed signal

Remark 20. Note that the intrusion detection system is set to raise an Alert as long as the received speed value (injected by the attacker) is outside the acceptance interval of ± 5 km/h of the predicted speed value. Thus we consider that an attack is happening if the injected speed signal is outside of this acceptance interval.

5.8.2.1 Random speed injection attack

Figure 5.33 shows the random speed injection attack use-case and Figure 5.34 shows alerts raised by the decision tree detection rule against this attacks. The results show that as long

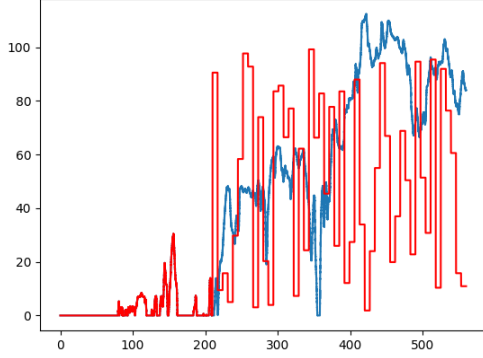


Figure 5.33: Evolution of the original speed signal (blue) and random speed signal (red) over time.

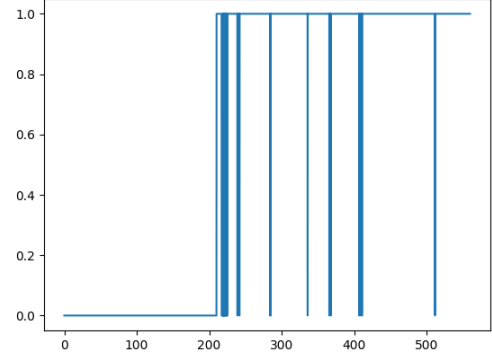


Figure 5.34: Alerts raised by the decision tree (depth=40) detection rule against random speed injection attack

as the injected speed value is outside the acceptance window of $\pm 5 \text{ km/h}$, alerts are raised. The alert is not raised when the injected speed value is close to the ground truth value (values around $t = 220s, 280s, 400s, 510s$). We obtained **0.13%** of false negatives when performing this attack.

5.8.2.2 Speed offset injection attack

Figure 5.35 shows the speed offset injection attack use-case and Figure 5.36 shows alerts raised by the decision tree detection rule against this attacks. For this attack, we can see that the

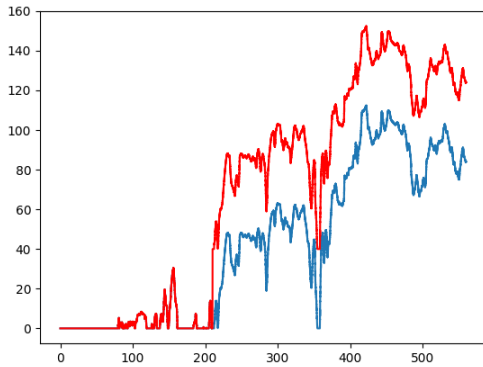


Figure 5.35: Evolution of the original speed signal (blue) and offset speed signal (red) over time.

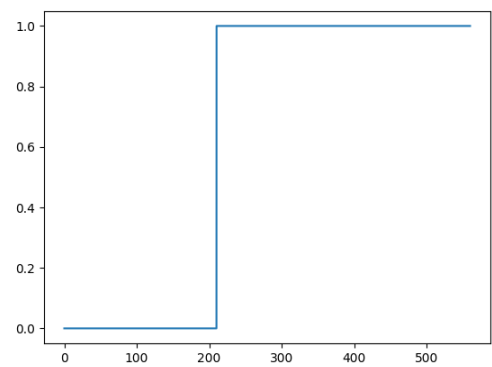


Figure 5.36: Alerts raised by the decision tree (depth=40) detection rule against speed offset injection attack

alert is raised as soon as the attack started (around $t = 210s$) and was sustained as long as the attack was running. In fact, since the speed offset of the attack is set to $+40 \text{ km/h}$, the received signal is always outside the acceptance window of $\pm 5 \text{ km/h}$. The detection in this

5. ON-BOARD INTRUSION DETECTION AND PREVENTION SYSTEM

case is perfect and we obtained $5.810^{-5}\%$ of false negatives.

5.8.2.3 Speed Denial of service (signal drop) attack

Figure 5.37 shows the speed DoS attack use-case and Figure 5.38 shows alerts raised by the decision tree detection rule against this attacks. Similar to the previous attacks, the same

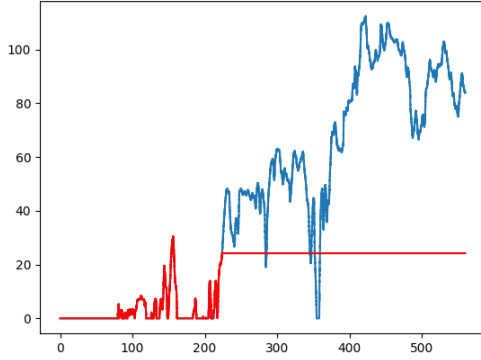


Figure 5.37: Evolution of the original speed signal (blue) and frozen speed signal (red) over time.

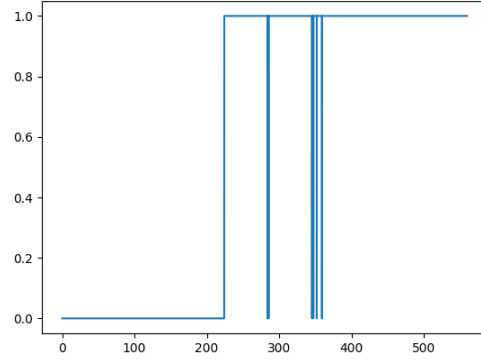


Figure 5.38: Alerts raised by the decision tree (depth=40) detection rule against random speed injection attack

reasoning applies. The injected speed is frozen at around 20 km/h , which means that most of the time the alarm is raised as the received speed is outside the acceptance window. However, as soon as the ground truth speed value approaches the injected value (around $t = 280s, 350s$), the alarm turns off. We obtained 0.19% of false negatives on this attack.

Table 5.8: False Negative rate of simulated attacks on the *Speed* signal

Attack type	False Negative(%)
Random speed injection	0.13%
Offset speed injection	$5.810^{-5}\%$
Denial of service (signal drop)	0.19%

5.8.3 Attacks against categorical signal

Similar to the speed signal and in order to test the intrusion detection rule set-up in the previous section for the *brake-lights-command* signal monitoring we perform three types of test:

- Random command injection: in this the attack, the attacker, substitutes the real command with a (0/1) random command.
- Inverse command injection: in this attack, the attacker inverts to the real command.

Table 5.9: False Negative rate of simulated attacks on the *brake-lights-command* signal

Attack type	False Negative(%)
Random command injection	0.98%
Inverse command injection	1.67%
Denial of service (force to 0)	0.4%

- Denial of service (force to 0): in this attack, the attacker always sends the 0 command value.

For convenience we grouped all the attacks in Figure 5.39.

Remark 21. Note that the detection rule is set to raise an Alert as long as the received command value (injected by the attacker) differs from the predicted command. Thus we consider that an attack is happening if the injected command signal is different from the real brake-lights-command signal.

5.8.3.1 Random command injection attack

The **first** column of Figure 5.39 shows the random command injection attack use-cases on the *brake-lights-command* signal. We can see from the Alerts raised by the detection rule for this attack that as long as the injected command differs from the ground truth command, alerts are raised. The alert is not raised when the injected and ground truth commands are the same. We obtained a false negative rate of **0.98%**.

5.8.3.2 Inverse command injection attack

The **second** column of Figure 5.39 shows the inverse command injection attack use-cases on the *brake-lights-command* signal. For this attack, we can see that the alert is raised as soon as the attack started. In fact, since the injected command is always the opposite of the ground truth command, the predicted signal is always different from the received signal. Thus an attack is detected from the start, and we obtained a false negative rate of **1.67%**.

5.8.3.3 Denial of service (force to 0) attack

The **third** column of Figure 5.39 shows the Denial of service attack use-cases on the *brake-lights-command* signal. In this attack, the injected command is set to 0. The ground truth *brake-lights-command* have occurrences of about 70% and 30% for 0 and 1 respectively. Thus, we consider that there is an attack only 30% of the time. Similarly, the alerts were raised when the injected command differs from the ground truth command. We obtained a false negative rate of **0.4%**.

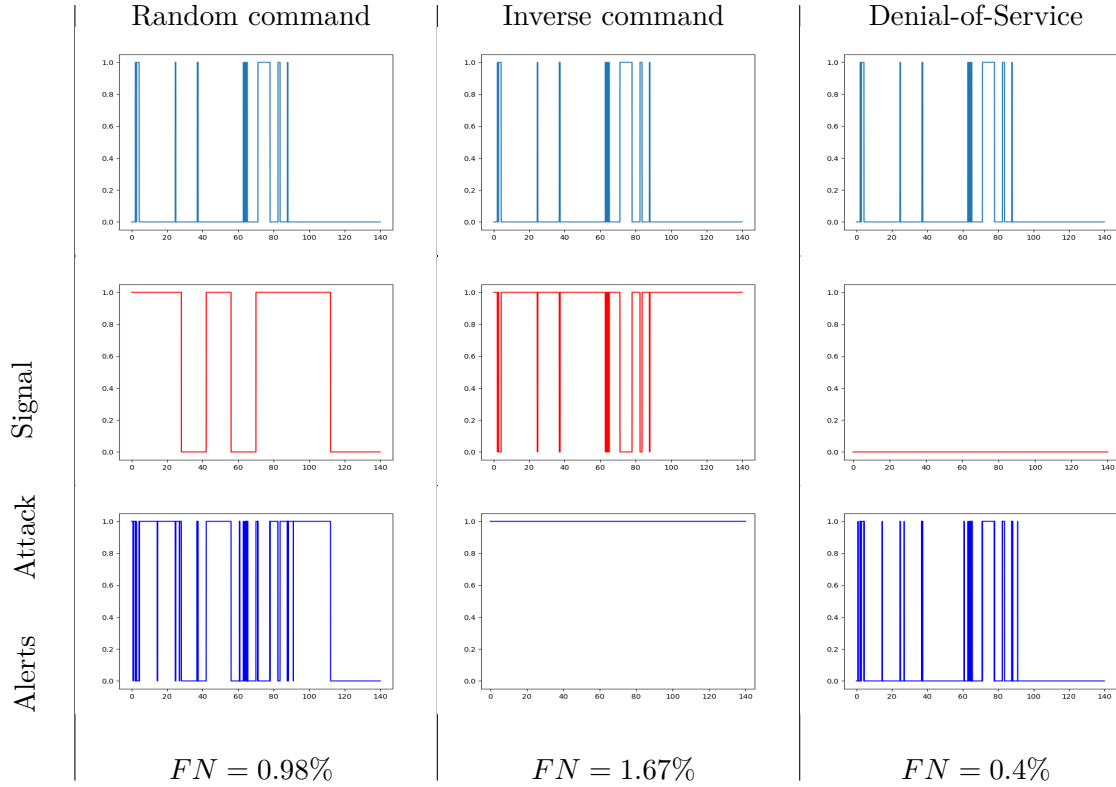


Figure 5.39: Alerts raised by the decision tree (depth=40) detection rule tested on three different attacks on the *brake-lights-command* signal. On top is the ground truth command, in the middle is the attack command and on the bottom is the Alerts raised by the detection rule when receiving the attack signal.

