An autonomous vehicle in a connected environment: case study of cyber-resilience

Guillaume Hutzler*, Hanna Klaudel*, Witold Klaudel†‡, Franck Pommereau* and Artur Rataj‡
* IBISC, Univ. Evry, Université Paris-Saclay, France
Email: {guillaume.hutzler, hanna.klaudel, franck.pommereau}@univ-evry.fr
† SafeTech Cybernetics, Palaiseau, France, Email: witold.klaudel@outlook.fr
‡ IRT SystemX, Palaiseau, France, Email: artur.rataj@irt-systemx.fr

***Abstract*—As the advancing autonomy of vehicles requires increasing assistance from the surrounding infrastructure, it becomes clear that the potential for cyberattacks necessitates a sophisticated implementation of resilience, capable of detecting and responding to both internal and external threats. Therefore, threat analysis and risk assessment, including careful modelling of resilience, are essential to prepare against cybersecurity risks. In this context, we extend our method of an automatic discovery of cost-ranked cyberattack scenarios by monitoring/fallback mechanisms. We then demonstrate that this extension allows an analysis of a realistic resilient model of cybersecurity aspects of a level 2 autonomous vehicle in a connected environment.**

***Index Terms*—security evaluation, formal modelling**

I. Introduction

AUTONOMOUS vehicles, still in the experimental stage, are far, from a technical and legal point of view, from established complex resilient systems such as e.g. energy networks. However, recent progress demonstrates that an autonomous vehicle will be part of large infrastructure systems, including elements such as manufacturer's diagnostic supervision, dealer authorisation, various services such as geospatial or even road infrastructure. Obviously, this shapes resilience goals. We reflect this by immersing, in our case study, an autonomous vehicle into its environment and modelling attacks that often cross the boundary between these two parties.

Technological advances have expanded the cyberattack surface of distributed information systems, and specifically critical ones such as autonomous vehicles, as they have become more complex and also more connected with the external world. This resulted in sophisticated implementations of resilience understood as active defence, capable to detect attacks and react to them [1]. Careful modelling of resilience mechanism may be necessary, given that a number of new applications of complex distributed information systems concern critical systems [2]. This is the case of the autonomous vehicle studied in this paper.

In order to meet these new expectations, we extended our framework, called SCORE [3], which is devoted to build a suitable automata model of attack propagation in the system and automatically discover complex cyberattack scenarios using abstract cost criteria. The new extension adds to SCORE a complementary monitoring/fallback mechanism implementing resilience on an abstraction level compatible with a Model-Based Systems Engineering (MBSE) architectural diagram of hardware and software. We then show that together with an extraction of security traits from a heterogeneous non-security oriented MBSE data, this allows analysing a realistic resilient model of the cybersecurity aspect of an autonomous vehicle. The automated security analysis of architectural traits provided by SCORE greatly optimises an analyst's work by allowing her/him to focus less on engineering aspects and more on purely cybersecurity related concepts like abstract costs of unitary compromise of software components.

Identifying threats to distributed information systems is challenging as they consist of many different entities exposed to a wide range of cyberattacks. A cyberattack is understood here as a sequence of unitary actions taken by an *attacker* to take control over some components of the system. This sequence starts with one or more entry points and ends with the loss of integrity of some system components leading in turn to a damage targeted by the attacker. Identifying cyberattacks is crucial for optimising both an architecture and cybersecurity features to meet an acceptable level of risk. It is necessary to calculate likelihood of cyberattacks and thus contribute to the estimation of the overall risk level.

When it comes to interpreting the concept of likelihood, many national cybersecurity agencies (e.g., NIST [4], ANSSI [5]) issue loosely defined recommendations and defer final decisions for its rating scale and methods to experts. The choice of scale and methods for assessing the strength of cyber protections is still an open research problem, *e.g.* [6]. In SCORE we lean towards the ANSSI approach translating the likelihood into an inversely proportional cost. We compute the cost of the entire attack as a sum of costs of all its unitary attacks, assuming that experts can provide the scale and value of costs associated with breaking the cyber protections of the systems under analysis.

Paper structure: We first provide a related work and a comparison between SCORE method and the existing ones (Sec. II). Then, in Sec. III we remind the essential characteristics of SCORE and Sec. IV extends it with resilience features such as monitoring and fallback. In Section V we apply SCORE to the cyber resilience risk assessment of a realistic electronic onboard architecture of autonomous vehicle. We develop a propagation model of this case study and discuss the compliance of the chosen security protections wrt the acceptable risk level.

TABLE I
FEATURE AND MODEL SIZE COMPARISON.

	Automatic calculation					Case study		
Method	Network access	Coalition	Peer position	MITM position	Active defence	Edges	Nodes	Hard. nodes
Static AT or ADT [7], [8]	n.a.[a]	+	n.a.[a]	n.a.[a]	-	<20	<20[b]	n.a.[c]
Architecture to AG [9], [10]	+	+	-	-	+	<20	<20[d]	n.a.[c]
Architecture to AT and AG [11]	+	+	-	-	+	<20	<20[d]	n.a.[c]
Architecture/VDB to AG [12]	+	+	-	-	-	<100	<20[d]	<20
Archit./Assert./VDB to AG [13]	-	-	+	-	-	<1000	<1000[b]	n.a.[c]
Archit./VDB to AG/BDD [14]	-	-	-	-	-	>50k	>50k[d]	n.a.[c]
Our approach SCORE	+	+	+	+	+	<10k	<100[d]	<100

[a] Because of lack of architecture. [b] Attacker's sub-goals [8]. [c] Architecture is not layered.
[d] Software nodes or *de-facto* software (a hardware node with no internal software structure given).

II. Contribution and related work

The first non-trivial approaches to vulnerability models were static Attack Trees (AT) [7], [15], [16], where nodes represented logical operations like an AND gate showing the necessity for several parallel breaches before an attack can proceed — we call it a coalition. While static, these trees could be created by various manual procedures [17] e.g. motivated by a security property of interest. Later, ATs were extended to include defence mechanisms to form Attack-Defence Trees (ADT) [8].

Due to the increasing complexity of distributed systems' architecture, the overhead of identifying attack propagation paths between architectural components grew exponentially. This posed a burden for security analysts and presented a potential source of human error. The necessity of devising an automated analysis of the topology of a complex distributed system became obvious. [9] proposed a method of creating a coincidence matrix between a number of architectural components like servers or routers, producing thus a general Attack Graph (AG) to be processed e.g. by a model checker in order to find possible attack sequences. We see that method as one of the first attempts at reusing a traditional MBSE architectural data for cybersecurity, then extended in numerous ways [18], [19], [20].

