Security-Minded Verification of Cooperative Awareness Messages

Marie Farrell[§], Matthew Bradbury[†], Rafael C. Cardoso[‡], Michael Fisher[§], Louise A. Dennis[§], Clare Dixon[§], Al Tariq Sheik[¶], Hu Yuan[‖], Carsten Maple[¶]

[†] School of Computing and Communications, Lancaster University, Lancaster, UK
[‡] Department of Computing Science, University of Aberdeen, Aberdeen, UK
[§] Department of Computer Science, University of Manchester, Manchester, UK
[¶] WMG, University of Warwick, Coventry, UK
[‖] School of Computer Science and Mathematics, Kingston University, London, UK
Email: marie.farrell@manchester.ac.uk, m.s.bradbury@lancaster.ac.uk, rafael.cardoso@abdn.ac.uk, michael.fisher@manchester.ac.uk, louise.dennis@manchester.ac.uk, clare.dixon@manchester.ac.uk, t.sheik@warwick.ac.uk, h.yuan@kingston.ac.uk, cm@warwick.ac.uk

Abstract—Autonomous robotic systems systems are both safety- and security-critical, since a breach in system security may impact safety. In such critical systems, formal verification is used to model the system and verify that it obeys specific functional and safety properties. Independently, threat modelling is used to analyse and manage the cyber security threats that such systems may encounter. Both verification and threat analysis serve the purpose of ensuring that the system will be reliable, albeit from differing perspectives. In prior work, we argued that these analyses should be used to inform one another and, in this paper, we extend our previously defined methodology for security-minded verification by incorporating runtime verification. To illustrate our approach, we analyse an algorithm for sending Cooperative Awareness Messages between autonomous vehicles. Our analysis centres on identifying STRIDE security threats. We show how these can be formalised, and subsequently verified, using a combination of formal tools for static aspects, namely Promela/SPIN and Dafny, and generate runtime monitors for dynamic verification. Our approach allows us to focus our verification effort on those security properties that are particularly important and to consider safety and security in tandem, both statically and at runtime.

Index Terms—Verification, Security, Safety, Threat Modelling, Connected Autonomous Vehicles, Cooperative Awareness Messages.

1 Introduction

Complex autonomous systems are often both safety- and security-critical, meaning that they must be analysed by experts in both areas to ensure that they behave correctly in the presence of malicious adversaries. These analyses are typically carried out by disparate teams and if one analysis prompts changes to the system then it must be reconsidered by both teams. This is a time consuming process, and since both kinds of analysis share some commonalities, there is clear benefit in holistically combining them.

Safety for these systems is typically verified using a range and combination of non-formal (e.g., testing) and formal (e.g., model-checking) methods. While a recent survey on formal verification techniques for autonomous robotic systems identified multiple challenges for applying formal methods to these systems [3], cyber security as a distinct challenge for formal methods has often been overlooked. Specifically, identifying which cyber security properties to verify can be difficult for formal methods practitioners [2].

In cyber security, threat analysis techniques identify the threats that are most impactful and likely for systems. This analysis essentially produces a set of events that can lead to a system being in a bad state or taking bad actions, from which properties that should hold in order for the system to be secure can be derived. These properties are typically written in abstract natural-language and are not formalised in a logic that is typically used for formal verification.

In this paper, we extend and strengthen our security-minded formal verification methodology [1] to incorporate runtime verification. To illustrate our approach, we apply this extended methodology to a previous case study of the Cooperative Awareness Message (CAM) protocol, used in vehicle-to-vehicle communications [4]. We presented an initial approach to security-minded verification of CAM in [2] and this paper applies our more detailed methodology to this use case. As a result, we provide modified formal models and runtime monitors for the CAM protocol that were not present in our original work [2]. Our contributions are:

1) We improve the security-minded verification methodology that was presented in our previous work [1] by incorporating Runtime Verification (RV) (§3).
2) We provide a more detailed threat modelling than in [2] and compare with the threats identified by ETSI (§4).
3) We extend our prior case study [2] to demonstrate how runtime monitors can be generated and applied following our improved methodology (§5).
4) The verification artefacts presented in this paper were drawn from [2] but they have been modified based on the improved methodology and more detailed threat

This paper extends previously published work [1, 2].

modelling in this paper. We also discuss mitigations and alternative approaches that were not present in [2] (§6).

We begin by describing the relevant background material and related work (§2). In §3 we present our framework for combining formal verification with cyber security threat analysis. In §4, we provide a threat analysis of the CAM protocol using the STRIDE classification. §5 describes our results of verifying properties related to Spoofing and Denial of Service (DoS) attacks using model checking, theorem proving and runtime verification. In §6 we discuss potential mitigations against the identified threats. In §7 we reflect on the approach taken. §8 concludes and outlines future work.

2 Background and Related Work

This section outlines the relevant background on: (i) the threat analysis techniques that we have used, (ii) formal verification, (iii) Cooperative Awareness Messages [5], and (iv) combining security and formal verification.

2.1 Threat Analysis

When engineering security-critical systems, developers employ threat analysis techniques to identify security vulnerabilities so that targeted mitigations can be put in place. There are many techniques for threat modelling, and our approach works equally well with any of them (e.g., CIA which stands for Confidentiality, Integrity and Availability [6]), but for ease of explanation, we adopt STRIDE [7].

The elements of STRIDE are: (i) Spoofing — an attacker pretends to be another entity, (ii) Tampering — an attacker manipulates data, (iii) Repudiation — an attacker can deny sending a message that it sent, (iv) Information Disclosure — an attacker can cause the system to reveal information to those it is not intended for, (v) DoS — an attacker can prevent the system from functioning, and (vi) Elevation of Privilege — an attacker can perform more actions than allowed.

Analysing a system in terms of STRIDE threats helps developers to secure the system by identifying vulnerable areas so that mitigations can be introduced. The identified threats will have their impact and likelihood assessed to calculate the risk of each threat [8], allowing the prioritisation of developing mitigations for threats with a higher risk.

2.2 Formal Verification

In order to assure the correctness of a software system, formal methods provide an array of mathematically-based tools and techniques for verifying that a system behaves correctly. Formal methods are predominantly used in safety-critical systems where a software failure can potentially cause harm.

In this paper, we use two distinct formal methods for static verification; Promela/SPIN [9] and Dafny [10] to verify properties about the CAM protocol[1]. In each case, we model the CAM protocol at a different abstraction level; Promela for system-level modelling and Dafny for algorithm-level verification. Since these systems are typically very complex, the use of multiple formal methods is necessary [11], and cyber security threat analysis techniques highlight the most relevant security properties to focus the verification effort.

Promela is a general purpose programming language, particularly developed for protocol verification, where the patterns of temporal behaviour that can be verified can be complex and varied [12]. SPIN is a model-checker that automatically checks temporal properties over system models which are encoded in the Promela programming language [9, 12]. Essentially, SPIN explores all possible runs of Promela input models and assesses these against an automaton capturing temporal behaviour that should *never* occur. If all runs have been explored without finding a violation of the temporal properties then the model is valid. If a violation is found, it is returned as a counter-example.

Dafny is a programming language that facilitates the use of specification constructs that allow the user to specify pre-/post-conditions, loop invariants and variants [10]. Dafny is used in the static verification of the functional correctness of programs. Dafny programs are translated into the Boogie intermediate verification language [13] and then the Z3 automated theorem prover discharges the associated proof obligations [14]. We chose Dafny for this case study because of its similarity to other programming languages making it easy to communicate the verified solution to security engineers that are unfamiliar with formal methods.