5.9 Alerts handling

5.9.1 Prevention mechanism

As explained in previous sections, the proposed intrusion detection system makes the signal prediction based solely on other signals and does not take the monitored signal as input in the prediction process. We have also presented in section 5.8 an evaluation against attacks and demonstrated how the intrusion detection rule could catch the attacks with a low false negative rate for an attacker that has full control over a legitimate ECU. This capability comes from the fact that the predictor can make very accurate signal predictions even when the ground truth signal is not received (i.e. based only on the other signals). In figure 5.40 we depicted the ground truth as well as the prediction of the speed signal. It shows that the prediction is almost always superposed to the real signal. As a result, the formulation that we proposed allows for a prevention strategy. Even when the signal has not been received the system can still make correct predictions given that the other signals are not maliciously manipulated. Thus, the predicted signal can serve as a replacement when an attack is detected, which can be considered as *fail-safe* mode for the vehicle.

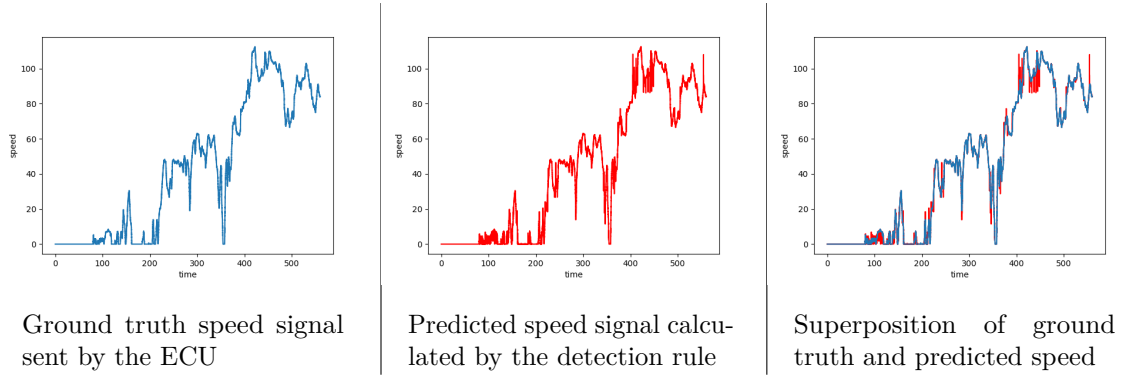


Figure 5.40: Comparison between the ground truth signal and the predicted signal (signal predicted with decision tree (depth=40) regression algorithm)

5.9.2 False positives reduction strategy

One may consider the false positive rate of 3% or even 1% not low enough given the high number of frames used within the communication buses. For this purpose in this section we propose to account for *successive alerts* as a remedy. In fact, in order to effectively influence the behavior of the car, the attacker needs to send successive attack frames. Thus, the idea is to consider that an isolated detection alert could be ignored, and focus on successive alerts. In case there are no attacks taking place, these alerts are in fact false positives. Thus it is crucial also to characterize the intrusion detection rules with regards to the successive false positives. The strategy is to set-up an acceptable number of successive false positives k and to effectively raise an alert only after k false predictions. In figure 5.41 we draw the probability (in percentage) of raising a false alarm as a function of the successive false predictions k for the decision tree detection rule applied to the speed signal.

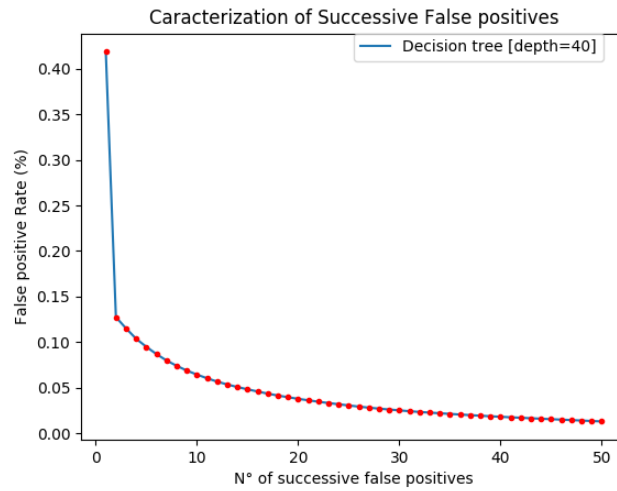


Figure 5.41: Characterization of the ratio of false alerts (in percentage%) raised as a function of the number of successive false positives.

This technique can tremendously reduce the number of false positives. We can see from figure 5.41 that simply by looking for two successive false predictions, we can already reduce the number of false alarms from 0.5% to less than 0.15%. We also set a requirement for the acceptable false alarms that can be tolerated by the IDS then find the number of successive false predictions that can be accepted in order to comply with the requirement.

Remark 22. *Note that the bigger the number of successive false positive allowed, the more leverage the attacker has. For instance, imagine we allow the IDS to raise alerts only after 10 successive false positives. The attacker may use this in order to send 9 attack frames and then pauses, waits for the counter to reset and re-start with another 9 attack frames.*

5.10 Conclusion and discussion

In this chapter, we introduced a novel in-vehicle intrusion detection system capable of detecting an attacker with full control over an ECU. This intrusion detection system is based on detection rules built with supervised machine learning techniques. The rules learn the nominal behavior of the system and make predictions for individual signal value. Alarms are raised when the predicted signal value is not similar to the received value. We showed first the effectiveness of the detection rules for separate drivers, then for a set of drivers. We also showed the effectiveness of the detection rules against examples of attacks. The advantage of the proposed method compared to the previous work is that it only needs collected data to learn nominal behavior, and does not need examples of attacks in order to recognize them. Plus, it gives the ability to target individual signals (for instance most safety critical). Since the detection rules are actually signal predictors, theoretically the approach could be used for prevention as well.

In order for this intrusion detection system to be effective, the training data has to be representative of all legitimate states of the vehicle. If some of the legitimate scenarios are not included in the training set and the test set, the detection rules may not learn the normal behavior corresponding to it, thus resulting in a lousy performance during online monitoring. Similarly, it is important to account for all the driving behaviors. This is clearly more complex to take into consideration as it involves collecting training data from a large set of drivers exhibiting different but legitimate behaviors. In the context of autonomous driving, this task can be more manageable as the driver behavior does not influence the signals that are mostly controlled by electronic control units. The data acquisition step can be much easier as the self-driving car can be simulated (for instance in Hardware In the Loop equipment).

Another important angle to address the effectiveness of the IDS is the complexity and resources. The final objective of the proposed intrusion detection system should be implemented inside the vehicle. Therefore, it is essential to consider the *prediction complexity* as well as *resources* needed of the algorithms. In fact, depending on the target embedded system environment on which the detection rule has to be implemented we may have resource constraints to cope with as well as safety constraint that can indicate how much delay the signal validation process can take.

6.1 Summary

Throughout this theses, we begin by investigating the state of the art of automotive security. Multiple attacks that have been performed during the last years against different automotive systems were reported and categorized with regards to their access vector. We emphasized the fact that security of automotive systems has to begin from the design phase as many decisions that are taken then have a substantial impact on the level of security. In this context, we reported some of the threat analysis and risk assessment methodologies that play an essential role in defining the security objectives and setting up a security strategy from which security requirements are established and pushed to the product development phase. We identified a gap with regards to the identification of different attack scenarios for a particular threat.

In chapter 3 we addressed this gap and proposed a formal model that allows, given a cyber-physical architecture of a vehicle and an attacker model along with its objectives (the defined threats), to generate different attack scenarios (attack paths) in the form of an attack tree. This attack tree serves later in the risk analysis step in order to prioritize the scenarios and propose adequate architectural solutions in order to build protection against the attacks.

We noticed however that in-vehicle communication networks constitute the last line of defense against the automotive cyber-physical attacks. In chapter 4 we proposed different solutions to protect against an attacker with direct physical access to the CAN network. These protection solutions are based on randomization strategies of the CAN identifier and turn out to be very effective against injection and replay attacks as well as reverse-engineering.

Considering a more complex attacker model that has indirect physical access and even remote access to one of the legitimate ECUs inside the vehicle, in chapter 5 we develop a protection mechanism for in-vehicle communication networks. The protection mechanism is an intrusion detection system that uses machine learning algorithms to extract a dependency

6. CONCLUSION

model between data collected from the shared communication buses. The proposed approach relies on the training and evaluation of multiple learning algorithms in order to select the one that performs the best.

6.2 Perspectives and future research directions

Some additional questions with regards to the general issues of automotive security are still to be answered and others can be raised through the findings of this thesis.

6.2.1 On risk assessment

First, we note that the attributes associated with the assets in the formal model were used in order to offer additional possibilities in the formulation of the security property (attack goal) to be verified and the resulting state space generation. Those attributes (and others that could be added) can be used to partly automate the attack rating process in the risk assessment step based on the attack tree. An obvious example of this approach could be to use the “Accessibility” attribute of the communication mediums in order to indicate the “window of opportunity” score. Additional attributes relative to the difficulty of exploitation of certain vulnerabilities can help indicate the “Expertise” score and “Time” needed. Furthermore, adding attributes to indicate for each attack the “Equipment” needed can help better assess for an attack scenario the set of equipment needed. The latter can solve the problem wherein a single attack (i.e. a conjunction combination) two attack steps may be enabled by the same equipment or require the same expertise.

Other improvements could be to enhance the vulnerability model. In the actual formal model, exploiting a vulnerability in a service from the attacker point of view would allow him to take control over that service and have the same access rights. In the real world, this is true, but we have other types of vulnerabilities that would allow the attacker to perform “privilege escalation” type of exploits. The latter is not included in the formal model yet and will help to discover additional attack paths and possibilities.