In SCORE, starting from hardware and software system architectures annotated by cybersecurity protection features, we build an attack propagation model in terms of a network of automata, each automaton modelling a software module of the system. Each automaton evolves according to a set of attack propagation rules computing the cost for each state change as a function of states of contributing neighbouring software components. The automaton state represents the type of software component compromise, called its status, and may be:

- nominal, not yet compromised, thus unable to propagate;
- active malware, which has largest spectrum of possible propagation,
- passive (called bad data), which may only send a corrupted data through the network, or
- non-available, which represents a component completely disabled by the attacker, so that even the attacker cannot use it anymore.

We compare our framework to representatives of different families of methods with respect to the attack structures and the size of systems under analysis in Tab. I.

Here we present an overview of features allowed by SCORE:

- all software components are divided into three classes: user, root and kernel, corresponding in our approach to system processes;
- we employ *functional interactions* representing producer-consumer relations between user software components;
- we consider *system interactions* allowing to model basic operating system relations, like a cheap attack from a kernel to a process it manages or an attack against a kernel via a network interface controlled by that kernel;
- thanks to the automatic calculation of possible flow of interactions through the hardware layer, attacks may take advantage from the so called attacker's position, which can be Man-In-The-Middle (MITM) or Peer, depending on the relationship between the attacking component with the interaction it attacks;
- we also classify certain elements in order to decrease the number of free security parameters like unitary attack costs; this eases the work of a security analyst and increases the manageability of a model; a unitary attack cost consists of a protocol and a component breach;
- thanks to the modelling of the system as a network of automata, we are able to model synchronous attacks (coalition) in which an attacker possibly propagates in a non-sequential way.

See that a component status and an attacker's position, combined with the class of a software component, can form together a rich Cartesian product whose tuples (like *malware* performing a *MITM* attack against a *user component*) can be used in security properties, like for example arrays of unitary costs. We see the abstract nature of such tuples, as opposed to concrete attack descriptions (like a worm X uses a vulnerability Y against a component version Z). This high level of abstraction results from the motivation behind the method: estimation of general resilience based on architectural traits and not an identification of concrete vulnerabilities/exploits as understood e.g. within the CVE database [21]. This is a substantial difference to attack graph-based vulnerability/exploit search tools [22], [10], [11] often connected to popular vulnerability databases [23], [12], or analysers of the source code of concrete IoT devices [24].

Functional and system interactions are a unique trait of our approach. They are similar to the notion of trust in [13]. Each interaction may be routed through the network or through the kernel if located on the same hardware component. In our approach, the routing possibilities of each interaction are pre-calculated statically on the basis of all possible network transmission paths between interaction partners. These paths may be seen as a Boolean function of the interaction accessibility for the attack propagation; it means that if it evaluates to false the attack cannot propagate through the interaction.

As [9] already noticed, solving attack graphs can be much more numerically intensive than solving attack trees. Thus various methods combining trees and graphs [25], [26] or assuming the criterion of monotonicity [27] *i.e.,* that a unitary attack may not decrease further attacking capabilities. Very large networks have been analysed thanks to the latter property [14] combined with binary decision diagrams [28]. However, using the above mentioned abstract classification with a very limited number of classes and a simple abstract model behaviour of a software component, a modern model checker on a fast hardware was able to solve in a reasonable time an involved MBSE model (Sec. V) without the monotonicity assumption. We take advantage of the latter which allows more realistic attack scenarios. For example, an attacker can disable a router or trigger a monitoring system, both potentially limiting the possibility of further propagation.

III. Formal description

We shortly remind the basic elements of SCORE before introducing its extension with monitoring and fallback. More detailed description of the SCORE approach may be found in [3].

The SCORE system specification starts with the definition of the hardware and software architectures and the respective mapping. Each software component exposes interfaces, which are necessary for its normal functioning but can be abused by an attacker to penetrate the system. Interface exposure depends on functional complexity of the system but also on hardware architecture constraints such as resource sharing and network routing. It is assumed that an attacker can pass through any interface but possibly with different abstract costs. The definition of these costs is another part of the system specification that is used to generate the propagation model.

Fig. 2 and 3 show an example of the hardware and software layers. Hardware components are connected by undirected network links while software components are connected by directed functional interactions encapsulated in communication protocols, *e.g.* HTTP for web applications. Software interaction endpoints are attached to software components through software ports called *roles*, *e.g.* client and server for HTTP protocol. Software components are hosted by hardware ones and functional interactions are routed through network links. SCORE assumes that each software component is assigned to exactly one hardware component. A privileged software component (kernel) may also manage its hardware host, including subordinate software components within the same hardware host and network or intra-system routing rules. The hardware link types correspond to the communication link types such as for example, Ethernet or CAN network; they connect components through hardware ports. Hardware components are managed by operating systems (kernels). Routing rules must ensure a physical realisation of all specified functional interactions. Often, due to technical and organisational constraints, routing rules are more permissive than necessary, which can result in the creation of additional attack opportunities.

The propagation model generation goes through an intermediate construction, called the *visibility graph*, which results from the synthesis of the architecture and focuses only on the information flow allowing propagation of the attacker in the system. The visibility graph is a directed graph indicating which software component (node) can propagate its corrupted status to its successors. Each interaction between two nodes is directed and gives rise to an edge in the visibility graph connecting a node to the role of its target node. Edges are labelled by the *attacker position* relative to this interaction:

- `peer` if the attacking node is a functional peer within that interaction or if the edge points to a system role;
- `mitm` (man in the middle) if the attacking node is on a routing path taken by the interaction;
- `side` if the routing rules merely allow the attacking node to see the target role but the node is not in `peer` or in `mitm` positions.

In order for the interaction to be effective, at least one communication routing path should be available, *i.e.,* the statuses of router nodes on this path should be different from non-available. This is expressed statically by a Boolean formula obtained from the architecture when generating the visibility graph.

To obtain an analysable propagation model P with secrets, the visibility graph is completed by some auxiliary information such as thresholds, secrets and roles' categories and the abstract costs of unitary attacks, set by experts. P is essentially composed of a network of automata $\{A_1, \ldots, A_{|N|}\}$ (one for each node $n \in N$ of the visibility graph), each of them being in its current state in $L = \{\mathcal{F}, \mathcal{N}, \mathcal{B}, \mathcal{M}\}$ meaning respectively functional, not available, bad-data or malware, and secrets in S refer to security keys protecting communication sessions, which can be stolen from the nodes where they are stored. All the automata in $\{A_1, \ldots, A_{|N|}\}$ have identical structure, *i.e.,* the same states and transitions, see Fig. 1, but different transition firing conditions. The last may be complex and depend on the type of the software component and on its connectivity with other components in the visibility graph. Detailed definitions of transition conditions are provided in [3].