In addition to traditional, static, formal verification we deploy runtime monitors to provide verification at runtime [15]. We use the LamaConv[2] tool to generate runtime monitors. Runtime Verification (RV) checks the traces of events that are produced by the system execution against formal properties. A runtime monitor reads a finite trace of events and yields a verdict (e.g., inconclusive, satisfaction, or violation) based on the formal property that it is checking. This serves two roles in our methodology and case study: (1) any formal model of a realistic system will be incomplete and so a runtime monitor is useful for checking outside of the modelled envelope; and (2) properties that are not amenable to static formal verification may be tackled through verification at runtime. As we will see later, an example of the second kind of property is a DoS attack, which can be detected at runtime by recognising an unexpectedly large number of messages.

2.3 Cooperative Awareness Messages (CAMs)

Emerging applications of autonomous robotic systems include Connected and Autonomous Vehicle (CAV) systems where self-driving vehicles communicate with each other to safely traverse different locations. This communication typically occurs over a wireless network that is vulnerable to attacks and these attacks could potentially impede the safety of the passengers. Ensuring that both cyber security and safety issues are properly addressed during the software development process is crucial for these CAV systems.

CAMs are heartbeat messages that are broadcast by vehicles in a CAV system to their neighbours providing basic vehicle status information including position, velocity, acceleration, and heading [4]. Since these vehicles communicate over an unsecured network, ensuring that CAMs are secure is crucial as we move toward driverless cars. We briefly summarise the CAM standard documentation [5] to give the

1. We used version 6.4.6 of SPIN, version 2.2.0 of Dafny and version 1.41 of Visual Studio Code on Ubuntu 18.04.

2. https://www.isp.uni-luebeck.de/lamaconv (Accessed 28/11/2023)

reader an understanding of the nature of the CAM protocol. In autonomous vehicles, the CA Basic Service (facilities layer) is responsible for operating the CAM protocol. It is composed of two services: (1) the sending of CAMs, including their generation and transmission, and, (2) the receiving of CAMs and the modification of the receiving vehicle's state in light of the received messages. The CA Basic Service controls how frequently CAMs are sent and interfaces to other services, such as SF-SAP which provides some basic security services (e.g. digital signatures and certificates) [5, §5.1].

CAMs are sent unencrypted as they are intended for all vehicles within range of the sender so encryption and decryption is not required. To ensure the authenticity of the sender (that a CAM sent from vehicle v actually came from vehicle v), digital signatures are used with a digital certificate to verify a CAM's origin and integrity. We focus on the protocol for sending and receiving CAMs and the threats that can be identified at this level rather than detailed cryptographic protocols and digital signing.

Once CAMs are received by surrounding vehicles, the receivers can modify their own state based on the received messages. In particular, if a vehicle receives a message from one proceeding it which indicates that the leading vehicle is slowing down, then the vehicle that received this CAM should also slow down in order to avoid a collision.

2.4 Combining Security and Verification

iUML-B and refinement in Event-B have been used to analyse a known security flaw (double tagging) in a network protocol [16]. Related, the TAMARIN prover has been used to formally analyse and identify one known functional correctness flaw and one unknown authentication flaw for a revocation protocol [17] in a vehicular networking system. Our work differs to these in that we use threat analysis to guide our verification rather than use formal methods to identify previously known bugs.

Vanspauwen and Jacobs have devised an approach to the static verification of cryptographic protocol implementations using their symbolic model of cryptography formalised in VeriFast [18]. They attach contracts to the primitives in an existing cryptography library. Their focus is on the verification of cryptographic protocols whereas we focus on using cyber security techniques to guide verification rather than verifying cryptographic protocol implementations.

Huang and Kang [19] use a probabilistic extension of the Clock constraint specification language (Ccsl) to analyse safety and security timing-related properties for a cooperative automotive system. They specified safety and security constraints including spoofing, secrecy, tampering and availability. Their work facilitates the verification, using the UPPAAL model-checker, of safety and security properties related to timing constraints. It does not, however, integrate results from a security engineering perspective to define these properties and focuses on timing-related properties.

The CSP process algebra has been used for protocol verification [20–22] focusing on authentication [22] and non-repudiation protocols [21]. Their approach involves specifying the relevant protocol, agents and environment in CSP [20]. Notably, they remark that, by modelling the protocol in CSP, they could provide a formal and verified specification of the protocol which allowed them to clarify the, usually informal, protocol description. Our modelling of the CAM protocol also allowed us to clarify what was described in the documentation but we focus on different properties as identified by our STRIDE threat analysis of the CAM protocol. Clarification of an informally specified protocol framework using abstract state machines as compared to a concrete implementation was considered in [23].

Kamali et al. [24] formally verified an autonomous vehicle platooning system to demonstrate the use of different formal techniques for distinct system subcomponents. In this case, autonomous decision-making, real-time properties and spatial aspects. Our approach uses different formal methods to verify distinct security-related properties of the CAM protocol at different levels of abstraction.

Security informed verification has been performed in different domains. For example, an IoT system was modelled using Alloy in [25], where specific sequences of events were used to model two attacks: eavesdropping and spoofing. Mitigations against each attack were also specified. A unified development approach for safety and security concerns for systems is advocated in [26] where the similarities and differences in each area are considered and the barriers to a joint approach discussed. These barriers are mainly concerned with a lack of a mature methodology and appropriate tool support for combining safety and security analyses. This paper, and our related work in [1], illustrates a novel methodology for combining these analyses.

In [27] a modelling and verification environment called AVATAR is used to capture both safety and security elements. The environment allows the system to be modelled and properties to be specified with verification via the UPPAAL and ProVerif tools. The paper focuses on security for an automotive example. However the security properties that can be considered are limited to confidentiality and authenticity.

Other related work includes the use of machine learning techniques to extract finite state machines from bank cards which implement variants of the EMV (Europay-MasterCard-Visa) protocol suite [28, 29]. These state machines can then be used for security analysis of the implemented protocol, although they do not appear to follow a cyber security threat modelling for their analysis which focuses predominantly on confidentiality and authentication properties.

3 Security-Minded Verification

In general, formal verification and threat modelling are distinct processes which are usually performed independently of one another. Therefore, a naïve approach to integrating these processes might consist of first performing formal verification and then subsequently carrying out threat analysis. Both the threat analysis or formal verification may result in requiring modifications to the system, thereby forcing a new system verification or analysis of the security threats to the system. This process could be iterated until the system meets its requirements. However, this can be ineffective since each independent analysis can lead to significant changes, and convergence can become a challenge. The advantage of our approach is that expertise from both the cyber security and formal methods domains is considered simultaneously.

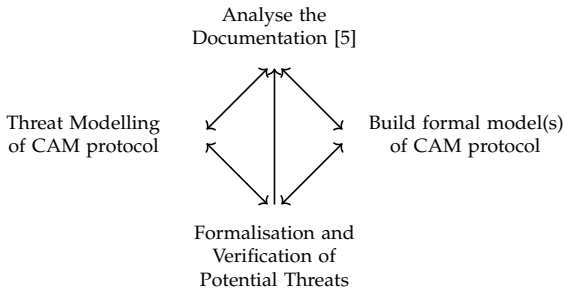

Fig. 1: Our high-level methodology for security-guided formal verification of the CAM protocol from [2].

	Tasks	Decisions	Outputs
Verification			
Threat Modelling			
Common to both		N/A	

TABLE 1: Key to Figures.

This potentially allows development time to be reduced by applying these two techniques in a complementary fashion.

To enhance the software engineering process and to encourage collaboration between cyber security and formal methods experts our prior work on the CAM protocol followed the high-level methodology in Fig. 1 [2]. We began by analysing the available documentation that informally describes the CAM protocol [5]. Then we independently carried out threat analysis and constructed formal models of the CAM protocol using the available documentation.