6.2.2 On in-vehicle secure communications

Following the promising results obtained in chapter 5 further improvements also can be considered. First, we note that during the evaluation against attacks, only the target signal (signal to be monitored) was attacked. As a perspective, we can investigate the effectiveness of the intrusion detection system against attacks on multiple signals simultaneously. Furthermore, it is interesting to combine multiple monitoring algorithms used for multiple target signals with each other in order to improve their respective performances. We can also investigate the effectiveness of an intrusion detection system that uses different machine learning algorithms on the same target signal during monitoring.

Additionally, the intrusion detection system as presented in chapter 5 is designed to output a binary output indicating if there is an intrusion or not. However, the prediction algorithm is built with machine learning algorithms training on collected use-cases. An important issue is when we encounter a use-case for which the algorithm is not trained well. It becomes necessary

to output some further information on the confidence of the prediction. In other words, for each prediction, the prediction algorithm should output a prediction and say whether it is confident enough in the output.

Interestingly, as reported in chapter 5, one of the concerns when training the intrusion detection system is the driving behavior. This concern disappears entirely in the context of self-driving cars. At the same time, the need for intrusion detection systems in order to protect against cyber-physical attacks become more critical. However, it is still necessary to evaluate the intrusion detection system based on data collected from a self-driving car.

6. CONCLUSION

List of Publications

Book chapters:

- Khaled Karray, Jean-Luc Danger, Sylvain Guilley, and M Abdelaziz Elaabid. Attack Tree Construction and Its Application to the Connected Vehicle. *Cyber-Physical Systems Security*, 2018, p. 175.
- Khaled Karray, Jean-Luc Danger, Sylvain Guilley, and M Abdelaziz Elaabid. Identifier Randomization: An Efficient Protection Against CAN-Bus Attacks. *Cyber-Physical Systems Security*, 2018, p. 219.

Conference papers:

- Khaled Karray, Jean-Luc Danger, Sylvain Guilley, and M Abdelaziz Elaabid. Prediction-based Intrusion Detection System for In-vehicle Networks using Supervised Learning and Outlier-detection. *International workshop on information security theory and practice*, 2018.

Patent Applications:

- Khaled Karray, M Abdelaziz Elaabid. “PROCEDE, SUR UN RESEAU DE COMMUNICATION EMBARQUE D’UN VEHICULE DE TRANSMISSION SECURISEE D’UN MESSAGE”
- Khaled Karray, M Abdelaziz Elaabid, Sylvain Guilley, Jean-Luc Danger. “PRECEDE DE SUPERVISION DE SIGNAUX SENSIBLES SUR RESEAU DE COMMUNICATION EMBARQUE D’UN VEHICULE PAR APPRENTISSAGE” SUPERVISE

Conference presentation and Posters:

- Khaled Karray, Sylvain Guilley, Jean-Luc Danger, and M Abdelaziz Elaabid. A model for attack path generation in automotive domain (Poster). *Workshop on Practical Hardware Innovations in Security Implementation and Characterization, PHISIC*, 3-4 october, 2016.

6. CONCLUSION

- Khaled Karray, Sylvain Guilley, Jean-Luc Danger, and M Abdelaziz Elaabid. Protections against the automotive CAN bus attacks (Poster). *IMT Cybersecurity Seminar*, 2017.
- Khaled karray, Jean-luc Danger, Sylvain Guilley, and M Abdelaziz Elaabid. Attack tree generation: an application to the connected vehicle. *International Workshops on Cryptographic architectures embedded in logic devices, CryptArch*, 2017. <https://labh-curien.univ-st-etienne.fr/cryptarchi/workshop17/abstracts/karray.pdf>
- Khaled Karray, Jean-Luc Danger, Sylvain Guilley, and M Abdelaziz Elaabid. Attack Tree Construction and Its Application to the Connected Vehicle. *CPSEd*, 17-19 July 2017. <http://koclab.cs.ucsb.edu/cpsed/files/Karray.pdf>

In this appendix we report the modeling of the transformation rules as implemented in the GROOVE tool for the use-case introduced in chapter 3. The transformation rules are presented in the form of graphs. However, unlike in chapter 3, these transformation rules are not presented with their LHS¹, RHS² and NAC³, but rather as a single color-coded graph, where colors are interpreted (according to the GROOVE manual) as follows:

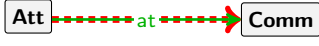
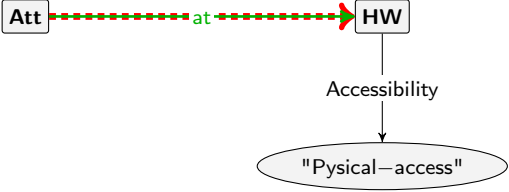
- **Black** elements are nodes and edges that should be present in the host graph for the rule to be applied.
- **Green** elements are nodes and edges created in the resulting graph if the rule matches.
- **Blue** elements are nodes and edges removed in the resulting graph if the rule matches.
- **Red** elements should not exist in the host graph for the rule to be applied.

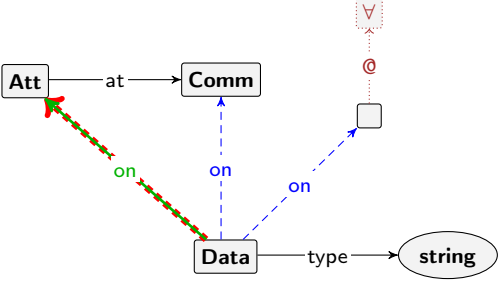
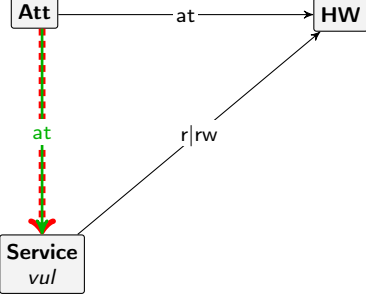
¹Left Hand Side

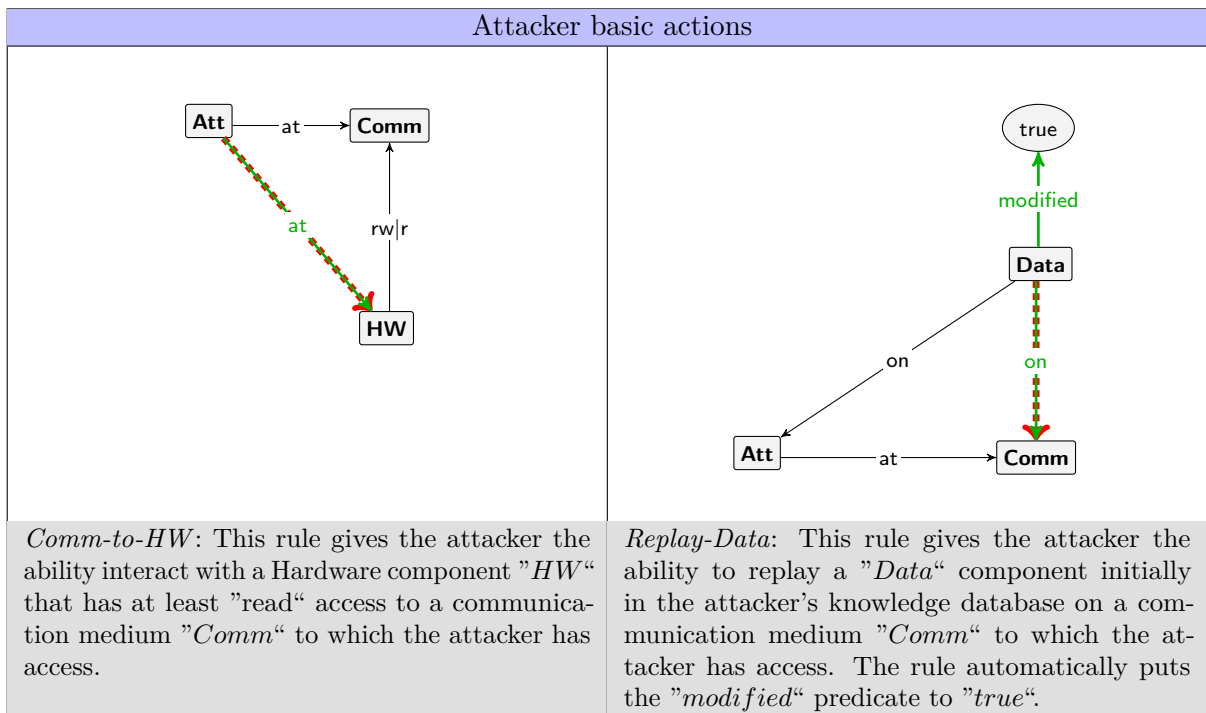
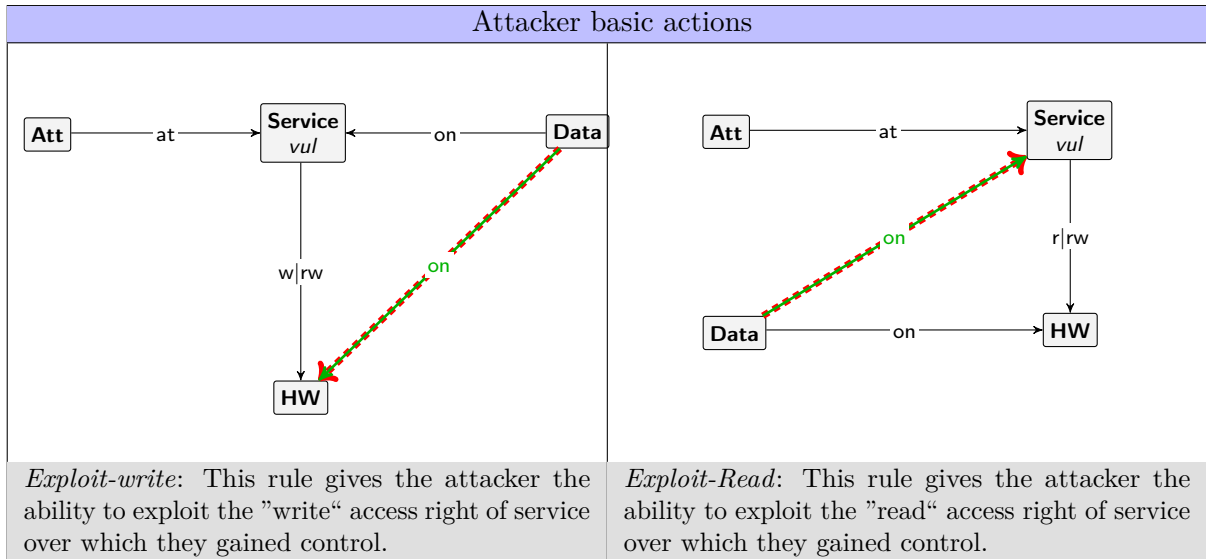
²Right Hand Side