A. Propagation model dynamics

A *configuration* of the propagation model P with secrets is the state of the network of automata and the state of secrets,

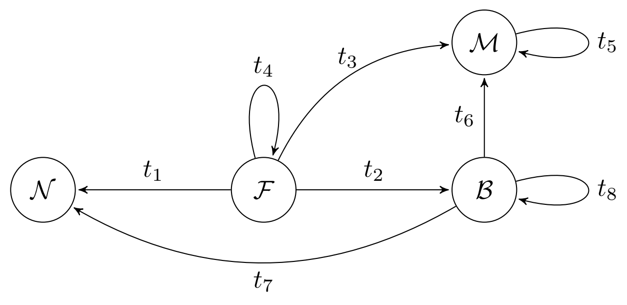

Fig. 1. General shape of the automaton A_n of node $n \in N$.

which is subject to evolve during the execution. More formally, a configuration c is a pair $(\vec{q}, \vec{s})$, where $\vec{q} \in L^{|N|}$ is the vector of states of the automata and $\vec{s} \in \mathbb{B}^{|S|}$ the current Boolean value of secrets. For example, $([\mathcal{F}, \mathcal{F}, \mathcal{F}, \mathcal{M}], [0, 0, 1])$ is a configuration of a model with four nodes (software components) and three secrets; first three nodes are functional ($\mathcal{F}$) and the last one is corrupted ($\mathcal{M}$), the third secret key is stolen (globally known).

The firing of a transition from a configuration to another corresponds to a transition in some automaton A_n in the network; it computes the abstract cost for the attacker to reach the target configuration. A transition resulting in a status change of node n in automaton A_n has a unitary cost and depends on the status of n itself and on the status of its inputs (predecessors) $p \in \texttt{pre}(n)$ in the visibility graph. The impact in terms of cost of a predecessor p on n may vary depending on the role r of n to which p is connected; it depends on the category and type of r, and on the attacker position labelling the edge from p to r in the visibility graph. Some transitions in A_n may need a set of predecessors (a coalition) to have particular statuses in order to be enabled. The theft of secrets impacts the cost calculation of the transition by eliminating the costs of breaking the communication protocols' security of the interactions that they protect. Stolen secrets are visible globally; this corresponds to an omniscient attacker that has a global knowledge of the system (or at least the parts that they control). The overall cost of a transition in A_n is the sum of the cost of breaking communication protocols and breaking node's role protections, or equals to the cost of stealing secrets by the node itself when its status is already $\mathcal{M}$ (malware).

Given a configuration of the propagation model, several transitions may potentially reach a successor configuration; however, we assume in SCORE that only the transition of minimal cost between a pair of configurations is considered in the definition of the system dynamics. This choice greatly improves performance while being consistent with SCORE's objective of discovering attacks with maximum likelihood, *i.e.,* minimum cost. Formally, this gives the following definition of the semantics:

Definition 3.1: The *semantics* of P with secrets S is a transition system $(\texttt{Config}_P, \rightarrow, c_0)$ where $\texttt{Config}_P$ is the set of all configurations reachable from an initial configuration $c_0 = (\vec{q_0}, \vec{s_0})$ by executing transitions defined as follows: A transition from $(\vec{q}, \vec{s})$ to $(\vec{q'}, \vec{s'})$ with cost κ, denoted by $(\vec{q}, \vec{s}) \xrightarrow{\kappa} (\vec{q'}, \vec{s'})$, exists if there exists an enabled transition $t_n = (\eta, \eta')$ in some automaton A_n with minimal cost κ, *i.e.,* $\vec{q}[n] = \eta$, $\vec{q'}[n] = \eta'$, and for all $i \in [1..|N|]$, $i \neq n$, $\vec{q'}[i] = \vec{q}[i]$, which updates accordingly the secrets, *i.e.,* $\vec{s'} = \texttt{update}_n(t_n, \vec{s})$. ◊

Attack discovery in the SCORE propagation model identifies sequences of interface crossings that lead from the initial configuration, with one or more attackers positioned in software components, to an undesirable target configuration. Among the huge number of possible attacks, SCORE selects, using model checking queries, that having a minimal cost or a cost under some fixed bound, representing the maximum likelihood needed to maintain the acceptable level of risk.

IV. Extension with resilience

Resilience is the ability of a system to operate under adverse conditions or stress, even if in a degraded mode, while maintaining essential operational capabilities, and to recover to a nominal operational mode. In the initial version of SCORE [3] only the mechanisms of access control, isolation and redundancy were proposed, which is not sufficient to cover most of the resilience requirements. In particular we were not able to model degraded mode nor recovery. As in distributed information systems the recovery process is complex and often includes human decision, in this paper we decided to focus on rising degraded modes. This was also needed by the application to the autonomous vehicle we had in mind.

In order to take into account a part of resilience requirements we introduce a *monitoring* concept, which considers for each target configuration $c \in \texttt{Config}_P$ a possible fallback one. More formally, we define a function $\texttt{fallback}\colon \texttt{Config}_P \to \texttt{Config}_P$ indicating for each configuration a corresponding fallback one, and an associated function $\mu\colon \texttt{Config}_P \to \mathbb{N}$ representing the cost of bypassing the monitoring in the target configuration.

Intuitively, the semantics of a propagation model P with resilience policy $P' = (P, \texttt{fallback}, \mu)$ is then obtained by replacing each transition $c_1 \xrightarrow{\kappa} c_2$ existing in the semantics of P, by two transitions: one with cost κ leading from c_1 to a fallback configuration $\texttt{fallback}(c_2)$ and another with cost $\kappa + \mu(c_2)$ from c_1 to the initial target configuration c_2. This means that from c_1 with cost κ we may only reach the fallback configuration of c_2, while c_2 remains reachable up to an additional cost $\mu(c_2)$.

Definition 4.1: The semantics of a propagation model with resilience policy $P' = (P, \texttt{fallback}, \mu)$ is a transition system $(\texttt{Config}_P, \hookrightarrow, c_0)$, where $\texttt{Config}_P$ is the set of all configurations reachable from c_0 by executing transitions in $\hookrightarrow$, defined as follows: if $c_1 \xrightarrow{\kappa} c_2$ with some cost κ, then we have $c_1 \xhookrightarrow{\kappa+\mu(c_2)} c_2$ and $c_1 \overset{\kappa}{\hookrightarrow} \texttt{fallback}(c_2)$. ◊

Note that the set of all configurations of P' and P are identical, $\texttt{Config}_{P'} = \texttt{Config}_P$. However, the accessibility of certain configurations may be modified.

V. Case study

In this section, we introduce our case study representing the on-board electronics of a Level 2 autonomous connected vehicle, which means that the vehicle can control its speed and direction in some specific situations but the human driver must be able to regain full control of the vehicle at any time.