We subsequently developed a more detailed security-minded verification methodology that was used for an autonomous satellite docking scenario [1]. This paper further extends this methodology to include runtime verification as shown in Fig. 2. We explore how the artefacts produced in our prior work [2] persist in this improved methodology and include appropriate modifications of the previous formal models and threat analysis. We also demonstrate the addition of runtime monitors for the CAM use case.

We propose the security-minded verification methodology, shown in Fig. 2, as a way to combine the usually distinct approaches to formal verification and threat modelling. Our approach to integrating security and verification follows a tightly coupled procedure and is inspired by techniques such as the agile software development methodology which has been recognised to be well-suited to iteratively improve systems [30].

Our security-minded verification methodology can be integrated into system development. Each time that the system is modified, the system model is updated and the security-minded verification methodology is iterated upon until a successful verification is performed where the security risk is appropriately managed. This iterative, agile approach should not only be managed at development time, but also in operation, similar to other agile approaches. Such an approach of performing continual, through-life adaptation and verification can be accomplished *procedurally* with ease. For brevity, we do not introduce the operations element here. It should be recognised, however, that continually refining a system in operation introduces challenges, such as monitoring, resolving and updating that may impact on operation and cost, so is non-trivial in practice.

Our approach is depicted in Fig. 2; the top half corresponds to the usual formal verification methodology and the bottom half to the threat modelling approach usually followed by cyber security analysts. Instead of performing formal verification and a security analysis sequentially, aspects of these processes are performed in parallel. The fundamental change is that the security properties are formalised and checked as part of the verification of the system. Previously, these properties would have been identified due to system changes if the threat modelling identified that the risk of a threat to the system was too high. By performing verification of security properties the verification of the system has an explicit consideration of the important security aspects that have been identified. Further, failures in verification may uncover bugs that cause security flaws.

Both formal verification and threat modelling begin with a *System Definition* that provides a unified starting point for the security-minded verification methodology in Fig. 2. We strengthen this starting point from a formal verification perspective by incorporating the use cases identified via threat modelling during the construction of the formal model of the system. Interestingly, the integration of the use cases that are identified by the cyber security approach helps to provide developers building a formal model of the system with more detail and focuses them on the relevant scenarios.

Central to our approach is using the results from threat modelling to devise formal Security Properties via the Formalise Threats step of Fig. 2. By assessing the Threat Risk, we choose the security properties that are the most interesting/important and these are subsequently formally verified using a suitable formal method for static verification.

For complex systems it is unlikely that static verification methods can verify all of the identified properties. These methods tend to require a *model* of the system rather than the final implementation so there is often a *reality gap* between the verified model and implemented system [3]. To address this and provide alternative means of verifying difficult properties we include a Generate Runtime Monitors step. These dynamic checks can be performed on log files and/or during execution. This extra verification layer serves to increase our confidence that the system behaves correctly. Runtime verification was not present in our original methodology [1] but was added to support dynamic verification of systems. Using the static and runtime verification results, we evaluate whether the risk has been managed. This may inspire the inclusion of mitigations causing our methodology to loop.

We believe that our integrated approach to security-minded verification provides a more streamlined development process and that the use of the security use cases in the development of the formal model is hugely beneficial from a verification perspective. Likewise, the formalisation and subsequent verification of security properties of interest gives more weight to the security analysis and helps to investigate whether any mitigations put in place work correctly.

There are several points in the methodology where developers may need to revise the system, model(s) and/or

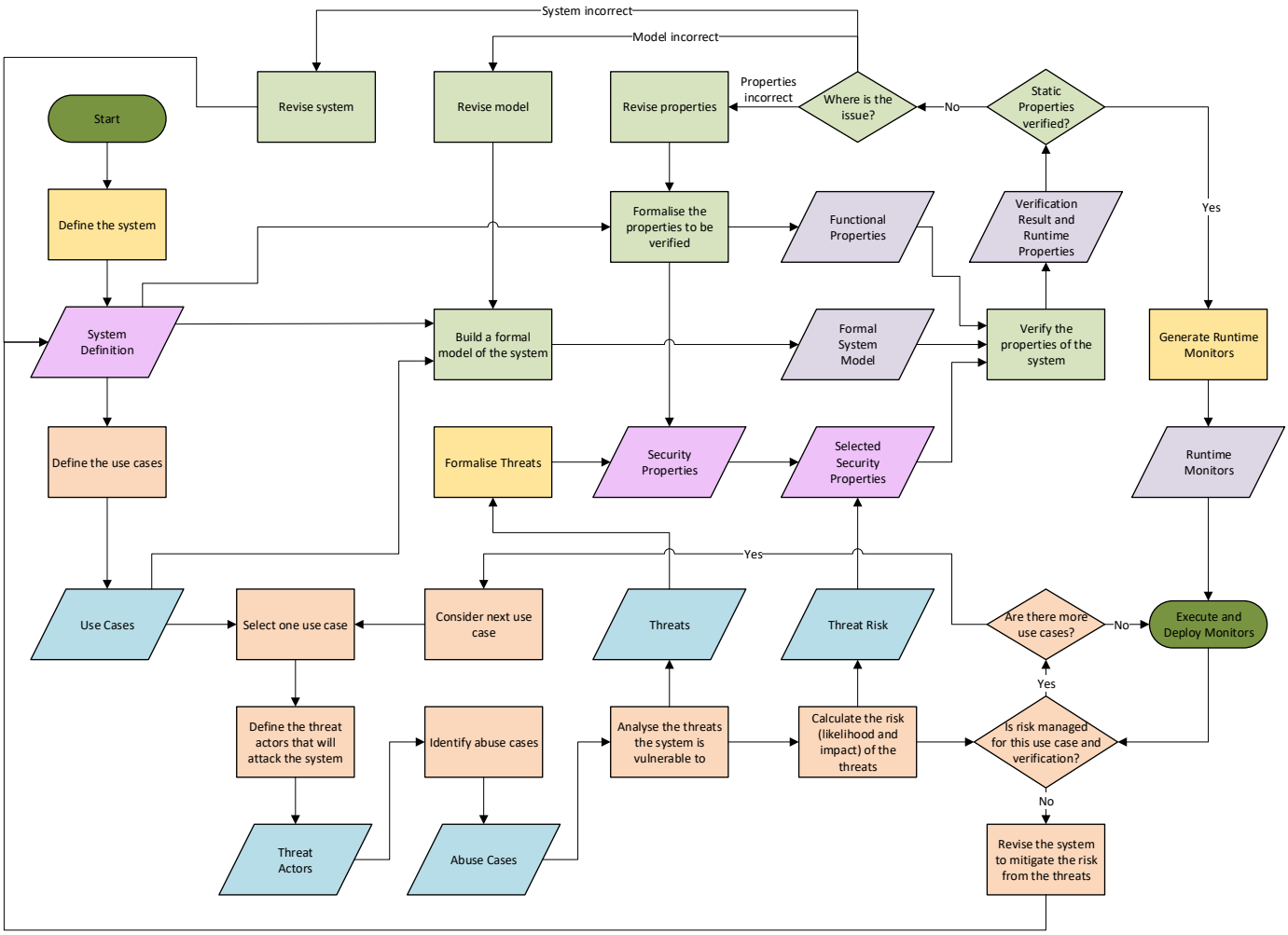

Fig. 2: Our integrated approach to combining verification and security analysis from our prior work [1] has been expanded to include the addition of runtime verification. The key in Table 1 indicates the nature of the elements in this diagram.

properties based on the outcomes of the verification and/or threat modelling activities. This is common in software development and can be time consuming if the system does not have a modular structure. We step through our methodology (Fig. 2), demonstrating how it can be applied in the case of the CAM protocol. We begin with threat analysis.