³Negative Application Condition

6. CONCLUSION

Attacker basic actions	
	
<p><i>Connect-Comm:</i> This rule gives the attacker the ability to connect to any “<i>Comm</i>” component (i.e. a communication medium). Optionally, we can add the type of the communication component or the accessibility if we want to restrict the attacker’s abilities.</p>	<p><i>Connect-HW:</i> This rule gives the attacker the ability to connect to Hardware components “<i>HW</i>” that are physically accessible.</p>

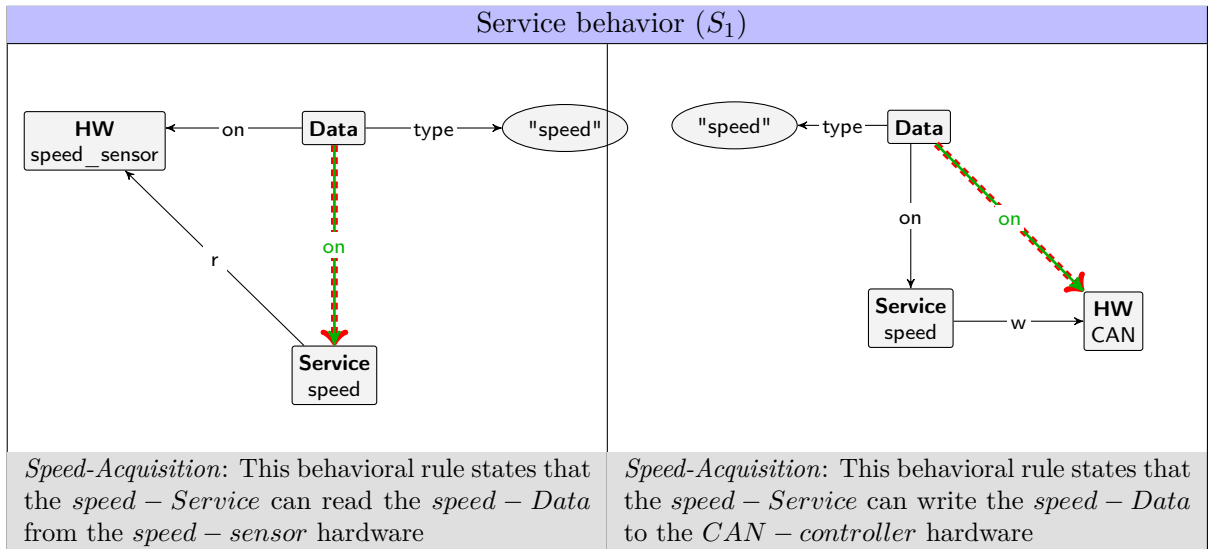
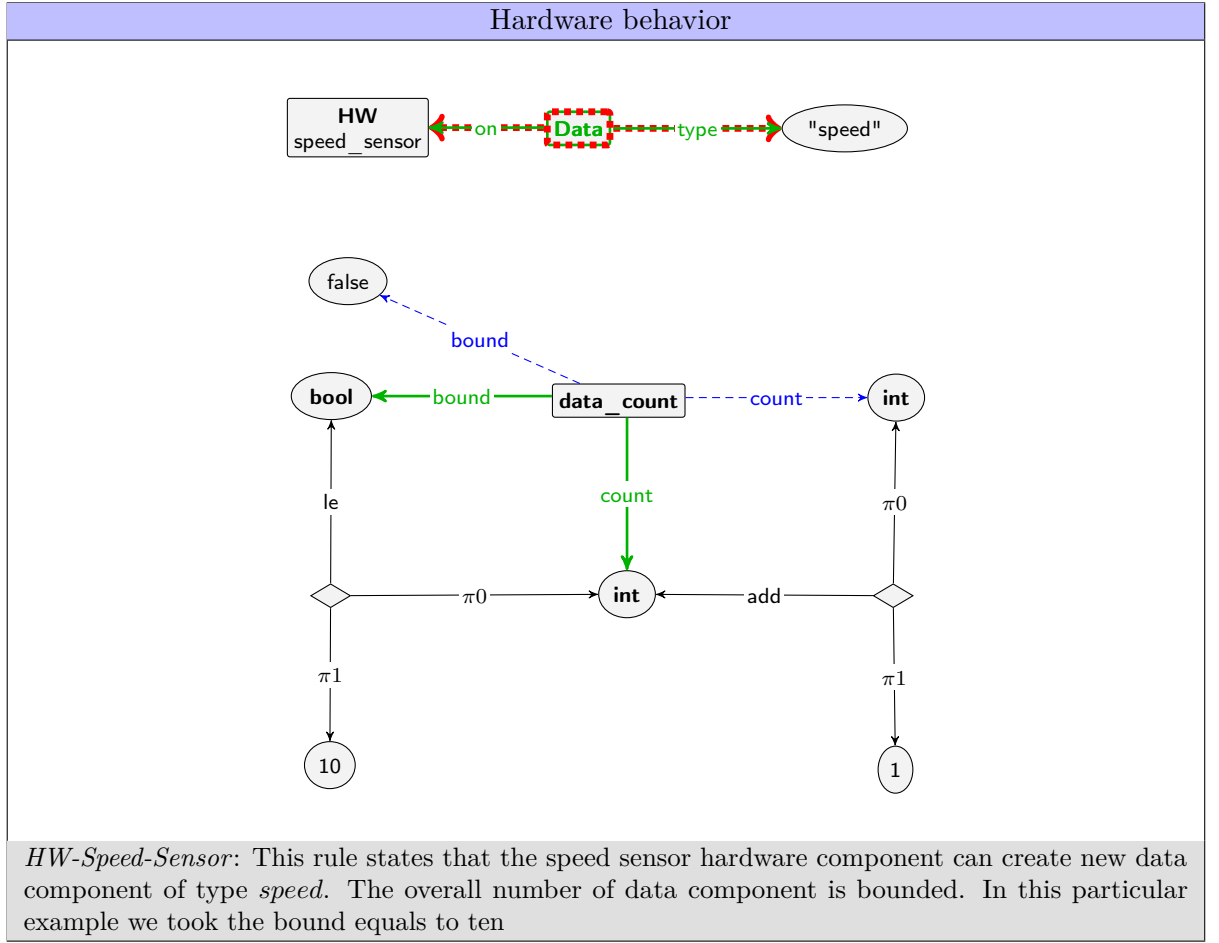
Attacker basic actions	
	
<p><i>Eavesdrop-Comm:</i> This rule gives the attacker the ability that gained access to a communication medium “<i>Comm</i>” to read the data that is being communicated on that communication medium (i.e transfer the data component to the attacker’s knowledge database).</p>	<p><i>Exploit-Service:</i> For an attacker that has access to a Hardware component “<i>HW</i>”, if there is a service with a vulnerability that has at least read access to the hardware component, the attacker can exploit the vulnerability which grants the attacker control over the service.</p>



6. CONCLUSION

Hardware behavior	
<p><i>HW-CAN-Read-From-Comm:</i> This rule states that any CAN-controller hardware connected to the CAN communication bus can read data from the bus.</p>	<p><i>HW-CAN-Write-To-comm:</i> This rule states that any CAN-controller hardware connected to the CAN communication bus can write data to the bus.</p>

Hardware behavior	
<p><i>HW-Cellular-Read-From-comm:</i> This rule states that any Cellular-controller hardware connected to the Cellular network can read data from the network.</p>	<p><i>HW-Cellular-Write-To-comm:</i> This rule states that any Cellular-controller hardware connected to the Cellular network can write data to the network.</p>



In this appendix we report the proofs of entropy and conditional entropy computations presented in chapter 4.

Let id_o be a random variable representing original identifiers whose outcome is id_1, id_2, \dots, id_N with probabilities $P(id_1), P(id_2), \dots, P(id_N)$. We consider a second random variable id_r representing randomized identifiers whose outcome is in $[0, 2^n - 1]$.

Entropy of Fixed mapping

The entropy of the fixed mapping solutions (IA-CAN, Equal-intervals, Frequency-intervals) is the following:

- IA-CAN: $H_{IA-CAN}(id_r) = H(id_o) + a$
- Equal Intervals: $H_{EI}(id_r) = H(id_o) + n - \log_2(N)$
- Frequency Intervals: $H_{FI}(id_r) = n$

Proof. According to the fixed mapping randomization functions (IA-CAN, Equal-intervals, Frequency-intervals) each identifier id_i is randomized over a fixed interval I_i of width $W(I_i)$. We begin by computing the probability that the random variable id_r takes the value $x \in [0, 2^n]$

$$P(id_r = x) = \sum_{i=1}^N P(id_r = x | id_i) \times P(id_i)$$

The conditional probability of id_r knowing the original identifier $id_o = id_i$.

$$P(id_r = x | id_i) = \frac{1_{I_i}(x)}{W(I_i)}$$

6. CONCLUSION

$$\begin{aligned}
H(id_r) &= \sum_{x \in [0, 2^n - 1]} P(id_r = x) \times \log_2\left(\frac{1}{P(id_r = x)}\right) \\
&= \sum_{x \in [0, 2^n - 1]} \left[\sum_{i=1}^N P(id_i) \frac{1_{I_i}(x)}{W(I_i)} \right] \times \log_2\left(\frac{1}{\left[\sum_{j=1}^N P(id_j) \frac{1_{I_j}(x)}{W(I_j)} \right]}\right) \\
&= \sum_{i=1}^N \sum_{x \in [0, 2^n - 1]} P(id_i) \frac{1_{I_i}(x)}{W(I_i)} \times \log_2\left(\frac{1}{\left[\sum_{j=1}^N P(id_j) \frac{1_{I_j}(x)}{W(I_j)} \right]}\right)
\end{aligned}$$

$$H(id_r) = \sum_{i=1}^N \sum_{x \in I_i} P(id_i) \frac{1_{I_i}(x)}{W(I_i)} \times \log_2\left(\frac{1}{\left[\sum_{j=1}^N P(id_j) \frac{1_{I_j}(x)}{W(I_j)} \right]}\right)$$

Since the intervals I_i are non overlapping : $\forall x \in I_i, \forall j \neq i \rightarrow 1_{I_j}(x) = 0$

We can thus simplify the expression: $\forall x \in I_i, \forall j \neq i \rightarrow \sum_{j=1}^N P(id_j) \frac{1_{I_j}(x)}{W(I_j)} = P(id_i) \frac{1_{I_i}(x)}{W(I_i)}$

$$\begin{aligned}
H(id_r) &= \sum_{i=1}^N \sum_{x \in I_i} P(id_i) \frac{1_{I_i}(x)}{W(I_i)} \times \log_2\left(\frac{1}{P(id_i) \frac{1_{I_i}(x)}{W(I_i)}}\right) \\
&= \sum_{i=1}^N \sum_{x \in I_i} P(id_i) \frac{1}{W(I_i)} \times \log_2\left(\frac{1}{P(id_i) \frac{1}{W(I_i)}}\right)
\end{aligned}$$