A. Presentation

As represented in Fig. 2 and 3, the use case architecture is composed of two main parts: the Vehicle part comprising

- the three vehicle control domains:
  - Power Train, which covers engine and gearbox control (if applicable);
  - Body, which covers vehicle access control, cabin lighting, headlights, windscreen wipers and air conditioning;
  - Chassis, which includes brakes, steering, ultrasound and cluster.
- ADAS (Advanced Driver Assistance Systems), which includes front camera, lidar, front and rear radars, and assist mode switch;
- Communication;
- Multimedia, and
- Central Gateway, separating the critical parts of the vehicle from the Internet connectivity and from the multimedia, and allowing the navigation interacting with the external world. The central gateway switch gateSwitch filters the network connections and the central gateway unit gateUnit contains software components in charge of central diagnostics and software updates and the proxying activities between critical and exposed parts of the vehicle,

and a simplified representation of the External Infrastructure comprising

- Internet with a content provider and cellular network,
- Dealership with the capability of vehicle diagnostics and vehicle software update,
- OEM (car maker) with the central management of vehicle software and navigation map delivery,
- GNSS (Global Navigation Satellite System) responsible of the vehicle geographical positioning.

Both, the Vehicle and the External Infrastructure, have their hardware and software architectures, as shown in Fig. 2 and 3, respectively.

Concerning the hardware architecture of the vehicle, each of the three vehicle domains: Power Train, Body and Chassis have its own domain controller, which communicates with the components inside the domain using a separate CAN network. The ADAS domain is built around an Ethernet switch adasSwitch allowing the exchange of a large volume of data between the ADAS controller adasCtrl and all the domain components. All these four domains mentioned above communicate through inter-domain CAN network interd. Three complementary Ethernet links between ADAS, Power Train and Chassis domains allow to exchange of a large data necessary for advanced functionalities (ADAS); for example, displaying the image from the rear camera rCamera on the cluster cluster. The central gateway area Central Gateway consists of two components: a switch gateSwitch in charge of network traffic filtering and redirection, and a gate unit gateUnit in charge of central management functions and proxying activities necessary for supplementary information flow verification and separation. The switch gateSwitch is connected by Ethernet to gateUnit, adasSwitch, multimedia/navigation mMedia, the communication unit commUnit, and to the dealer diagnostic devices dealer during the dealer intervention. The communication unit commUnit connects the vehicle to the Internet through the cellular network. The gate unit gateUnit has also a supplementary link to the inter-domain CAN network interd. The body controller bodyCtrl communicates via radio link with the vehicle's access card card, and the multimedia and navigation unit mMedia communicates via Bluetooth with a smartphone phone, for example to stream music to the vehicle. Of course, the smartphone is connected to the Internet.

The above architecture may seem suboptimal, but it reflects the current real-world situation of automakers who prefer to reuse existing solutions to extend the functionality of their products. These incremental transformations require a very detailed analysis of the cyber risks generated by the newly introduced interactions between initially independent subsystems.

Concerning the software architecture, we distinguish the following categories of components:

- kernels, shown with a dashed border,
- components with root privileges, shown in light pink with a brown border,
- user space components, shown in light blue with a dark border.

Each software component has a name starting with a capital letter and its hosting hardware component indicated on a dark background in the lower part. The interactions are indicated using links and sometimes using pairs of labels having the same colour, especially when it would be too complicate to trace lines, but there is no semantic difference between these two representations. A hardware component may host at most one kernel. Kernels have no depicted explicit interactions but implicitly, each kernel manages system interactions with all software components hosted by the same hardware component. Kernel-less hardware components can only host software components having root privileges.

B. Attacker entry points

Potential attacker entry points to the system are also indicated in the architecture definition. They appear in orange in Fig. 2 with a Trojan icon. We consider the following entry points for attackers:

- intHacker is located on the Internet;
- dealHacker is located in the dealership;
- oemHacker is located in the intranet of the carmaker (OEM);

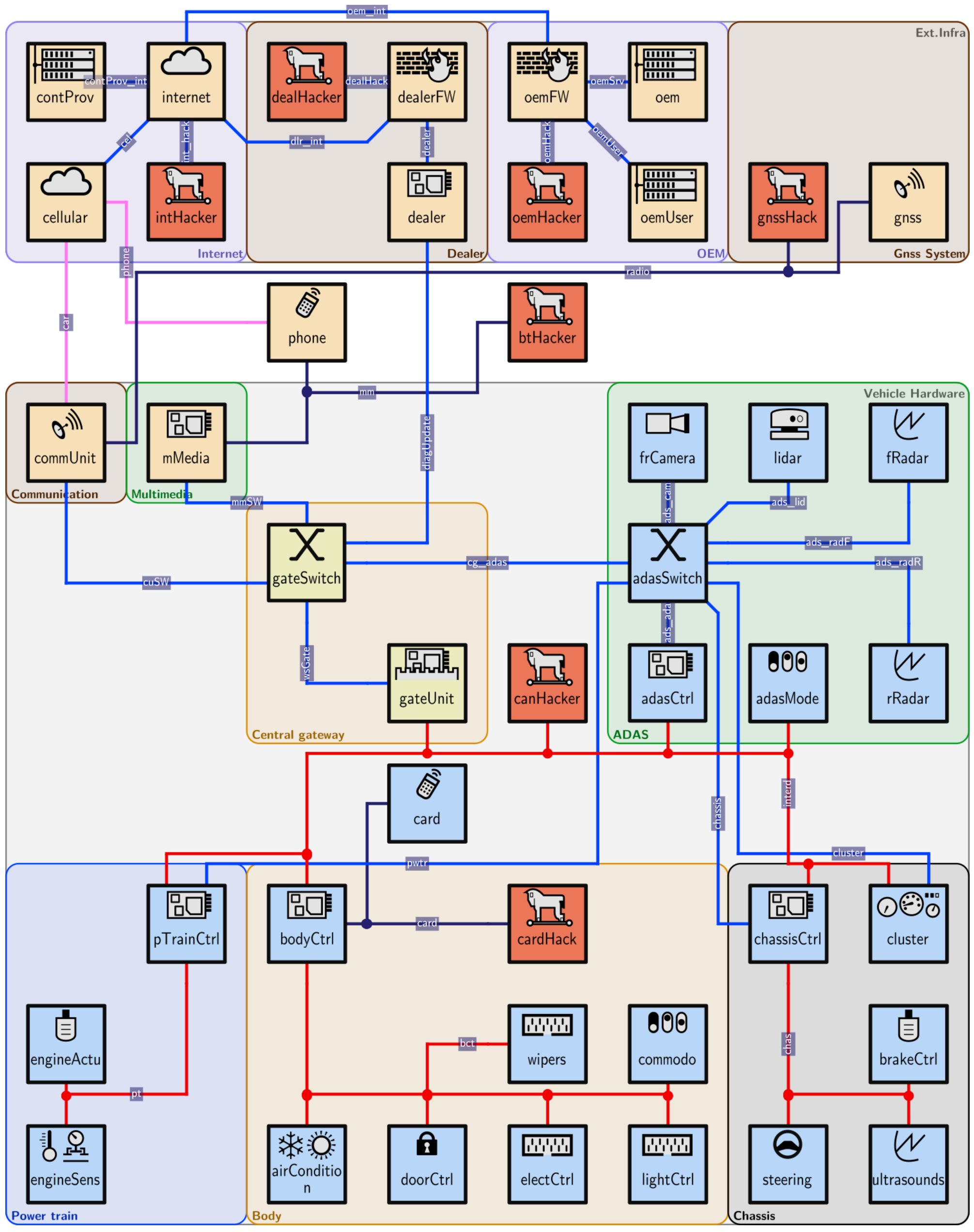

Fig. 2. Hardware architecture.