4 Threat Analysis of CAM

Common to both aspects of our methodology is starting by defining the system. In our case the documentation for the CAM protocol provided this high-level system definition [5]. After this step, we branch to building a formal model of the system and/or to defining the use cases for threat analysis. This section begins from the threat modelling perspective and follows the steps in the lower part of Fig. 2.

Threat analysis is important for ensuring the security of a system since it is used to identify the potential threats. There are many threat modelling methods including STRIDE, SAHARA, HARA, TARA and others that are suggested in multiple industry standards (i.e., ISO26262, SAE J3061). In this paper, we use the STRIDE classification [7].

We chose STRIDE since it is a stand-alone threat classification approach. This comes with multiple advantages, the first is that it is not necessary to modify the stages in an alternate threat modelling methodology to fit the integrated approach to threat modelling that we use [1]. The second is that the identified security properties derived from the threats are already classified by the type of threat. This simplifies decision making about classes of threats, specifically with which approach a security property should be verified. Finally, STRIDE is used by the European Telecommunications Standards Institute (ETSI) to perform an analysis of intelligent transportation system threats, allowing us to link the threats that we focus on to those ETSI identifies.

CAMs are vital to safe CAV systems since they are used by each vehicle to inform surrounding vehicles of their current status. Each vehicle must trust that the values contained within a CAM are timely and accurate. If this is not the case then autonomous vehicles could make incorrect and even unsafe decisions. We start by identifying suitable Use Cases.

4.1 Use Cases

ETSI TR 102 638 [31, §6] defines a basic set of applications with four high-level application classes: (i) Active road safety, (ii) Cooperative traffic efficiency, (iii) Co-operative local services, and (iv) Global internet services. Of these,

(i) and (ii) specifically target CAM-provided functionality. A wide variety of specific use-cases are provided in [31, Table A.1] including approaching emergency vehicle, wrong way driving, V2X road obstacle warning, overtaking vehicle, intersection management and V2V lane change assistance.

To focus our analysis and illustrate the application of our security-minded verification methodology (Fig. 2) we use a single use case of collision detection and avoidance. Here, CAMs are used to disseminate vital information (e.g., position, velocity and heading) which is used by vehicles to detect if a collision will occur. If a collision will occur, then vehicles will take action by reducing their speed.

Although we focus the remainder of this paper on the collision detection and avoidance use case, in practice each relevant use case should be considered for completeness. Following Fig. 2, once a use case has been selected then the next step involves identifying the associated Threat Actors.

4.2 Threat Actors

A threat actor is "an individual or a group posing a threat" [32, Appendix B] and a threat is an action with the potential to negatively impact a target. We identify the threat actors for our use case in Table 2. These include individuals, insiders, ad-hoc groups and large organised groups [8, Appendix D]. ETSI have specified multiple high-level descriptions of threat actors in [33, §10]:

- "malicious or mischievous use of an ITS-S (Vehicle) as an attack proxy by a remote agent; or"
- "malicious or mischievous use as an ITS-S (Vehicle) providing false or misleading information to other vehicles"

Understanding threat actors is important for understanding what attacks may be performed on a vehicle, how they will be performed, and why. Different threat actors will have varying capabilities, meaning that an attack such as DoS from one threat actor (e.g., an outsider) may have limited impact compared to DoS from a group with significant resources. After identifying the potential threat actors, we examine the specific Abuse Cases which represent the attacks they intend to perform and the aims of these attacks.

4.3 Abuse Cases

Abuse cases [36] define the interaction between one or more threat actors and their target system where those interactions are intended to cause harm. There are many possible attacks that an adversary may perform against CAMs. To focus our threat analysis we specifically consider the generation of CAMs (defined in [5]). In this scenario, an adversary wants to interact with ITS stations generating CAMs or generate its own CAMs. This could be for many reasons including: to cause a collision, to cause reputation damage to commercial competitors, for fun, or exploration of what is possible.

Table 3 describes four abuse cases where an adversary attempts to impact the sending, receiving or use of CAMs for the collision detection and avoidance use case. For example, the first abuse case in Table 3 describes the scenario where an attacker might intentionally cause vehicles to collide.

The next step in our methodology (Fig. 2) involves the identification of specific Threats for the CAM protocol based on the use case, actors and abuse cases identified previously.

4.4 Identifying Threats

We now identify the threats to the system that formed part of the abuse cases in §4.3. We use the STRIDE framework to classify these threats and compare them to those identified by ETSI in order to evaluate if this is a sufficiently thorough enumeration of threats. We focus on CAMs generated by vehicle On-Board Units (OBU) rather than Road Side Units (RSU) for two reasons: (1) the algorithm for CAM generation from OBUs is more complex and thus more interesting from a formal methods perspective, and (2) it is attacks against vehicular OBU CAM generation that will lead to potential safety violations (such as collisions between vehicles). For the CAM protocol, we specialise the STRIDE threats as:

Spoofing: an attacker sends CAMs masquerading as another vehicle.

Tampering: an attacker tampers with a CAM sent by another vehicle.

Repudiation: a vehicle denies sending a CAM that it has actually sent.

Information Disclosure: a vehicle receives and accesses information from CAMs which are not intended for them.

DoS: CAMs are not sent within a reasonable time frame.

Elevation of Privilege: an attacker sends CAMs without having permission to do so.

ETSI, who publish the CAM specification, have previously performed a general threat assessment [33] for Intelligent Transport Systems (ITS). As part of this analysis the following threats were identified:

[V-V1] DoS: A vehicle receives such a high volume of CAMs that it cannot process in a timely manner causing CAMs to be missed.

[V-V2] DoS: An attacker jams radio signals preventing CAMs from being received.

[V-V3] Tampering: False information is received.

[V-V4] Spoofing: When a vehicle is unable to verify the authenticity of a CAM quickly, spoofed CAMs may be used to update local state.

[V-V5] Tampering: A CAM is received that contains incorrect contents even when the authenticity can be verified.

[V-V6] Replay of old or expired CAMs.

[V-V7] Malware can be installed onto vehicles.

[V-V8] Information Disclosure: CAMs can be eavesdropped.

[V-V9] Repudiation: A vehicle may not send CAMs as frequently as it should.

There exists overlap between our STRIDE threat modelling and ETSI's, however, there are also differences. Two threats (**[V-V6]** and **[V-V7]**) are not applicable to our investigation of CAMs, as they are either out of scope (**[V-V7]**) or protected against by mechanisms provided by the ETSI CAM stack (**[V-V6]**). Other overlaps and differences include:

Spoofing: **[V-V4]** matches our definition.

Tampering: **[V-V3]** and **[V-V5]** match our definition.

Repudiation: is not considered by ETSI in the same way.

Information Disclosure: **[V-V8]** matches our definition.

DoS: **[V-V9]** matches our definition which focuses on sending CAMs. **[V-V1]** does not match our definition as it focuses on ITS stations receiving and processing CAMs in a timely manner. **[V-V2]** does not match our definition as signal jamming is out of scope.