– IA-CAN entropy: $\forall i \in [1, N], W(I_i) = 2^a$

$$H(id_r) = \sum_{i=1}^N \sum_{x \in I_i} P(id_i) \frac{1}{2^a} \times \log_2\left(\frac{1}{P(id_i) \frac{1}{2^a}}\right) = H(id_o) + a$$

– Equal intervals entropy: $\forall i \in [1, N], W(I_i) = \frac{2^n}{N}$

$$H(id_r) = \sum_{i=1}^N \sum_{x \in I_i} P(id_i) \frac{1}{\frac{2^n}{N}} \times \log_2\left(\frac{1}{P(id_i) \frac{1}{\frac{2^n}{N}}}\right) = H(id_o) + n - \log_2(N)$$

– Frequency intervals entropy: $\forall i \in [1, N], W(I_i) = 2^n \times P(id_i)$

$$H(id_r) = \sum_{i=1}^N \sum_{x \in I_i} P(id_i) \frac{1}{2^n \times P(id_i)} \times \log_2\left(\frac{1}{P(id_i) \frac{1}{2^n \times P(id_i)}}\right) = n$$

□

Conditional entropy of Fixed mapping

The conditional entropy of randomized identifiers knowing the original identifiers of the fixed mapping solutions (IA-CAN, Equal-intervals, Frequency-intervals) is the following:

- IA-CAN: $H_{IA-CAN}(id_r|id_o) = a$
- Equal Intervals: $H_{EI}(id_r|id_o) = n - \log_2(N)$
- Frequency Intervals: $H_{FI}(id_r|id_o) = n - H(id_o)$

Proof.

$$H(id_r|id_o) = H(id_r, id_o) - H(id_o)$$

$$H(id_r, id_o) = \sum_{x \in [0, 2^n - 1]} \sum_{i=0}^N P(id_r = x, id_o = id_i) \log_2 \left(\frac{1}{P(id_r = x, id_o = id_i)} \right)$$

$$P(id_r = x, id_o = id_i) = \begin{cases} P(id_i) \times \frac{1}{w(I_i)} & , \quad x \in I_i \\ 0 & , \quad elsewhere \end{cases}$$

$$H(id_r, id_o) = \sum_{i=0}^N \sum_{x \in I_i} \frac{P(id_i)}{w(I_i)} \log_2 \left(\frac{1}{P(id_i) \frac{1}{w(I_i)}} \right)$$

$$H(id_r, id_o) = H(id_r)$$

$$H(id_r|id_o) = H(id_r) - H(id_o)$$

- IA-CAN conditional entropy : $H(id_r|id_o) = a$
- Equal intervals conditional entropy : $H(id_r|id_o) = n - \log_2(N)$
- Frequency intervals conditional entropy : $H(id_r|id_o) = n - H(id_o)$

□

Entropy of Dynamic intervals

Let id_o^t be a Markov chain over the space of original identifiers $(id_1, id_2, \dots, id_N)$. And the matrix M presented in equation (4.26) be its transition matrix. Let id_r be the random variable over $[0, 2^n - 1]$, generated using the dynamic interval randomization strategy applied to id_o^t . we have: $H_{DI}(id_r) = n$

6. CONCLUSION

Proof.

$$\begin{aligned}
H(id_r) &= \sum_{x \in [0, 2^n - 1]} P(id_r = x) \times \log_2\left(\frac{1}{P(id_r = x)}\right) \\
P(id_r = x) &= \sum_i^N P(id_r = x | id_o^t = id_i) \times P(id_o^t = id_i) \\
P(id_r = x) &= \sum_i^N \sum_j^N P(id_r = x | id_i^t, id_j^{t+1}) \times P(id_j^{t+1} | id_i^t) \times P(id_i^t) \\
P(id_r = x | id_i^t, id_j^{t+1}) &= \frac{1_{I_{i,j}}(x)}{W(I_{i,j})}
\end{aligned}$$

Where $W(I_{i,j}) = P(id_j^{t+1} | id_i^t) \times 2^n$ is the width of the interval $I_{i,j}$

$$\begin{aligned}
P(id_r = x) &= \sum_i^N \sum_j^N \frac{1_{I_{i,j}}(x)}{W(I_{i,j})} \times P(id_j^{t+1} | id_i^t) \times P(id_i^t) \\
&= \sum_i^N \sum_j^N \frac{1_{I_{i,j}}(x)}{P(id_j^{t+1} | id_i^t) \times 2^n} \times P(id_j^{t+1} | id_i^t) \times P(id_i^t) \\
&= \sum_i^N \sum_j^N \frac{1_{I_{i,j}}(x)}{2^n} \times P(id_i^t)
\end{aligned}$$

$$\forall x \in [0, 2^n - 1], \sum_j^N 1_{I_{i,j}}(x) = 1$$

$$\begin{aligned}
P(id_r = x) &= \sum_i^N \frac{1}{2^n} \times P(id_i^t) = \frac{1}{2^n} \\
H(id_r) &= \sum_{x \in [0, 2^n - 1]} \frac{1}{2^n} \times \log_2\left(\frac{1}{2^n}\right) \\
H(id_r) &= n
\end{aligned}$$

□

Entropy of Arithmetic masking

Proof.

$$\begin{aligned}
H(id_r) &= \sum_{x \in [0, 2^n - 1]} P(id_r = x) \log_2\left(\frac{1}{P(id_r = x)}\right) \\
P(id_r = x) &= \begin{cases} \sum_{i=0}^x \frac{P(id_i)}{2^{n-N+1}} & , \quad x \in [0, N-2] \\ \frac{1}{2^{n-N+1}} & , \quad x \in [N-1, 2^n - N] \\ \sum_{i=x-2^n+N}^{N-1} \frac{P(id_i)}{2^{n-N+1}} & , \quad x \in [2^n - N + 1, 2^n - 1] \end{cases}
\end{aligned}$$

$$\begin{aligned}
 H(id_r) &= \sum_{x \in [N-1, 2^n-N]} \frac{1}{2^n - N + 1} \times \log_2(2^n - N + 1) \\
 &+ \sum_{x \in [0, N-2]} \left[\sum_{i=0}^x \frac{P(id_i)}{2^n - N + 1} \right] \times \log_2\left(\frac{1}{\sum_{i=0}^x \frac{P(id_i)}{2^n - N + 1}}\right) \\
 &+ \sum_{x \in [2^n-N+1, 2^n-1]} \left[\sum_{i=x-2^n+N}^{N-1} \frac{P(id_i)}{2^n - N + 1} \right] \times \log_2\left(\frac{1}{\sum_{i=x-2^n+N}^{N-1} \frac{P(id_i)}{2^n - N + 1}}\right)
 \end{aligned}$$

$$\begin{aligned}
 H(id_r) &= \frac{2^n - 2(N-1)}{2^n - N + 1} \log_2(2^n - N + 1) \\
 &+ \sum_{x \in [0, N-2]} \left[\sum_{i=0}^x \frac{P(id_i)}{2^n - N + 1} \right] \times \log_2\left(\frac{1}{\sum_{i=0}^x \frac{P(id_i)}{2^n - N + 1}}\right) \\
 &+ \left[\sum_{i=x+1}^{N-1} \frac{P(id_i)}{2^n - N + 1} \right] \times \log_2\left(\frac{1}{\sum_{i=x+1}^{N-1} \frac{P(id_i)}{2^n - N + 1}}\right)
 \end{aligned}$$

$$\begin{aligned}
 H(id_r) &= \frac{2^n - 2(N-1)}{2^n - N + 1} \log_2(2^n - N + 1) + \sum_{x \in [0, N-2]} \frac{1}{2^n - N + 1} \log_2(2^n - N + 1) \\
 &+ \sum_{x \in [0, N-2]} \sum_{i=0}^x \frac{P(id_i)}{2^n - N + 1} \times \log_2\left(\frac{1}{\sum_{i=0}^x P(id_i)}\right) \\
 &+ \sum_{i=x+1}^{N-1} \frac{P(id_i)}{2^n - N + 1} \times \log_2\left(\frac{1}{\sum_{i=x+1}^{N-1} P(id_i)}\right)
 \end{aligned}$$

$$\begin{aligned}
 H(id_r) &= \frac{2^n - 2(N-1)}{2^n - N + 1} \log_2(2^n - N + 1) + \frac{N-1}{2^n - N + 1} \log_2(2^n - N + 1) \\
 &+ \frac{1}{2^n - N + 1} \sum_{x \in [0, N-2]} \sum_{i=0}^x P(id_i) \times \log_2\left(\frac{1}{\sum_{i=0}^x P(id_i)}\right) \\
 &+ \sum_{i=x+1}^{N-1} P(id_i) \times \log_2\left(\frac{1}{\sum_{i=x+1}^{N-1} P(id_i)}\right)
 \end{aligned}$$

$$\begin{aligned}
 H(id_r) &= \frac{2^n - N + 1}{2^n - N + 1} \log_2(2^n - N + 1) \\
 &+ \frac{1}{2^n - N + 1} \sum_{x \in [0, N-2]} \sum_{i=0}^x P(id_i) \times \log_2\left(\frac{1}{\sum_{i=0}^x P(id_i)}\right) \\
 &+ \sum_{i=x+1}^{N-1} P(id_i) \times \log_2\left(\frac{1}{\sum_{i=x+1}^{N-1} P(id_i)}\right)
 \end{aligned}$$

6. CONCLUSION

$$\begin{aligned}
H(id_r) &= \log_2(2^n - N + 1) + \frac{1}{2^n - N + 1} \sum_{x \in [0, N-2]} \sum_{i=0}^x P(id_i) \times \log_2\left(\frac{1}{\sum_{i=0}^x P(id_i)}\right) \\
&\quad + \sum_{i=x+1}^{N-1} P(id_i) \times \log_2\left(\frac{1}{\sum_{i=x+1}^{N-1} P(id_i)}\right)
\end{aligned}$$

□

Conditional entropy of Arithmetic masking

The Arithmetic masking conditional entropy is:

$$H_{AM}(id_r|id_o) = \log_2(2^n - N + 1)$$

Proof.

$$\begin{aligned}
P(id_r = x|id_o = id_i) &= \frac{1_{[i-1, 2^n - N + i - 1]}}{2^n - N + 1} \\
H_{AM}(id_r|id_o) &= \sum_{i=0}^N P(id_i) H(id_r|id_o = id_i) \\
H_{AM}(id_r|id_o) &= \sum_{i=0}^N P(id_i) \sum_{x \in [i-1, 2^n - N + i - 1]} P(id_r = x|id_o = id_i) \log_2\left(\frac{1}{P(id_r = x|id_o = id_i)}\right) \\
H_{AM}(id_r|id_o) &= \sum_{i=0}^N P(id_i) \sum_{x \in [i-1, 2^n - N + i - 1]} \frac{1}{2^n - N + 1} \log_2\left(\frac{1}{\frac{1}{2^n - N + 1}}\right) \\
H_{AM}(id_r|id_o) &= \sum_{i=0}^N P(id_i) \log_2(2^n - N + 1) \\
H_{AM}(id_r|id_o) &= \log_2(2^n - N + 1)
\end{aligned}$$

□

Fixed mapping optimality proof

If we adopt a fixed mapping randomization strategy, the optimal solution in terms of conditional entropy is the Frequency-intervals solutions.