- gnssHack can attack the GNSS radio transmission;
- btHacker can attack the Bluetooth connectivity of the vehicle;
- canHacker can attack through the physical access to the vehicle CAN network;
- cardHacker can attack via the radio link used by the vehicle's access card.

Each of the above hackers has a corresponding software component in Fig. 3 which is a kernel surrounded by a red box.

C. *Model of propagation*

Our tool implementing the SCORE method generates the propagation model for UPPAAL model checker [29] from the above description of architectures and a pattern encoding the automata with the related unitary costs. In our case the obtained output is composed of a large data structure and a set of 59 automata of the form of that in Fig. 1, one for each software component (including hackers), each of them having four states (one for each possible status of a component in $\{\mathcal{F}, \mathcal{M}, \mathcal{B}, \mathcal{N}\}$). For each of the following properties we need to specify the initial and target configuration. The initial one

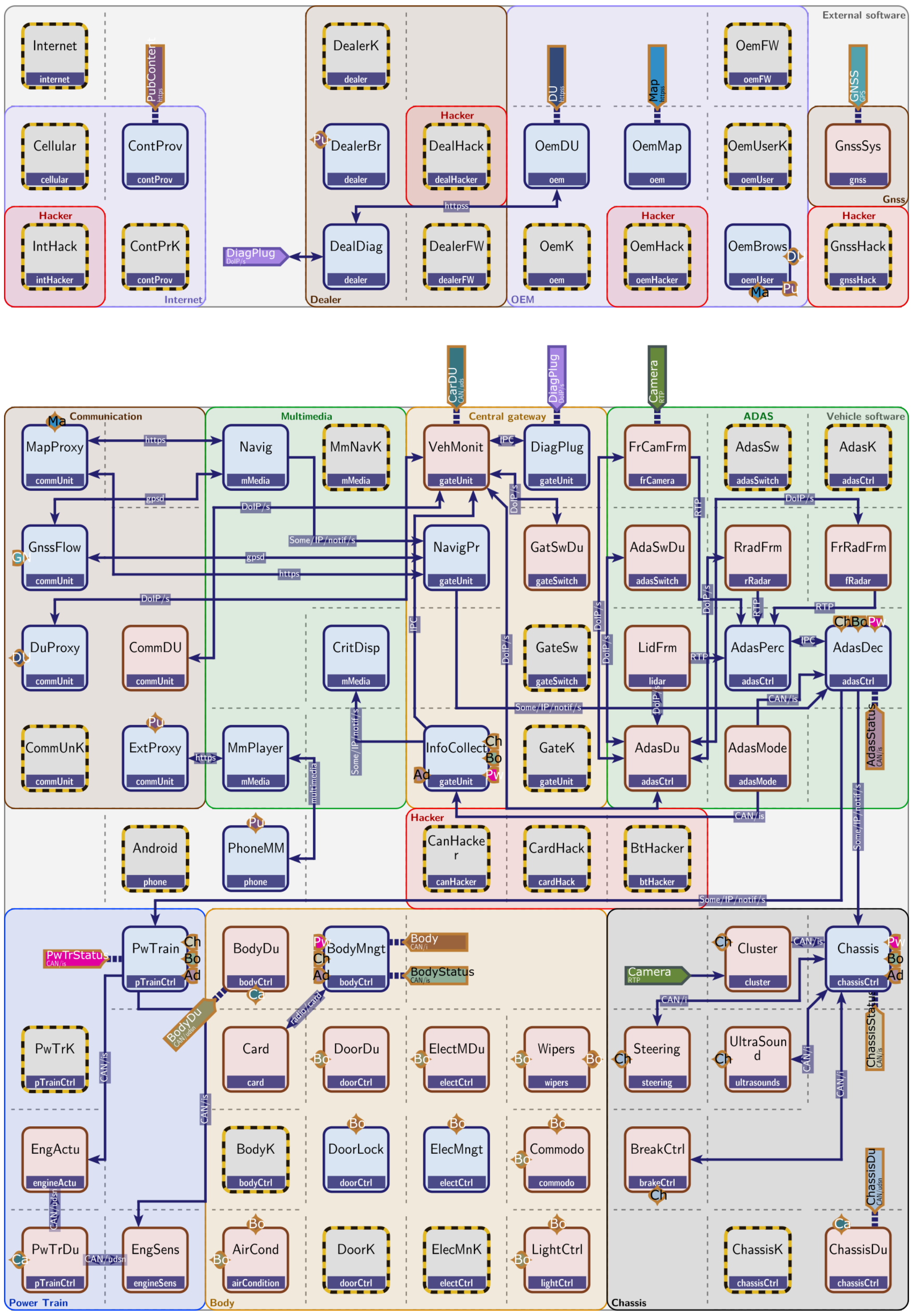

Fig. 3. Software architecture.

consists in defining at least one active hacker, *i.e.,* such that its status is different from functional $\mathcal{F}$; usually we set it to malware $\mathcal{M}$. The target configuration consists in indicating which components have to be compromised, *i.e.,* the status of one or more of them should be different from $\mathcal{F}$.

D. Analysis

Our study is a part of the risk analysis process such as defined in standards like ISO 21434 [30] or EBIOS Risk Manager [31]. It starts from the definition of so called *feared events* associated here with vehicle functionalities and reveals a harmful breach of them. Each feared event is given a severity degree and an accepted level of risk. Both together allow to estimate for each feared event the maximal accepted likelihood of attacks, which can trigger this event.

In this paper, we assume that the above stage is already provided and the list of feared events is known along with the related likelihood, represented by its inverse, *i.e.,* the acceptable cost, as shown in Tab. II. We also assume that the unitary costs are already defined for all protocols and all the types of software components used in our architectures, taken from libraries provided by experts. Intuitively, the costs may be divided in three categories: weak (under 16), intermediate (between 15 and 25) and strong (between 26 and 50). Our contribution consists then in looking for potentially dangerous attacks capable to trigger the mentioned feared events, computing their costs in order to either confirm that the level of risk remains acceptable, or to propose modifications leading to better protections. The analysis we present in the following is developed in order to illustrate the method. It is of course partial and provides only a small subset of attacks, which should be included in a complete industrial cybersecurity risk assessment.