	Threat Actor	Example	Goals & Motivations	Capabilities	Presence	Resources
Solo	Outsider	Hacktivist / Thief	Personal satisfaction; Passion; Ideology.	Limited	Remote access	Minimal
	Insider	Employee / Contractor	Financial gain; Discontent	Moderate to High	Internal access	Internal knowledge
Group	Ad hoc	A group formed in response to an event (e.g., climate)	Dependant on group purpose: Ideological, financial, political	Limited to Moderate	Remote access	Limited knowledge and financial
	Established	A group (e.g., terror)		Moderate to High	Remote access	Moderate knowledge and financial
Organisation	Competitor	Provides similar services / equipment	Corporate espionage; Financial gain; Reputation damage	Dependent on size of organisation	Remote access	Dependent on size of organisation
	Partner	A supplier with which a relationship is ending	Information gain; Financial gain		Limited internal access; Knowledge of internal structure	
Nation-State		Geopolitical rival	State rivalry; Geopolitics	Sophisticated; Coordinated; Access to state secrets	Remote and internal access	Extensive knowledge & financial; Advanced equipment

TABLE 2: Example Threat Actors based on [8, Appendix D] and [34] with dimensions from [35].

Abuse Goal	Description	Threat Actors	Techniques
Intentionally cause a collision	Induce a collision by preventing V2V communication, spoofing V2V communication with incorrect information, or repudiating previously send information	Individual / Group intent on causing harm; Competitor causing reputation damage	External jammer
Reduce traffic on intended route	Impersonate or otherwise disseminate messages that imply heavy traffic (when there is none) on the adversary's intended route, or alternatively that traffic is lower (when it is not) on other routes the adversary does not intend to take.	Individual	Pose as genuine vehicle
Facilitate vehicle theft via influencing of its route or position	An adversary may indicate an accident or roadworks exists where there is none, or manufacture evidence of a fake accident or roadworks in order to manipulate a vehicle into taking a route where theft of the vehicle becomes easier	Individual; Established Criminal Group	Malware on target vehicle
Masquerade in order to be given priority	An adversary may emulate an emergency vehicle based on previously observed characteristics to be given priority in traffic.	Individual	Pose as genuine vehicle

TABLE 3: Abuse cases derived from [33], with threat actors and relevant STRIDE threats identified.

Elevation of Privilege: is not considered by ETSI.

These differences are likely caused by a different threat classification framework. For example, **[V-V9]** is a property that we have classified as DoS, but ETSI has classified as Repudiation [33, Table 13]. ETSI chose Repudiation as they focused on the lack of a requirement for vehicles to maintain auditable logs of the messages sent, whereas, in this paper, we focus on the lack of messages sent being a DoS to other vehicles. This highlights the subjectivity in how threats are classified and that they have not been formally defined.

At this stage our methodology branches (Fig. 2). The Threats are formalised for the formal verification branch. On the other branch, we discuss how Threat Risk is evaluated.

4.5 Threat Risk

Assessing risk can be challenging [8]. Ideally a quantitative data-driven approach would be used, however, for novel systems there is insufficient data to undertake a quantitative risk assessment [8]. For security-minded verification, a detailed and highly accurate risk assessment is not required, as risks are only used to prioritise which threats are focused on during verification. Instead, quick to perform indicative qualitative risk assessed by experts is sufficient and preferred. Further, as we lack quantitative data we describe the likelihood of attacks being performed as *anticipated*.

Table 5 contains the risk assessment of the specific types of threats identified for the abuse cases in Table 3. The risk matrix in Table 4 is used to assign a risk value from the impact and anticipated likelihood of the assessed threats. Impact is determined by the attack's effect on safety. A high impact attack would lead to an unsafe situation for the vehicle's occupants, or nearby pedestrians. From these results we can see that Spoofing and DoS attacks tend to have a high or medium risk associated with them (Table 5). Some Repudiation threats also have a medium impact. Next, we use this risk analysis to distil the Selected Security Properties (Fig. 2) that are most important for verification.

4.6 Selected Security Properties

We have identified multiple STRIDE threats and as part of the threat analysis process have identified the risk that those threats pose. Based on our analysis, we conclude that

		Anticipated Likelihood		
		High	Medium	Low
Impact	High	High	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low

TABLE 4: Risk is a combination of impact and likelihood.

Spoofing and DoS are the two most relevant and important threats for our case study. For completeness, we describe each of the threats in detail here and highlight why they are or are not relevant and/or important for this case study.

Tampering: In practice, tampering of CAM content is mitigated via digital signatures and certificates, the verification of which is beyond the scope of this paper (related work in [38]). Other aspects of the system may be tampered with, such as inputs to the CAM generation algorithm, under which the verification needs to ensure that the availability properties are maintained.

Repudiation: We previously modelled and verified a Repudiation property for CAM in [2]. However, in practice and similar to Tampering, Repudiation is mitigated by the use of digital signatures and certificates. If a CAM is received by a vehicle and the signature is verified, this means that the private key of the sender must have been used to sign the message. If the private key is revealed then the origin of the sender cannot be guaranteed. This threat can be mitigated by adding the compromised certificate to a Certificate Revocation List [39] or by using other public-key cryptography schemes [40].

Elevation of Privilege: The CA Basic Service, responsible for operating the CAM protocol, interfaces with the SF-SAP security entity [5, §5.1 & §6.2.2] which provides access to security services for CAM such as digital signing and certificates. Certificates belonging to each ITS station indicate the holder's permissions including whether they are allowed to send CAMs. ITS stations receiving CAMs accept incoming messages if the permissions in the sender's certificate allow them to send CAMs. Certificates of an ITS station can be requested from the Certification Authority [39]. Elevation of Privilege attacks could enable a malicious ITS station to send CAMs when lacking permissions as defined in [5, §6.2.2]. The response to receiving one of these messages that lacks the necessary permissions is to not accept it (i.e. not process the information contained in the CAM).

Information Disclosure: We don't analyse Information Disclosure as CAMs are intended for all who receive them.

To our knowledge, no formal, mathematical definition of the STRIDE properties exists since they are to be specialised for a given system. However, if we wish to include these in our formal verification of the CAM protocol then we must more closely consider properties of interest (Spoofing and DoS):

Spoofing: an attacker pretends to be another vehicle and sends false information about that vehicle (e.g., speed) in CAMs. This could cause vehicles to collide and we analyse this using Promela/SPIN in §5.1 by modelling an attacker of the system.

DoS: a compromised vehicle does not send CAMs within a reasonable amount of time. If a vehicle sends too many CAMs then the network becomes overloaded. Conversely, if a vehicle does not send CAMs frequently enough then the most recent CAMs sent may be deemed out of date and thus ignored. In particular, a replay attack could occur where an attacker or a compromised vehicle resends CAMs that have already been sent causing a network overload. If suitable measures are not taken to ensure that the time at which the message was sent was not too far in the past then vehicles may react to an out-of-date message. We address this using Dafny in §5.2 by verifying an availability property of the algorithm for sending CAMs. This is examined further in §5.3 where discuss deploying runtime monitors to recognise such situations.

Based on our threat analysis, we consider these threats to be the most relevant/likely with respect to the CAM protocol and we use them to guide our formal verification effort[3].

5 Formal Verification

This section describes the formal verification steps in our methodology (upper half of Fig. 2). It is increasingly the case that complex autonomous systems require multiple verification approaches to adequately verify different properties of the system at varying levels of abstraction [11]. As a result, there are three distinct approaches used here: (1) Promela/SPIN to investigate Spoofing via model-checking, (2) Dafny to verify algorithmic properties and examine DoS, and (3) runtime verification to capture properties that could not be verified by other methods and to support properties that were verified statically at runtime. We begin by discussing model-checking with Promela/SPIN.