Proof. In the context of fixed mapping, we want to find the best decomposition of intervals that maximizes the conditional entropy. We previously showed that the conditional entropy of all fixed mapping solutions can be expressed as: $H(id_r|id_o) = \sum_{i \in [1, N]} P(id_i) \times \log_2(W_i)$, where I_i is the randomization interval of id_i of width $W(I_i)$. For the fixed mapping solutions

the intervals are non overlapping. Besides the width of each interval I_i is positive ($W(I_i) \geq 0$) and their sum equals 2^n . Thus we define the following problem:

$$\underset{\{I_i\}, i \in [1, N]}{\text{Argmax}} H(id_r | id_o) = \sum_i P(id_i) \times \log_2(W_i)$$

Subject to the following constraints:

$$\begin{aligned} h_0 & : \sum_{i \in [1, N]} W_i - 2^n = 0 \\ h_i & : \forall i \in [1, N], -W_i \leq 0 \end{aligned}$$

To find the solution to this problem we use the Lagrangian multiplier:

$$\mathcal{L}(W_1, \dots, W_N, \lambda_1, \dots, \lambda_N, \lambda_0) = H(id_r | id_o) + \sum_{j=0}^N \lambda_j h_j$$

and solve the equation system: $\frac{\partial \mathcal{L}}{\partial W_i} = 0, \quad \forall i \in [1, N]$

$$\frac{\partial \mathcal{L}}{\partial W_i}(W_1, \dots, W_N, \lambda_1, \dots, \lambda_N, \lambda_0) = \frac{\partial H}{\partial W_i} + \sum_{j=0}^N \lambda_j \frac{\partial h_j}{\partial W_i}$$

$$\forall i \in [0, N] : \lambda_i \times h_i = 0$$

$$\begin{aligned} h_0 & : \sum_{i \in [1, N]} W_i - 2^n = 0 \\ h_i & : \forall i \in [1, N], -W_i \leq 0 \end{aligned}$$

We have: $\frac{\partial H}{\partial W_i} = P(id_i) \times \frac{1}{W_i}$ and $\frac{\partial h_0}{\partial W_i} = 1$ and $\frac{\partial h_j}{\partial W_i} = -1$ if $(i = j)$, 0 otherwise

$$\forall i \in [1, N] : P(id_i) \times \frac{1}{W_i} + \lambda_0 - \lambda_i = 0$$

$$\forall i \in [1, N] : \lambda_i \times h_i = 0$$

Resolving this system of equations gives:

$$\lambda_i = 0, \quad \forall i \in [1, N]$$

$$\lambda_0 = \frac{-1}{2^n}$$

Hence:

$$\Rightarrow \forall i \in [1, N] : W_i = P(id_i) \times 2^n$$

□

6. CONCLUSION

Bibliography

- [1] Automotive electronic systems. <https://cecas.clemson.edu/cvel/auto/systems/auto-systems.html>. Accessed: 2018-09-01. xv, 10
- [2] E-safety vehicle intrusion protected applications (evita). <https://www.evita-project.org/>. 5, 20, 22
- [3] Ebios-2010 expression des besoins et identification des objectifs de sécurité. <https://www.ssi.gouv.fr/guide/ebios-2010-expression-des-besoins-et-identification-des-objectifs-de-securite/>. 5, 22
- [4] Experimental security assessment of bmw cars: A summary report. https://keenlab.tencent.com/en/Experimental_Security_Assessment_of_BMW_Cars_by_KeenLab.pdf. 4
- [5] “groove: Graphs for object-oriented verification”. <http://groove.cs.utwente.nl/>. 60, 64
- [6] Healing vulnerabilities to enhance software security and safety (heavens). <https://research.chalmers.se/en/project/5809>. 22
- [7] Iso/sae-21434: Road vehicles – cybersecurity engineering. <https://www.iso.org/standard/70918.html>. 21
- [8] Mcp2515: Stand-alone can controller with spi interface. <http://ww1.microchip.com/downloads/en/DeviceDoc/MCP2515-Stand-Alone-CAN-Controller-with-SPI-20001801J.pdf>. 116
- [9] Oversee: Open vehicular secure platform. <https://www.oversee-project.com/>. 19
- [10] Martín Abadi, Mihai Budiu, Ulfar Erlingsson, and Jay Ligatti. Control-flow integrity. In *Proceedings of the 12th ACM conference on Computer and communications security*, pages 340–353. ACM, 2005. 21
- [11] Mike Adler. An algebra for data flow diagram process decomposition. *IEEE Transactions on Software Engineering*, 14(2):169–183, 1988. 52

BIBLIOGRAPHY

- [12] Amer Aijaz, Bernd Bochow, Florian Dötzer, Andreas Festag, Matthias Gerlach, Rainer Kroh, and Tim Leinmüller. Attacks on inter vehicle communication systems-an analysis. *Proc. WIT*, pages 189–194, 2006. 46
- [13] Christopher J Alberts, Sandra G Behrens, Richard D Pethia, and William R Wilson. Operationally critical threat, asset, and vulnerability evaluation (octave) framework, version 1.0. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 1999. 22
- [14] Sintsov Alexey. Testing CAN Network with help of CANtoolz. <https://www.slideshare.net/AlexeySintsov/testing-can-network-with-help-of-cantoolz>, 2016. Accessed: 2018-01-01. 15, 32, 33, 34
- [15] Paul Ammann, Duminda Wijesekera, and Saket Kaushik. Scalable, graph-based network vulnerability analysis. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 217–224. ACM, 2002. 46, 48
- [16] Daniel Angermeier, Alexander Nieding, and Jörn Eichler. Supporting risk assessment with the systematic identification, merging, and validation of security goals. In *International Workshop on Risk Assessment and Risk-driven Testing*, pages 82–95. Springer, 2016. 44
- [17] Ludovic Apvrille, Letitia Li, and Yves Roudier. Model-driven engineering for designing safe and secure embedded systems. In *Architecture-Centric Virtual Integration (ACVI), 2016*, pages 4–7. IEEE, 2016. 48
- [18] Ludovic Apvrille and Yves Roudier. Sysml-sec attack graphs: compact representations for complex attacks. In *International Workshop on Graphical Models for Security*, pages 35–49. Springer, 2015. 48
- [19] William A Arbaugh, David J Farber, and Jonathan M Smith. A secure and reliable bootstrap architecture. In *Security and Privacy, 1997. Proceedings., 1997 IEEE Symposium on*, pages 65–71. IEEE, 1997. 20
- [20] Johan Bengtsson, Kim Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Up-paal tool suite for automatic verification of real-time systems. In *International Hybrid Systems Workshop*, pages 232–243. Springer, 1995. 48
- [21] Bruno Blanchet, Ben Smyth, and Vincent Cheval. Proverif 1.93: Automatic cryptographic protocol verifier, user manual and tutorial. *Internet*[cited June 2016], Available from: <https://www.bensmyth.com/publications/2010-ProVerif-manualversion-1.93>, 2016. 48
- [22] P Borazjani, C Everett, and Damon McCoy. Octane: An extensible open source car security testbed. In *Proceedings of the Embedded Security in Cars Conference*, 2014. 40
- [23] Alexandre Bouard, Hendrik Schweppe, Benjamin Weyl, and Claudia Eckert. Leveraging in-car security by combining information flow monitoring techniques. *VEHICULAR 2013*, page 6, 2013. 21

- [24] Manfred Broy. Automotive software and systems engineering. In *Proceedings of the 2nd ACM/IEEE International Conference on Formal Methods and Models for Co-Design*, pages 143–149. IEEE Computer Society, 2005. 10
- [25] Eric J Byres, Matthew Franz, and Darrin Miller. The use of attack trees in assessing vulnerabilities in scada systems. In *Proceedings of the international infrastructure survivability workshop*, pages 3–10. Citeseer, 2004. 46
- [26] Robert N Charette. This car runs on code. *IEEE spectrum*, 46(3):3, 2009. 9
- [27] Madeline Cheah, Hoang Nga Nguyen, Jeremy Bryans, and Siraj A Shaikh. Formalising systematic security evaluations using attack trees for automotive applications. In *IFIP International Conference on Information Security Theory and Practice*, pages 113–129. Springer, 2017. 45, 46, 48
- [28] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, Tadayoshi Kohno, et al. Comprehensive experimental analyses of automotive attack surfaces. In *USENIX Security Symposium*. San Francisco, 2011. 15, 16, 33
- [29] Kyong-Tak Cho and Kang G Shin. Fingerprinting electronic control units for vehicle intrusion detection. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 911–927. USENIX Association, 2016. 31, 33, 34, 39, 41
- [30] Christian S. Collberg and Clark Thomborson. Watermarking, tamper-proofing, and obfuscation-tools for software protection. *IEEE Transactions on software engineering*, 28(8):735–746, 2002. 20
- [31] SAE Vehicle Electrical System Security Committee et al. Sae j3061-cybersecurity guidebook for cyber-physical automotive systems. *SAE-Society of Automotive Engineers*, 2016. 44
- [32] Vehicle Electrical System Security Committee. Sae j3061-cybersecurity guidebook for cyber-physical automotive systems. 2016. xv, 21, 22
- [33] Barbara J Czerny. System security and system safety engineering: Differences and similarities and a system security engineering process based on the iso 26262 process framework. *SAE International Journal of Passenger Cars-Electronic and Electrical Systems*, 6(2013-01-1419):349–359, 2013. 44
- [34] Stabili Dario, Marchetti Mirco, and Colajanni Michele. Detecting attacks to internal vehicle networks through hamming distance. In *IEEE 2017 AEIT International Annual Conference-Infrastructures for Energy and ICT (AEIT 2017)*, 2017. 40, 41
- [35] Eloi de Chérisey, Sylvain Guilley, Annelie Heuser, and Olivier Rioul. On the optimality and practicability of mutual information analysis in some scenarios. *Cryptography and Communications*, 10(1):101–121, 2018. 80
- [36] Trajce Dimkov, Wolter Pieters, and Pieter Hartel. Portunes: representing attack scenarios spanning through the physical, digital and social domain. In *Joint Workshop*