TABLE II
FEARED EVENTS AND ACCEPTABLE COSTS VS COMPANY RISK MANAGEMENT POLICY

	feared event	acceptable cost
1	vehicle theft	80
2	ADAS sabotage	80
3	driver disturbance	40

Our analysis will proceed as follows: For each feared event `feared`, we will look for possibly several attacks depending on the initial and target configurations. More precisely, we choose a set of meaningful attacker entry points and a set of target software components to be compromised in order to trigger `feared`. This leads to several cases represented in Tab. III and IV together with their calculated minimal costs and the corresponding shortest path.

In order to compute these attack paths of minimal costs we set for each case the initial configuration in the propagation model expressed in UPPAAL and use the model checker. It is worth to mention that the UPPAAL formula allowing to find the minimal cost of an attack of a given target may be prohibitively long to compute due to the size of the model and the combinatorial explosion of the number of reachable configurations. Actually, using the "brute force" is not efficient (we stopped the computation after a few hours) and we need to use hints to make this computation feasible. The method consists in first assuming that such an attack exists and use the model checker to confirm that it is under a given estimated cost. The difference is that such a restricted request is generally much faster. It's even faster as this estimated cost gets closer to the searched minimum cost. Usually, we start from largest estimations and refine them by dichotomy until obtaining an acceptable computation time, usually under 30m. Then, still under the constraint of the estimated cost, we search the attack of the minimum cost, which take usually a few minutes.

For example, in the case 1a in Tab. III we set first the status of IntHacker to $\mathcal{M}$ and that of all the other components to $\mathcal{F}$ in the initial configuration. The UPPAAL formula

$$\texttt{inf\{DoorLock.B\}: costs} \tag{1}$$

looks for the minimal value of variable *costs* of an attack path reaching the target configuration where the component DoorLock has status $\mathcal{B}$. However, as mentioned above, it is not efficient without constraining the cost under some estimated bound. To find such a bound, we check under different cost constraints the UPPAAL CTL formula

$$\texttt{E<>(DoorLock.B)} \tag{2}$$

saying that there is a path reaching eventually the configuration where the component DoorLock has status $\mathcal{B}$. If the cost constraint is close to the minimum, the formula (1) becomes efficient and it allows to find the attack of the minimal cost quite fast (in few minutes, under 10m). The minimal calculated cost is 100 and the obtained path is shown in Tab. III together with the cost of each step.

We consider six cases for the vehicle theft depicted in Tab. III. In the first three, the attacker comes from the Internet, and looks for compromising the functionality of the access to the vehicle and its start, which are managed by the software component BodyMngt. In all these cases the attack path passes through the update functionality. The second case shows that if BodyMngt is compromised, the opening of the doors becomes almost costless. The third case shows a malware installation on BodyMngt, which results in a persistent access to the vehicle, but at the price of a higher cost. In the fourth case the objective is as in the previous ones but when the vehicle is connected to the dealer diagnostic tool via DiagPlug so that the attack passes through the dealership. In the fifth case, the attacker enters through Bluetooth interface. This attack is costly because in our setting Bluetooth does not allow any vehicle opening functionalities. In the sixth case, the attacker is initially located in the dealership, attacks first the diagnostic device, then the body management in order to install a malware to be used when the vehicle is outside the dealership.

Concerning ADAS sabotage, see Tab. IV, we consider two cases with the same attacker entry point from the Internet. In the first case the target is the break control BreakCtrl and the second the ADAS decision function AdasDec.

TABLE III
ATTACKS, COSTS AND PATHS FOR FEARED EVENT 1.

	entry/status	target/status	cost	path (comp/status:cost)
1a	IntHacker/M	DoorLock/B	100	IntHacker/M → DuProxy/B:50 → VehMonit/B:20 → BodyDu/B:5 → BodyMngt/B:20 → DoorLock/B:5
1b	IntHacker/M	BodyMngt/B	95	IntHacker/M → DuProxy/B:50 → VehMonit/B:20 → BodyDu/B:5 → BodyMngt/B:20
1c	IntHacker/M	BodyMngt/M	105	IntHacker/M → DuProxy/B:50 → VehMonit/B:20 → BodyDu/B:5 → BodyMngt/M:30
1d	IntHacker/M	BodyMngt/M	105	IntHacker/M → DealDiag/B:55 → DiagPlug/B:5 → VehMonit/B:20 → BodyDu/B:5 → BodyMngt/M:30
1e	BtHacker/M	BodyMngt/B	120	BtHacker/M → MmPlayer/M:25 → Navig/B:10 → NavigPr/B:20 → AdasDec/B:15 → BodyMngt/M:50
1f	DealHack/M	BodyMngt/M	90	DealHack/M → DealDiag/B:30 → DiagPlug/B:5 → VehMonit/B:20 → BodyDu/B:5 → BodyMngt/B:30

TABLE IV
ATTACKS, COSTS AND PATHS FOR FEARED EVENTS 2 AND 3.

	entry/status	target/status	cost	path (comp/status:cost)
2a	IntHacker/M	BreakCtrl/B	80	IntHacker/M → DuProxy/B:50 → VehMonit/B:20 → ChassisDu/B:5 → BreakCtrl/B:5
2b	IntHacker/M	AdasDec/B	85	IntHacker/M → MapProxy/B:50 → NavigPr/B:20 → AdasDec/B:15
3a	IntHacker/M	MmPlayer/B	45	IntHacker/M → PhoneMM/B:40 → MmPlayer/B:5
3b	IntHacker/M	MmPlayer/B	45	IntHacker/M → ExtProxy/M:40 → MmPlayer/B:5

For the possibility of driver disturbance, see also Tab. IV, we consider two attacks starting from the Internet and targeting the multimedia player MmPlayer, for example to force the maximal speaker volume. The first attacks through the phone connected to the vehicle, while the second through the HTTP proxy; both have the same cost.

The calculated minimal costs of all the above attacks are acceptable from the risk assessment point of view, however one may observe a high dependability on the resistance of the proxy functionalities. As a consequence, a further reinforcing could be useful. A possible solution could be provided by a monitoring with a reaction such as a fallback. This will be illustrated in the next section.

E. Monitoring and fallback

In this section we provide the system with monitoring and fallback of three proxies: ExtProxy, DuProxy and MapProxy. Each of them may of course be lured but at the price of a supplementary cost.

We assume that an attempt to attack ExtProxy may lead to switch it off (*i.e.,* force its status to non-available $\mathcal{N}$) meaning that the vehicle looses the access to the Internet multimedia content, we call this fallback action Fb_1. The remaining two proxies are more critical, so we assume that the fallback action Fb_2 cuts off the whole communication between the vehicle and the external world, *i.e.,* the whole Central gateway part is forced to switch off. In the propagation model this is represented by forcing GateK and GateSw to status $\mathcal{N}$.