5.1 Model-Checking with Promela/SPIN

In this case study, it is easy to see that safety and security are inextricably linked. For CAV systems, the most important safety property is that collisions are avoided at all costs (Table 5). Therefore, an attacker of the system who is attempting to cause harm will likely target security vulnerabilities that have the potential to violate this safety property. We recognise that there may be malicious vehicles on the road that are attempting to cause collisions, perhaps a disgruntled taxi driver who is unemployed due to the adoption of autonomous vehicles. Such a collision could be caused by Spoofing the CAMs sent between vehicles. Our analysis of Spoofing and how it can impact the safety of the CAV system is captured here in a SPIN analysis of a simplified scenario involving CAMs between vehicles in a platoon/convoy.

5.1.1 Formal System Model and Functional Properties

We begin, following our methodology (Fig. 2), by building a Formal System Model then we identify and verify the basic functional safety property that we are interested in. We investigated message passing between multiple autonomous vehicles by applying SPIN to an abstracted Promela model for sending and receiving CAMs. Fig. 3 illustrates three vehicles travelling in a platoon/convoy: one leader; one middle; and one tail vehicle. The leader sends messages

3. Artefacts available at: https://github.com/autonomy-and-verification/security-minded-verification (Accessed 28/11/2023)

Adversary Goal	Type	Impact	Anticipated Likelihood	Risk	Rationale and Justification
Intentionally cause a collision	S	High	Medium	High	Information containing spoofed data, or spoofing the presence of other vehicles could lead to incorrect decision making [33, Table 10].
	R	Medium	Low	Low	Repudiating previously sent accurate information will have limited impact if the information has already been processed. Mitigations exist [33, §11.4.1.4].
	D	High	Medium	High	Preventing receiving or processing CAMs could lead to unsafe situations. Partly caused by expensive public key cryptography operations [33, §11.3.7 & §11.3.10] to mitigate other threats. Lowering broadcast rate reduces timeliness of information [33, §11.3.1] and will not prevent a capable adversary. Rated critical in [33, Table 9].
Reduce traffic on intended route	S	Low	Medium	Low	Dissemination of incorrect information can be used to negatively impact traffic management techniques [37].
	D	Low	Medium	Low	Preventing access to disseminated information (e.g., from RSUs) means that vehicles are unaware of how they are being directed by a traffic management system.
	E	Medium	Low	Low	Adversary could escalate capabilities to act as a traffic management system. Public Key Infrastructure (PKI) mitigates this [33, §11.3.7].
Facilitate vehicle theft	S	Medium	Medium	Medium	Dissemination of incorrect information to cause traffic in specific areas of the road network [37].
	R	Low	Low	Low	Make claims that a vehicle acts upon and then repudiate them such that the vehicle needs to keep re-evaluating its planned route.
	D	High	Medium	High	Prevent access to routing information. Prevent target vehicle from sending safety data that could be used for tracking or theft recovery.
Masquerade to be given priority	S	Medium	Medium	Medium	While plausibility tests could be used to identify masquerading via a heuristic [33, §11.3.20], they will incur non-zero false negatives and false positives which may lead to incorrect behaviour.
	I	Medium	High	High	Information disclosed by the system could be used to build a profile that the adversary will replicate [33, Table 12].

TABLE 5: Example threat risk allocation for STRIDE threats (classified by 'Type') forming the abuse cases in Table 3 with risk calculated using Table 4. Impact and anticipated likelihood have been estimated based on details provided by other sources.

to both other vehicles while the tail sends messages to the middle vehicle only. The middle and tail vehicles follow simple protocols, as outlined in [2]:

- If either vehicle receives no CAMs then it continues unchanged.
- If the middle vehicle receives exactly one CAM then sets its speed to half the speed in the CAM[4].
- If the middle vehicle receives two CAMs then it sets its speed to the average of the two speeds (rounded down).
- If the tail vehicle receives a CAM then it sets its speed to that in the CAM.

We used the following default conditions to analyse this Promela model with SPIN: the leader chooses a random discrete speed of 1, 3, or 5 every three time steps and communicates this to the other vehicles. At the next time step, the middle vehicle adjusts its speed and at the time step after that the tail vehicle adjusts its speed and communicates this new speed to the middle vehicle.

Each time that a vehicle modifies its speed, the distances of the middle and tail vehicles from the leader are calculated assuming that the speeds are measured in distance per time step. The middle vehicle starts at a distance of 10 units behind the leader and the tail vehicle at a distance of 25 units behind the leader. This is more detailed than our Promela model in [2] which did not calculate vehicle distances. The properties that are checked have been updated to reflect this.

4. Only occurs at initialisation when the other vehicle's speed is 0.

The safety property that we verified is that the vehicles can never crash, that is that the middle vehicle's position is never less than or equal to zero and that the tail vehicle's position is never less than or equal to that of the middle vehicle. We write this in temporal logic as:

$$\Box\neg(vpos \leq 0) \wedge \Box\neg(tpos \leq vpos)$$

where '$\Box$' is LTL's [41] "always" operator, $vpos$ is the position of the middle vehicle relative to the lead vehicle and $tpos$ is the position of the tail vehicle relative to the lead vehicle.

We have successfully verified that this property holds of our model using SPIN. Next, we use this model to investigate how an attacker executing a Spoofing attack could lead to an unsafe scenario for the vehicle platoon.

5.1.2 *Investigating Spoofing*

Inspired by the threat analysis in §4.6, we have modelled a Spoofing attack in Promela for the above scenario. In order to analyse this kind of threat, we include an attacker as a process in our Promela model as shown in Fig. 3. At one point in the execution trace the attacker may:

- replace a message on the channel between the leader and middle vehicles stating that the leader's speed is 1 or 5 (lines 8–13 in Fig. 4)
- replace a message on the channel between the tail and middle vehicles stating that the tail's speed is 1 or 5 (lines 14–19 in Fig. 4)

In each of these cases, both the speed, whether to insert a message, and the time that the message is inserted are chosen

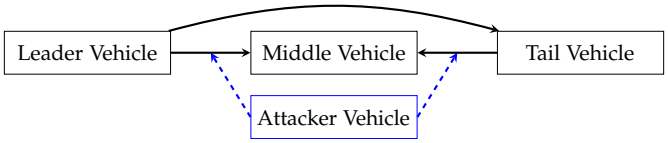

Fig. 3: Our three vehicle model. CAMs are sent from the leader and tail to the middle vehicle, and from the leader to the tail. An attacker vehicle executes a Spoofing attack.

```
1  proctype attacker(chan l_in, t_in) {  /* attacker */
2    printf("starting\n");
3    int lin1, tin1;
4    bool head = 0, tl = 0;
5    A:  (clock > 10); /* wait until under way */
6      if
7        :: (head == 0 && len(l_in) != 0) →
8          atomic{ printf("ins. vspeed of 1\n");
9          l_in?lin1; l_in!1; head = 1; } goto A;
10       :: (head == 0 && len(l_in) != 0) →
11         atomic{ printf("ins. vspeed of 5\n");
12         l_in?lin1; l_in!5; head = 1; } goto A;
13       :: (tl == 0 && len(t_in) != 0) →
14         atomic{ printf("ins. tspeed of 1\n");
15         t_in?tin1; t_in!1; tl = 1; } goto A;
16       :: (tl == 0 && len(t_in) != 0) →
17         atomic{ printf("ins. tspeed of 5\n");
18         t_in?tin1; t_in!5; tl = 1; } goto A;
19       :: (clock <= 100) → goto A;
20       :: (clock > 100) → goto FIN;
21     fi;
22   FIN:  printf("finishing\n")
23 }
```

Fig. 4: Promela model (updated from [2]) of the attacker who sends false speed messages between the vehicles violating our safety property and causing a collision.

at random. Running SPIN with this attacker model and the model described above, we can see that our safety property:

$$\Box\neg(vpos \leq 0) \wedge \Box\neg(tpos \leq vpos)$$

has been violated and we get a counter-example trace. In this case, a situation where the attacker informs the middle vehicle that the leader's speed is 1 when it is in fact 5. This causes the middle vehicle to slow down sufficiently so that the tail vehicle (matching the leader's speed) crashes into it. Thus, we have shown that a breach in security (Spoofing by an attacker) can potentially lead to an unsafe scenario.