BIBLIOGRAPHY

- on Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security*, pages 112–129. Springer, 2010. 46, 49, 60
- [37] Morris J Dworkin. Recommendation for block cipher modes of operation: The cmac mode for authentication. Technical report, 2016. 36
- [38] Jung-Ho Eom, Min-Woo Park, Seon-Ho Park, and Tai-Myoung Chung. A framework of defense system for prevention of insider’s malicious behaviors. In *Advanced Communication Technology (ICACT), 2011 13th International Conference on*, pages 982–987. IEEE, 2011. 46
- [39] TS ETSI. 102 165-1: "telecommunications and internet converged services and protocols for advanced networking (tispan). *Methods and protocols*, pages 2011–03. 5, 22
- [40] ETSI/SAGE. KASUMI specification, version: 1.0. *3GPP Confidentiality and Integrity Algorithms*, 1999. 35
- [41] PUB FIPS. 198 (federal information processing standards publication) the keyed hash message authentication code (hmac). *Information Technology Laboratory, National Institute of Standards and Technology, Gaithersburg, MD*, pages 20899–8900. 36
- [42] Ian D Foster, Andrew Prudhomme, Karl Koscher, and Stefan Savage. Fast and vulnerable: A story of telematic failures. In *WOOT*, 2015. 16
- [43] Olga Gadyatskaya, Ravi Jhawar, Piotr Kordy, Karim Lounis, Sjouke Mauw, and Rolando Trujillo-Rasua. Attack trees for practical security assessment: ranking of attack scenarios with adtool 2.0. In *International Conference on Quantitative Evaluation of Systems*, pages 159–162. Springer, 2016. 47
- [44] Olga Gadyatskaya, Ravi Jhawar, Sjouke Mauw, Rolando Trujillo-Rasua, and Tim AC Willemse. Refinement-aware generation of attack trees. In *International Workshop on Security and Trust Management*, pages 164–179. Springer, 2017. 45
- [45] Amir Hossein Ghamarian, Maarten de Mol, Arend Rensink, Eduardo Zambon, and Maria Zimakova. Modelling and analysis using groove. *International journal on software tools for technology transfer*, 14(1):15–40, 2012. 59, 60
- [46] André Groll, Jan Holle, Christoph Ruland, Marko Wolf, Thomas Wollinger, and Frank Zweers. Oversee a secure and open communication and runtime platform for innovative automotive applications. In *7th Embedded Security in Cars Conf.(ESCAR)*, 2009. 19
- [47] Bogdan Groza, Stefan Murvay, Anthony Van Herrewege, and Ingrid Verbauwhede. Libracan: a lightweight broadcast authentication protocol for controller area networks. In *International Conference on Cryptology and Network Security*, pages 185–200. Springer, 2012. 36
- [48] Kyusuk Han, André Weimerskirch, and Kang G Shin. Automotive cybersecurity for in-vehicle communication. In *IQT QUARTERLY*, volume 6, pages 22–25, 2014. 37, 77, 81

- [49] Kyusuk Han, André Weimerskirch, and Kang G Shin. A practical solution to achieve real-time performance in the automotive network by randomizing frame identifier. In *Proc. Eur. Embedded Secur. Cars (ESCAR)*, pages 13–29, 2015. 37, 77, 81
- [50] Oliver Hartkopp, Cornel Reuber, and Roland Schilling. MaCAN - Message Authenticated CAN. In *Escar Conference, Berlin, Germany*, 2012. 35, 36
- [51] Reiko Heckel. Graph transformation in a nutshell. *Electronic notes in theoretical computer science*, 148(1):187–198, 2006. 59
- [52] Olaf Henniger, Ludovic Apvrille, Andreas Fuchs, Yves Roudier, Alastair Ruddle, and Benjamin Weyl. Security requirements for automotive on-board networks. In *Intelligent Transport Systems Telecommunications, (ITST), 2009 9th International Conference on*, pages 641–646. IEEE, 2009. xv, xix, 22, 24, 46, 47, 68
- [53] Steven A Hofmeyr, Stephanie Forrest, and Anil Somayaji. Intrusion detection using sequences of system calls. *Journal of computer security*, 6(3):151–180, 1998. 21
- [54] Tobias Hoppe, Stefan Kiltz, and Jana Dittmann. Security threats to automotive can networks—practical examples and selected short-term countermeasures. In *International Conference on Computer Safety, Reliability, and Security*, pages 235–248. Springer, 2008. 15, 31, 34, 39
- [55] Tobias Hoppe, Stefan Kiltz, and Jana Dittmann. Security threats to automotive can networks—practical examples and selected short-term countermeasures. *Reliability Engineering & System Safety*, 96(1):11–25, 2011. 33
- [56] Abdulmalik Humayed and Bo Luo. Using ID-Hopping to Defend Against Targeted DoS on CAN. In *Proceedings of the 1st International Workshop on Safe Control of Connected and Autonomous Vehicles*, pages 19–26. ACM, 2017. 37, 77
- [57] Kyle Ingols, Richard Lippmann, and Keith Piwowarski. Practical attack graph generation for network defense. In *Computer Security Applications Conference, 2006. ACSAC’06. 22nd Annual*, pages 121–130. IEEE, 2006. 48
- [58] Rob Millerb Ishtiaq Roufa, Hossen Mustafaa, Sangho Ohb Travis Taylora, Wenyan Xua, Marco Gruteserb, Wade Trappeb, and Ivan Seskarb. Security and privacy vulnerabilities of in-car wireless networks: A tire pressure monitoring system case study. In *19th USENIX Security Symposium, Washington DC*, pages 11–13, 2010. 16
- [59] Mafijul Md Islam, Aljoscha Lautenbach, Christian Sandberg, and Tomas Olovsson. A risk assessment framework for automotive embedded systems. In *Proceedings of the 2nd ACM International Workshop on Cyber-Physical System Security*, pages 3–14. ACM, 2016. 44
- [60] ISO. 11898-1—Road vehicles—Controller area network (CAN)—Part 1: Data link layer and physical signalling. *International Organization for Standardization*, 2003. 27
- [61] ISO. 11898-2—Road vehicles—Controller area network (CAN)—Part 2: High-speed medium access unit. *International Organization for Standardization*, 2003. 27

BIBLIOGRAPHY

- [62] ISO. 11898-3–Road vehicles–Controller area network (CAN)–Part 2: Fault tolerant medium access unit. *International Organization for Standardization*, 2003. 27
- [63] ISO. ISO 26262-5:Road vehicles – Functional safety – Part 5: Product development at the hardware level. *International Organization for Standardization*, 2011. 98
- [64] ISO/IEC. ISO/IEC 18045–Information technology – Security techniques – Methodology for IT security evaluation. *International Organization for Standardization*, 2008. 47
- [65] ISO/IEC. ISO/IEC 15408–Information technology – Security techniques – Evaluation criteria for IT security. *International Organization for Standardization*, 2009. 47
- [66] Marieta Georgieva Ivanova, Christian W Probst, René Rydhof Hansen, and Florian Kammüller. Transforming graphical system models to graphical attack models. In *International Workshop on Graphical Models for Security*, pages 82–96. Springer, 2015. 49
- [67] Sushil Jajodia and Steven Noel. Topological vulnerability analysis. In *Cyber situational awareness*, pages 139–154. Springer, 2010. 46, 48, 49
- [68] Ravi Jhawar, Barbara Kordy, Sjouke Mauw, Saša Radomirović, and Rolando Trujillo-Rasua. Attack trees with sequential conjunction. In *IFIP International Information Security Conference*, pages 339–353. Springer, 2015. 46
- [69] David Joyner, Ondřej Čertík, Aaron Meurer, and Brian E Granger. Open source computer algebra systems: SymPy. *ACM Communications in Computer Algebra*, 45(3/4):225–234, 2012. 65
- [70] Min-Joo Kang and Je-Won Kang. Intrusion detection system using deep neural network for in-vehicle network security. *PloS one*, 11(6):e0155781, 2016. 40, 41
- [71] Khaled Karray, Jean-Luc Danger, Sylvain Guilley, and M Abdelaziz Elaabid. Attack tree construction and its application to the connected vehicle. *Cyber-Physical Systems Security*, page 175, 2018. 43
- [72] Khaled Karray, Jean-Luc Danger, Sylvain Guilley, and M Abdelaziz Elaabid. Identifier randomization: An efficient protection against can-bus attacks. *Cyber-Physical Systems Security*, page 219, 2018. 77
- [73] Khaled Karray, Jean-Luc Danger, Sylvain Guilley, and M Abdelaziz Elaabid. Prediction-based intrusion detection system for in-vehicle networks using supervised learning and outlier-detection. In *IFIP International Workshop on Information Security Theory and Practice*, 2018. 97
- [74] Pierre Kleberger, Tomas Olovsson, and Erland Jonsson. Security aspects of the in-vehicle network in the connected car. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pages 528–533. IEEE, 2011. 20
- [75] M Kleine-Budde et al. Socketcan–the official can api of the linux kernel. In *Proceedings of the 13th International CAN Conference (iCC 2012), Hambach Castle, Germany CiA*, pages 05–17, 2012. 116