TABLE V
ATTACKS, COSTS AND PATHS WITH MONITORING AND FALLBACK FOR FEARED EVENT 1.

	entry/status	target/status	cost	path (comp/status:cost)
1a	IntHacker/M	DoorLock/B	130	IntHacker/M → DuProxy/B:80 → VehMonit/B:20 → BodyDu/B:5 → BodyMngt/B:20 → DoorLock/B:5
1b	IntHacker/M	BodyMngt/B	125	IntHacker/M → DuProxy/B:80 → VehMonit/B:20 → BodyDu/B:5 → BodyMngt/B:20
1c	IntHacker/M	BodyMngt/M	135	IntHacker/M → DuProxy/B:80 → VehMonit/B:20 → BodyDu/B:5 → BodyMngt/M:30
1d	IntHacker/M	BodyMngt/M	105	IntHacker/M → DealDiag/B:55 → DiagPlug/B:5 → VehMonit/B:20 → BodyDu/B:5 → BodyMngt/M:30
1e	BtHacker/M	BodyMngt/B	120	BtHacker/M → MmPlayer/M:25 → Navig/B:10 → NavigPr/B:20 → AdasDec/B:15 → BodyMngt/M:50
1f	DealHack/M	BodyMngt/M	90	DealHack/M → DealDiag/B:30 → DiagPlug/B:5 → VehMonit/B:20 → BodyDu/B:5 → BodyMngt/B:30

In Tab. V and VI we show the same combinations of entry points and targets for the three mentioned above feared events. As expected, the costs of attacks which are able to avoid proxies are unchanged. All the attack paths passing through the proxies are more expensive. However, new paths having lower costs appear, like those of cases 2a' and 2b', which pass through the dealership.

In order to find the attacks luring the monitoring and calculate their costs, we need to assume that either the vehicle is not plugged on the dealer diagnostic tool (cases 1a, 1b, 1c, 1e, 2a and 2b), or that it is not connected to the phone (case 3b). It may be obtained using the UPPAAL formula, for example `inf{MmPlayer.B and Android.F and PhoneMM.F}: costs` for the case 3b.

TABLE VI
ATTACKS, COSTS AND PATHS WITH MONITORING AND FALLBACK FOR FEARED EVENTS 2 AND 3.

	entry/status	target/status	cost	path (comp/status:cost)
2a	IntHacker/M	BreakCtrl/B	110	IntHacker/M → DuProxy/B:80 → VehMonit/B:20 → ChassisDu/B:5 → BreakCtrl/B:5
2a'	IntHacker/M	BreakCtrl/M	105	IntHacker/M → DealDiag/B:55 → DiagPlug/B:5 → VehMonit/B:20 → ChassisDu/B:5 → BreakCtrl/M:15
2b	IntHacker/M	AdasDec/B	115	IntHacker/M → MapProxy/B:80 → NavigPr/B:20 → AdasDec/B:15
2b'	IntHacker/M	AdasDec/B	105	IntHacker/M → DealDiag/B:55 → DiagPlug/B:5 → VehMonit/B:20 → AdasDu/B:5 → AdasDec/B:15
3a	IntHacker/M	MmPlayer/B	45	IntHacker/M → PhoneMM/B:40 → MmPlayer/B:5
3b	IntHacker/M	MmPlayer/B	95	IntHacker/M → ExtProxy/M:70 → MmPlayer/B:5

As shown above, adding monitoring and fallback improves system protection. However, because this introduces a new cyber attack surface, triggering fallback mode may constitute a new target for the attacker. This is visible in our case study as an attack from the Internet taking as the objective to rise fallback Fb_2 is possible for a rather low cost. The UPPAAL formula

```
inf{(GateK.N and GateSw.N)}: costs
```

returns the minimum cost of 50.

F. Redundancy

In this section we illustrate a situation where the attacker is able to pass through the routing restrictions on the switches of Central Gateway and ADAS, *i.e.,*GateSw and AdasSw, for example if it succeeded to get the network access key. This allows it to attack communication protocols between perception (AdasPerc) and various sensors such as lidar, radar or camera, in order to force perception to create a fake scene, *i.e.,* a false image of the road situation. Usually, these protocols are weakly protected.

As these sensors are redundant in the sense that they provide partly the same information, in our case the attacker has to compromise at least two of them, which makes it more costly.

Concerning the network access key, we consider that the attacker steals it from the OEM Diagnostic and Update Center (OemDU).

We are able to detect such an attack by placing the attacker in the Internet, *i.e.,* in IntHacker and by looking for paths compromising ADAS perception with the following formula:

```
inf{(AdasPerc.B and DiagPlug.F)}: costs
```

The condition DiagPlug.F is added to confirm that the vehicle is not connected to the dealer diagnostic tool. The corresponding path and cost is given in Tab. VII.

TABLE VII
ATTACK, COST AND PATH FOR SENSOR COMPROMISE.

entry/status	target/status	cost	path (comp/status:cost)
IntHacker/M	AdasPerc/B	110	IntHacker/M → OemDU/key-theft:35 → ExtProxy/M:55 → AdasPerc/B:20

The first step in the path consists in stealing remotely the protection key without changing the status of the OEM Diagnostic and Update Center. Then, the attacker installs malware on the External Proxy luring its monitoring but taking profit from the stolen access key. Finally, the attacker compromises synchronously two interactions between the Perception and two of its sensors to produce fake data. This is costly but less expensive than the installation of a malware on the Perception.

VI. CONCLUSION AND PERSPECTIVES

We show how this method can be used for cybersecurity risk assessment of large critical systems. It can also suggest directions for optimising both passive architectural and active cyber protections in order to achieve an acceptable level of risk. We expect that such mechanisms will eventually become compulsory for resilient critical systems, and in particular for autonomous vehicles.

We notice, that certain threat scenarios depend on the functionality of software components which is unrelated to security and is thus not covered by our security–focused approach. For example, in the case of a vehicle, it may be interesting to know whether the vehicle is stationary or running, whether the navigation system is connected to the Internet or not, or whether only certain ADAS functionalities, such as the autopilot, is active or not. These extensions are of course possible, but may degrade computational efficiency. The challenge for our future work will be to find an acceptable trade-off between these aspects.

We see two improvements related to unitary attack costs: their automatic synchronisation with the CVE database [32] and and an enhanced method of their cumulation into the total scenario expense. The former would improve on resilience thanks to timely dissemination of vulnerability information. The latter would take into accounts elements such as partial observability of the system from the perspectives of different actors.

Acknowlegment

This work was supported by the French government as part of the "France 2030" program, within the framework of the SystemX Technological Research Institute.

References

[1] A. Clark and S. Zonouz, "Cyber-physical resilience: Definition and assessment metric," *IEEE Transactions on Smart Grid*, vol. 10, no. 2, pp. 1671–1684, 2017. doi: 10.1109/TSG.2017.2776279

[2] N. Leveson, N. Dulac, D. Zipkin, J. Cutcher-Gershenfeld, J. Carroll, and B. Barrett, "Engineering resilience into safety-critical systems," in *Resilience engineering*. CRC Press, 2017. doi: 10.1201/9781315605685-12 pp. 95–123.