We are now on the No branch of Static Properties Verified? in Fig. 2. Thus, we question whether the issue is caused by the system, model or properties and respond with suitable modifications. Here the system is the point of failure and we discuss a possible way to mitigate this threat by modifying the system definition to include LIDAR sensor data in §6.

Our Promela model describing an attacker and three vehicles is only one potential scenario, particularly as there may be more vehicles in a real-world scenario. To our knowledge, there is no systematic way of identifying all possible models of the system that include a Spoofing attack. However, we can systematically work through the attributes sent in CAMs as likely vulnerable to Spoofing to examine how such attacks can influence safe system behaviour.

5.2 Deductive Verification with Dafny

To examine the CAM protocol at algorithm-level, we construct and verify a CAM send and receive implementation using Dafny [10]. We focus on two basic methods; `sendCAM` (Fig. 5) and `receiveCAM` (Fig. 8). We have formalised the specification of CAM using the available documentation [5, §6.1.3] and followed its nomenclature. The Dafny model that we present here is based on the one in [2] with some updates that we highlight throughout this section.

Here, we focus on the DoS security threat at algorithm-level. In our implementation we have simplified the structure of CAMs from the ASN.1 encoding to focus on the semantic contents of the message as follows:

`CAM(id:` **`int`**`,time:` **`int`**`,heading:` **`int`**`,speed:` **`int`**`,position:` **`int`**`)`

Here, `id` refers to the vehicle that is sending the CAM and `time` is the timestamp at which the CAM was sent. These attributes are required by the documentation [5]. CAMs are sent periodically, or when any of the status information (e.g., speed) contained in the CAM has changed since the last one was sent. Several global constants can be configured, with their default values from [5] shown in brackets below:

- `T_GenCamMin`: The minimum time between CAM generations (100 ms).
- `T_GenCamMax`: The maximum time between CAM generations (1000 ms).
- `N_GenCamDefault`: Used to limit the rate of CAM generation (3).
- `N_GenCamMax`: Maximum value of `N_GenCam` (3).
- `headingthreshold`: Heading trigger threshold (4°).
- `speedthreshold`: Speed trigger threshold (0.5 m/s).
- `posthreshold`: Distance trigger threshold (4 m).

Using the above CAM structure and variables we now describe our Dafny implementation of the `sendCAM` algorithm.

5.2.1 Sending CAMs

Fig. 5 contains the verified Dafny code corresponding to the `sendCAM` algorithm which is responsible for generation and transmission of CAMs. We have made modifications to the Dafny model from [2] that we indicate in the text below.

We capture the fact that CAMs should be sent periodically within the time bounds specified by the CA Basic Service, by the method's first two input variables. `T_CheckCamGen` describes how often to check if another CAM should be sent and `T_GenCam_DCC` describes the minimum time interval between two consecutive CAM generations. It returns a sequence of CAMs that have been sent, called `msgs`, and the current time given by the variable, `now`.

Other parameters specify the identity of the vehicle (`j`), at what time the CAM generation starts (`start` was not present in our previous model [2]) and the maximum number of messages to send before terminating (`max_msgs`). Typically CAM generation would continue until halted, however, Dafny requires that termination is proven, so `max_msgs` allows a finite number of messages to be considered instead.

The pre-conditions, indicated by the **`requires`** keyword (lines 2–3), provide constraints on these variables, such as the bounds for `T_GenCam_DCC` (line 2) [5, §6.1.3]. The conditions on line 3 were not present in [2] but were added to improve our reasoning related to DoS. The addition of these properties caused us to update the post-condition on line 4 to refer to `start` and thus improve our concept of timing.

The post-conditions on lines 4–6 use the **`ensures`** keyword to specify that the expected number of CAMs have

```
1  method sendCAM(T_CheckCamGen: int, T_GenCam_DCC: int, j: int, start: int, max_msgs: int) returns (msgs: seq<CAM>,now: int)
2    requires 0 < T_CheckCamGen ≤ T_GenCamMin ∧ T_GenCamMin ≤ T_GenCam_DCC ≤ T_GenCamMax;
3    requires start ≥ 0 ∧ max_msgs ≥ 0 ∧ 0 < N_GenCamDefault ≤ N_GenCamMax;
4    ensures T_GenCam_DCC * |msgs| ≤ (now - start) ≤ T_GenCamMax * |msgs|;
5    ensures |msgs| ≥ 2 ⟹ ∀ i: int • 1 ≤ i < |msgs| ⟹ T_GenCam_DCC ≤ msgs[i].time - msgs[i-1].time ≤ T_GenCamMax;
6    ensures |msgs| = max_msgs;
7  {
8    now, msgs := start, [];
9    var T_GenCam, T_GenCamNext, N_GenCam, trigger_two_count := T_GenCamMax, T_GenCam, N_GenCamDefault, 0;
10   var LastBroadcast, PrevLastBroadcast, prevsent := now, now, [];
11   var heading, speed, pos := GetHeading(j, now), GetSpeed(j, now), GetPosition(j, now);
12   var prevheading, prevspeed, prevpos := Nil, Nil, Nil;
13
14   while (|msgs| < max_msgs)
15   decreases max_msgs - |msgs|;
16   invariant 0 ≤ |msgs| ≤ max_msgs ∧ 0 < N_GenCam ≤ N_GenCamMax;
17   invariant T_GenCamMin ≤ T_GenCamNext ≤ T_GenCamMax ∧ T_GenCamMin ≤ T_GenCam ≤ T_GenCamMax;
18   invariant start ≤ PrevLastBroadcast ≤ now;
19   invariant now = LastBroadcast;
20   invariant now - T_GenCamMax ≤ PrevLastBroadcast ≤ LastBroadcast;
21   invariant |msgs| ≥ 1 ⟹ msgs[|msgs|-1].time = LastBroadcast;
22   invariant |msgs| ≥ 2 ⟹ msgs[|msgs|-2].time = PrevLastBroadcast;
23   invariant now > start ⟹ T_GenCam_DCC ≤ LastBroadcast - PrevLastBroadcast ≤ T_GenCamMax;
24   invariant now > start ⟹ CAM(j,now,heading,speed,pos) in msgs;
25   invariant now > start ⟹ |prevsent| + 1 = |msgs|;
26   invariant |msgs| ≥ 2 ⟹ ∀ i: int • 1 ≤ i < |msgs| ⟹ (T_GenCam_DCC ≤ msgs[i].time - msgs[i-1].time ≤ T_GenCamMax);
27   invariant T_GenCam_DCC * |msgs| ≤ (now - start);
28   invariant now > start ⟹ (now - start) ≤ T_GenCamMax * |msgs|;
29   {
30     prevsent, PrevLastBroadcast, T_GenCam := msgs, LastBroadcast, T_GenCamNext;
31     var statechanged := false;
32     now := now + T_GenCam_DCC; // Advance time to the earliest a CAM can be sent
33     while (true) // Find the time at which information has changed or we have waited T_GenCam
34     decreases LastBroadcast + T_GenCam - now;
35     invariant now - LastBroadcast ≤ T_GenCam_DCC ∨ now - LastBroadcast ≤ T_GenCam; {
36       heading, speed, pos := GetHeading(j, now), GetSpeed(j, now), GetPosition(j, now);
37       statechanged := HeadingChng(prevheading, heading) ∨ SpeedChng(prevspeed, speed) ∨ PosChng(prevpos, pos);
38       if (statechanged ∨ now - LastBroadcast + T_CheckCamGen ≥ T_GenCam) {
39         break; // Send CAM now because values have changed sufficiently or the interval time limit has been reached
40       } else {
41         now := now + T_CheckCamGen; // Sleep for T_CheckCamGen to advance time
42       }
43     }
44     msgs := msgs + [CAM(j,now,heading,speed,pos)];
45     if (statechanged) { // Trigger 1
46       T_GenCamNext := now - LastBroadcast;
47       trigger_two_count := 0; // Reset
48     } else if (now - LastBroadcast ≥ T_GenCam) { // Trigger 2
49       trigger_two_count := trigger_two_count + 1;
50       if (trigger_two_count = N_GenCam) { T_GenCamNext := T_GenCamMax; }
51     }
52     LastBroadcast := now;
53     prevheading, prevspeed, prevpos := Prev(heading), Prev(speed), Prev(pos);
54   }
55   return msgs, now;
56 }
```