- [76] Barbara Kordy, Piotr Kordy, Sjouke Mauw, and Patrick Schweitzer. Adtool: security analysis with attack–defense trees. In *International Conference on Quantitative Evaluation of Systems*, pages 173–176. Springer, 2013. 47
- [77] Barbara Kordy, Sjouke Mauw, Saša Radomirović, and Patrick Schweitzer. Foundations of attack–defense trees. In *International Workshop on Formal Aspects in Security and Trust*, pages 80–95. Springer, 2010. 45, 68
- [78] Barbara Kordy, Ludovic Piètre-Cambacédès, and Patrick Schweitzer. Dag-based attack and defense modeling: Don’t miss the forest for the attack trees. *Computer science review*, 13:1–38, 2014. 47
- [79] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, et al. Experimental security analysis of a modern automobile. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 447–462. IEEE, 2010. 15
- [80] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. Hmac: Keyed-hashing for message authentication. Technical report, 1997. 36
- [81] Rajesh Kumar, Enno Ruijters, and Mariëlle Stoelinga. Quantitative attack tree analysis via priced timed automata. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 156–171. Springer, 2015. 47
- [82] Hyunsung Lee, Seong Hoon Jeong, and Huy Kang Kim. Otids: A novel intrusion detection system for in-vehicle network by using remote frame. In *2017 15th Annual Conference on Privacy, Security and Trust (PST)*, pages 57–5709. IEEE, 2017. 39, 41
- [83] Gabriele Lenzini, Sjouke Mauw, and Samir Ouchani. Security analysis of socio-technical physical systems. *Computers & electrical engineering*, 47:258–274, 2015. 46, 49
- [84] Cullen Linn and Saumya Debray. Obfuscation of executable code to improve resistance to static disassembly. In *Proceedings of the 10th ACM conference on Computer and communications security*, pages 290–299. ACM, 2003. 20
- [85] Isograph LTD. AttackTree. <https://www.isograph.com/software/attacktree/>, 2019. Accessed: 2019-01-01. 47
- [86] Florian Lugou, Letitia W Li, Ludovic Apvrille, and Rabéa Ameer-Boulifa. Sysml models and model transformation for security. In *Conférence on Model-Driven Engineering and Software Development (Modelsward’2016)*, 2016. 48
- [87] Tobias Madl, Jasmin Brückmann, and Hans-Joachim Hof. Can obfuscation by randomization (canora). 2017. 37
- [88] Kevin Mahaffey. Hacking a tesla model s: What we found and what we learned, 2015. 4
- [89] Mirco Marchetti and Dario Stabili. Anomaly detection of CAN bus messages through analysis of ID sequences. In *Intelligent Vehicles Symposium (IV), 2017 IEEE*, pages 1577–1583. IEEE, 2017. 39, 41

BIBLIOGRAPHY

- [90] Edward J Markey. Tracking & hacking: Security & privacy gaps put american drivers at risk. *US Senate*, 2015. 4
- [91] Fabio Martinelli, Francesco Mercaldo, Vittoria Nardone, Albina Orlando, and Antonella Santone. Who’s driving my car? a machine learning based approach to driver identification, 2018. 15, 32
- [92] Fabio Martinelli, Francesco Mercaldo, Albina Orlando, Vittoria Nardone, Antonella Santone, and Arun Kumar Sangaiah. Human behavior characterization for driving style recognition in vehicle system. *Computers & Electrical Engineering*, 2018. 15, 32
- [93] Sjouke Mauw and Martijn Oostdijk. Foundations of attack trees. In *International Conference on Information Security and Cryptology*, pages 186–198. Springer, 2005. 45, 46, 47, 68
- [94] Charlie Miller and Chris Valasek. Adventures in automotive networks and control units. *DEF CON*, 21:260–264, 2013. 15, 31, 33, 34, 39
- [95] Charlie Miller and Chris Valasek. Remote exploitation of an unaltered passenger vehicle. *Black Hat USA*, 2015, 2015. xv, 4, 13, 14, 15, 16, 17, 39, 41
- [96] Kai-Uwe Müller, Robin Ulrich, Alexander Stanitzki, and Rainer Kokozinski. Enabling secure boot functionality by using physical unclonable functions. In *2018 14th Conference on Ph. D. Research in Microelectronics and Electronics (PRIME)*, pages 81–84. IEEE, 2018. 20
- [97] Philipp Mundhenk, Artur Mrowca, Sebastian Steinhorst, Martin Lukasiewicz, Suhaib A Fahmy, and Samarjit Chakraborty. Open source model and simulator for real-time performance analysis of automotive network security. *Acm Sigbed Review*, 13(3):8–13, 2016. 36
- [98] Philipp Mundhenk, Andrew Paverd, Artur Mrowca, Sebastian Steinhorst, Martin Lukasiewicz, Suhaib A Fahmy, and Samarjit Chakraborty. Security in automotive networks: Lightweight authentication and authorization. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 22(2):25, 2017. 36
- [99] Michael Müter and Naim Asaj. Entropy-based anomaly detection for in-vehicle networks. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pages 1110–1115. IEEE, 2011. 40, 41
- [100] Michael Müter, André Groll, and Felix C Freiling. A structured approach to anomaly detection for in-vehicle networks. In *Information Assurance and Security (IAS), 2010 Sixth International Conference on*, pages 92–98. IEEE, 2010. 39, 40, 41
- [101] Sandeep Nair Narayanan, Sudip Mittal, and Anupam Joshi. OBDSecureAlert: An Anomaly Detection System for Vehicles. In *Smart Computing (SMARTCOMP), 2016 IEEE International Conference on*, pages 1–6. IEEE, 2016. 40, 41
- [102] Dennis K Nilsson, Ulf E. Larson, and Erland Jonsson. Efficient in-vehicle delayed data authentication based on compound message authentication codes. In *Vehicular Technology Conference, 2008. VTC 2008-Fall. IEEE 68th*, pages 1–5. IEEE, 2008. 35

- [103] Xinming Ou, Sudhakar Govindavajhala, and Andrew W Appel. Mulval: A logic-based network security analyzer. In *USENIX security*, 2005. 48
- [104] Andrea Palanca, Eric Evenchick, Federico Maggi, and Stefano Zanero. A stealth, selective, link-layer denial-of-service attack against automotive networks. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 185–206. Springer, 2017. 32, 33, 34
- [105] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011. 120, 127
- [106] Cynthia Phillips and Laura Painton Swiler. A graph-based system for network-vulnerability analysis. In *Proceedings of the 1998 workshop on New security paradigms*, pages 71–79. ACM, 1998. 48
- [107] Stjepan Picek, Annelie Heuser, Alan Jovic, Shivam Bhasin, and Francesco Regazzoni. The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(1):209–237, 2019. 127
- [108] Sophie Pinchinat, Mathieu Acher, and Didier Vojtisek. Towards synthesis of attack trees for supporting computer-aided risk analysis. In *International Conference on Software Engineering and Formal Methods*, pages 363–375. Springer, 2014. 45
- [109] Arend Rensink. The groove simulator: A tool for state space generation. In *International Workshop on Applications of Graph Transformations with Industrial Relevance*, pages 479–485. Springer, 2003. 60
- [110] Ronald W Ritchey and Paul Ammann. Using model checking to analyze network vulnerabilities. In *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*, pages 156–165. IEEE, 2000. 48
- [111] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986. 116
- [112] Martin Salfer and Claudia Eckert. Attack surface and vulnerability assessment of automotive electronic control units. In *e-Business and Telecommunications (ICETE), 2015 12th International Joint Conference on*, volume 4, pages 317–326. IEEE, 2015. 46, 48, 52
- [113] Martin Salfer, Hendrik Schweppe, and Claudia Eckert. Efficient attack forest construction for automotive on-board networks. In *International Conference on Information Security*, pages 442–453. Springer, 2014. 46, 48
- [114] Bruce Schneier. Attack trees. *Dr. Dobbs’s journal*, 24(12):21–29, 1999. 22, 45, 68
- [115] Hendrik Schweppe and Yves Roudier. Security and privacy for in-vehicle networks. In *Vehicle Communications, Sensing, and Computing (VCSC), 2012 IEEE 1st International Workshop on*, pages 12–17. IEEE, 2012. 21

BIBLIOGRAPHY

- [116] Hendrik Schweppe, Yves Roudier, Benjamin Weyl, Ludovic Apvrille, and Dirk Scheuermann. Car2x communication: securing the last meter-a cost-effective approach for ensuring trust in car2x applications using in-vehicle symmetric cryptography. In *Vehicle Technology Conference (VTC Fall), 2011 IEEE*, pages 1–5. IEEE, 2011. 36
- [117] Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeannette M Wing. Automated generation and analysis of attack graphs. In *Security and privacy, 2002. Proceedings. 2002 IEEE Symposium on*, pages 273–284. IEEE, 2002. 46, 48
- [118] Craig Smith. *The Car Hacker's Handbook: A Guide for the Penetration Tester*. No Starch Press, 2016. 15, 32, 34
- [119] Hyun Min Song, Ha Rang Kim, and Huy Kang Kim. Intrusion detection system based on the analysis of time intervals of can messages for in-vehicle network. In *2016 international conference on information networking (ICOIN)*, pages 63–68. IEEE, 2016. 39, 41
- [120] CAN Specification. Version 2.0. *Robert Bosch GmbH*, 1991. 27
- [121] Ivan Studnia, Eric Alata, Vincent Nicomette, Mohamed Kaâniche, and Youssef Laarouchi. A language-based intrusion detection approach for automotive embedded networks. In *The 21st IEEE Pacific Rim International Symposium on Dependable Computing (PRDC 2015)*, 2014. 39, 41
- [122] Ivan Studnia, Eric Alata, Vincent Nicomette, Mohamed Kaâniche, and Youssef Laarouchi. A language-based intrusion detection approach for automotive embedded networks. *International Journal of Embedded Systems*, 10(1), 2018. 39, 41
- [123] Ivan Studnia, Vincent Nicomette, Eric Alata, Yves Deswarte, Mohamed Kaaniche, and Youssef Laarouchi. Survey on security threats and protection mechanisms in embedded automotive networks. In *2013 43rd Annual IEEE/IFIP Conference on Dependable Systems and Networks Workshop (DSN-W)*, pages 1–12. IEEE, 2013. 20
- [124] Adrian Taylor, Nathalie Japkowicz, and Sylvain Leblanc. Frequency-based anomaly detection for the automotive CAN bus. In *Industrial Control Systems Security (WCICSS), 2015 World Congress on*, pages 45–49. IEEE, 2015. 39, 41
- [125] Adrian Taylor, Sylvain Leblanc, and Nathalie Japkowicz. Anomaly detection in automobile control network data with long short-term memory networks. In *Data Science and Advanced Analytics (DSAA), 2016 IEEE International Conference on*, pages 130–139. IEEE, 2016. 39, 41
- [126] Amenaza Technologies. SecurITree. <https://www.amenaza.com//>, 2001-2018. Accessed: 2019-01-01. 47
- [127] Chee-Wooi Ten, Chen-Ching Liu, and Manimaran Govindarasu. Vulnerability assessment of cybersecurity for scada systems using attack trees. In *Power Engineering Society General Meeting, 2007. IEEE*, pages 1–8. IEEE, 2007. 46

- [128] C Valasek and C Miller. A survey of remote automotive attack surfaces. *Scribd, Washington, USA*, 2014. 18
- [129] Anthony Van Herrewege, Dave Singelee, and Ingrid Verbauwhede. Canauth-a simple, backward compatible broadcast authentication protocol for can bus. In *ECRYPT Workshop on Lightweight Cryptography*, volume 2011, 2011. 35, 36
- [130] David Wagner and R Dean. Intrusion detection via static analysis. In *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on*, pages 156–168. IEEE, 2001. 21
- [131] Marko Wolf, André Weimerskirch, and Thomas Wollinger. State of the art: Embedding security in vehicles. *EURASIP Journal on Embedded Systems*, 2007(1):074706, 2007. 20