[3] G. Hutzler, H. Klaudel, W. Klaudel, F. Pommereau, and A. Rataj, "Automatic discovery of cyberattacks," in *IEEE CSR*, 2024, to appear.

[4] S. Quinn, N. Ivy, M. Barrett, L. Feldman, G. Witte, and R. Gardner, "Identifying and estimating cybersecurity risk for enterprise risk management," 2021. doi: 10.6028/NIST.IR.8286A https://nvlpubs.nist.gov/nistpubs/ir/2021/NIST.IR.8286A.pdf.

[5] "Digital risk management," French Cybersecurity Agency, 2024, https://cyber.gouv.fr/en/digital-risk-management.

[6] S. Gupta Bhol, J. Mohanty, and P. Kumar Pattnaik, "Taxonomy of cyber security metrics to measure strength of cyber security," *Materials Today: Proceedings*, vol. 80, pp. 2274–2279, 2023. doi: 10.1016/j.matpr.2021.06.228 SI:5 NANO 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214785321046009

[7] S. Mauw and M. Oostdijk, "Foundations of attack trees," in *Information Security and Cryptology-ICISC 2005*. Springer, 2006. doi: 10.1007/11734 pp. 186–198.

[8] J. Arias, C. E. Budde, W. Penczek, L. Petrucci, T. Sidoruk, and M. Stoelinga, "Hackers vs. security: attack-defence trees as asynchronous multi-agent systems," in *International Conference on Formal Engineering Methods*. Springer, 2020. doi: 10.1007/978-3-030-63406-3_1 pp. 3–19.

[9] R. Ritchey and P. Ammann, "Using model checking to analyze network vulnerabilities," in *IEEE Symposium on Security and Privacy*, 2000. doi: 10.1109/SECPRI.2000.848453 pp. 156–165.

[10] S. Jajodia, S. Noel, and B. O'berry, "Topological analysis of network attack vulnerability," *Managing Cyber Threats: Issues, Approaches, and Challenges*, pp. 247–266, 2005. doi: 10.1145/1229285.1229288

[11] M. Ge, J. B. Hong, W. Guttmann, and D. S. Kim, "A framework for automating security analysis of the internet of things," *Journal of Network and Computer Applications*, vol. 83, pp. 12–27, 2017. doi: 10.1016/j.jnca.2017.01.033

[12] C. Hankin, P. Malacaria *et al.*, "Attack dynamics: an automatic attack graph generation framework based on system topology, capec, cwe, and cve databases," *Computers & Security*, vol. 123, p. 102938, 2022. doi: 10.1016/j.cose.2022.102938

[13] O. Sheyner and J. Wing, "Tools for generating and analyzing attack graphs," in *International symposium on formal methods for components and objects*. Springer, 2003. doi: 10.1007/978-3-540-30101-1_17 pp. 344–371.

[14] K. Piwowarski, K. Ingols, and R. Lippmann, "Practical attack graph generation for network defense," in *Computer Security Applications Conference*. IEEE Computer Society, 2006. doi: 10.1109/ACSAC.2006.39. ISSN 1063-9527 pp. 121–130. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ACSAC.2006.39

[15] B. Schneier, "Attack trees," *Dr. Dobb's journal*, vol. 24, no. 12, pp. 21–29, 1999.

[16] B. Kordy, S. Mauw, S. Radomirović, and P. Schweitzer, "Attack–defense trees," *Journal of Logic and Computation*, vol. 24, no. 1, pp. 55–87, 06 2012. doi: 10.1093/logcom/exs029. [Online]. Available: https://doi.org/10.1093/logcom/exs029

[17] D. M. Kienzle and W. A. Wulf, "A practical approach to security assessment," in *Proceedings of the 1997 workshop on New security paradigms*, 1998. doi: 10.1145/283699.283731, pp. 5–16.

[18] M. S. Barik, A. Sengupta, and C. Mazumdar, "Attack graph generation and analysis techniques," *Defence Science Journal*, vol. 66, no. 6, p. 559, 2016. doi: 10.14429/dsj.66.10795

[19] H. S. Lallie, K. Debattista, and J. Bal, "A review of attack graph and attack tree visual syntax in cyber security," *Computer Science Review*, vol. 35, p. 100219, 2020. doi: 10.1016/j.cosrev.2019.100219. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574013719300772

[20] K. Kaynar, "A taxonomy for attack graph generation and usage in network security," *Journal of Information Security and Applications*, vol. 29, pp. 27–56, 2016. doi: 10.1016/j.jisa.2016.02.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214212616300011

[21] MITRE, "Common weakness enumeration," 2023, https://cwe.mitre.org/data/index.html.

[22] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. Wing, "Automated generation and analysis of attack graphs," in *IEEE Symposium on Security and Privacy*, 2002. doi: 10.1109/SECPRI.2002.1004377, pp. 273–284.

[23] I. Chokshi, N. Ghosh, and S. K. Ghosh, "Efficient generation of exploit dependency graph by customized attack modeling technique," in *Advanced Computing and Communications*. IEEE Computer Society, 2012. doi: 10.1109/ADCOM.2012.6563582, pp. 39–45. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ADCOM.2012.6563582

[24] Z. B. Celik, P. McDaniel, and G. Tan, "Soteria: Automated {IoT} safety and security analysis," in *USENIX Annual Technical Conference*, 2018. doi: 10.48550/arXiv.1805.08876, pp. 147–158.

[25] J. Hong and D.-S. Kim, "Harms: Hierarchical attack representation models for network security analysis," 2012. doi: 10.4225/75/57b559a3cd8da

[26] J. B. Hong and D. S. Kim, "Towards scalable security analysis using multi-layered security models," *Journal of Network and Computer Applications*, vol. 75, pp. 156–168, 2016. doi: 10.1016/j.jnca.2016.08.024,

[27] P. Ammann, D. Wijesekera, and S. Kaushik, "Scalable, graph-based network vulnerability analysis," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, 2002. doi: 10.1145/586110.586140, pp. 217–224.

[28] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *Computers, IEEE Transactions on*, vol. 100, no. 8, pp. 677–691, 1986. doi: 10.1109/TC.1986.1676819

[29] G. Behrmann, A. David, and K. G. Larsen, "A tutorial on UPPAAL," in *LNCS*, vol. 3185. Springer, 2004. doi: 10.1007/978-3-540-30080-9_7, pp. 200–236.

[30] "Road vehicles, Cybersecurity engineering," International Organization for Standardization, Geneva, CH, Standard, 2021.

[31] "Ebios risk manager," French Cybersecurity Agency, 2024, https://www.ssi.gouv.fr/uploads/2019/11/anssi-guide-ebios_risk_manager-en-v1.0.pdf.

[32] "Common vulnerabilities and exposures," MITRE, 2024. [Online]. Available: http://cve.mitre.org