Fig. 5: Dafny implementation of the `sendCAM` algorithm with a Denial of Service post-condition specified on lines 7–9.

been sent within the required time bounds. This corresponds to the DoS threat by ensuring that messages are sent on time and arrive within specified time bounds.

We initialise the relevant local variables on lines 8–12. Some of these variables are specified in the CAM documentation but others have been included for implementation purposes. In particular, `T_GenCam` on line 9 represents the current upper limit of the CAM generation interval, by default this is equal to `T_GenCam_Max` [5, §6.1.3]. We assume the existence of verified helper functions for `GetHeading`, `GetSpeed` and `GetPosition` as used on line 11.

The first loop iterates until `max_msgs` CAMs are sent. The loop variant is specified using the **`decreases`** keyword (line 15) to prove termination. The loop invariants on lines 16–18 ensure that particular variables stay within allowed bounds. For example, the invariant on line 16 relates to the post-condition on line 6 by specifying that the number of CAMs sent so far is less than or equal to `max_msgs`. Invariants on lines 19–25 ensure that once time has begun then one CAM is sent per iteration. These have been modified from our model [2] to refer to `start`. The invariant on line 26 ensures that once more than one message has been sent, the interval between these CAMs is correctly bound by the minimum and maximum broadcast rate. The invariants on lines 27–28 ensure that the number of CAMs sent is correctly bound by the rates at which CAMs can be sent and the time that has elapsed. These have been modified from our prior model [2].

During each loop iteration we update the appropriate variables. Note that we increment the current time, `now`, by `T_GenCam_DCC` to allow time to advance until the earliest time that the next CAM can be sent (line 32). This inner loop (lines 33–43) checks if any state information has changed and updates the `statechanged` variable accordingly. If the vehicle's state has changed (checked in Fig. 6) or it is time to send another CAM then we break from this inner loop. Otherwise, we loop to allow time to advance until either

```
datatype Prev<T> = Nil | Prev(value: T)
function method HeadingChng(s: Prev<int>, c: int): bool {
  match s
  case Nil ⇒ true
  case Prev(x: int) ⇒ abs(c - x) ≥ headingthreshold
}
function method SpeedChng(s: Prev<int>, c: int): bool {
  match s
  case Nil ⇒ true
  case Prev(x: int) ⇒ abs(c - x) ≥ speedthreshold
}
function method PosChng(s: Prev<int>, c: int): bool {
  match s
  case Nil ⇒ true
  case Prev(x: int) ⇒ Distance(c, x) ≥ posthreshold
}
```

Fig. 6: Dafny specification of detecting whether the vehicle state has sufficiently changed.

```
function method GetHeading(j: int, now: int):int
  ensures 0 ≤ GetHeading(j, now) ≤ 359

function method GetSpeed(j: int, now: int):int
  ensures 0 ≤ GetSpeed(j, now) ≤ 100

function method GetPosition(j: int, now: int):int
  ensures -10000 ≤ GetPosition(j, now) ≤ 10000
```

Fig. 7: Dafny specification of the helper functions that are used to ascertain the status of the vehicle.

the state has changed or sufficient time has passed since the last CAM was sent. Based on why the CAM was sent, i.e. whether the state changed or it was time to send a CAM, the relevant variables are updated (lines 46–52) [5, §6.1.3].

In this way, the Dafny algorithm illustrated in Fig. 5 is verified with respect to the STRIDE DoS threat, which was also identified by ETSI [33] (**[V-V9]** in §4.4). We verified other correctness properties that were derived from the documentation [5], e.g., specific variables remain within particular bounds and loops terminate. We include Fig. 6 which contains the Dafny implementations of the functions that we use to determine whether the vehicle's state has changed. These were not contained in our previous work [2].

5.2.2 Obtaining the Status of the Vehicle

The sensors can be used to obtain the vehicle's status. For example, GNSS sensors provide position, wheel rotation sensors provide speed and a magnetometer provides the heading. The returned sensor data should lie within specific ranges. Our Dafny function specification (Fig. 7) of these sensor queries omits their implementation, but specifies bounds on the returned values. In this way, CAM generation can be proved correct if any of these values are returned. However, a downside is that the Dafny code can no longer be compiled and executed in Visual Studio Code, as these functions do not have an implementation. To overcome this, we assigned random values to the output of these functions to test that the Dafny implementation behaves as expected.

If an attacker has access to the internal systems of a vehicle, they may manipulate the data provided to the CAM generation algorithm. To ensure that CAM generation continues to maintain availability and sends messages within the required time bounds, the post-conditions can be removed from these functions. Even when they are removed, CAM generation is verified to be correct. This shows resilience against an adversary capable of manipulating the sensor

```
method receiveCAM(j: int, now: int, cams: seq<CAM>,
     TTSFactor: real, HeadDist: int, SensDist: int)
     returns (brake: bool)
requires |cams| > 0;
requires ∀ i: int • 0≤i<|cams| ⟹ cams[i].time < now;
requires TTSFactor ≥ 1.0 ∧ HeadDist ≥ 0 ∧ SensDist > 0;
ensures brake = ∃ k: int • 0 ≤ k < |cams|
  ∧ abs(GetHeading(j,now) - cams[k].heading) ≤ HeadDist
  ∧ (now - cams[k].time ≤ T_GenCamMax ∨
     SensorDistance(j,cams[k].id,now) < SensDist)
  ∧ TimeToStop(j,now) * TTSFactor ≤ TimeToCollision(j,
      now,cams[k]);
{
  brake := false;
  var i := 0;
  while (i < |cams|)
  decreases |cams| - i;
  invariant 0 ≤ i ≤ |cams|;
  invariant brake = ∃ k: int • 0 ≤ k < i
    ∧ abs(GetHeading(j,now) - cams[k].heading) ≤
        HeadDist
    ∧ (now - cams[k].time ≤ T_GenCamMax ∨
        SensorDistance(j,cams[k].id,now) < SensDist)
    ∧ TimeToStop(j,now) * TTSFactor ≤ TimeToCollision(j
        ,now,cams[k]);
  {
    // If travelling in a similar direction
    if(abs(GetHeading(j,now) - cams[i].heading) ≤
        HeadDist)
    { // Recent CAM or close vehicle
      if(now - cams[i].time ≤ T_GenCamMax ∨
          SensorDistance(j,cams[i].id,now) < SensDist)
      {
        brake := brake ∨ TimeToStop(j,now) * TTSFactor
            ≤ TimeToCollision(j,now,cams[i]);
      }
    }
    i := i + 1;
  }
}
const Deceleration : real := 4.5; // m/s
function method TimeToStop(j: int, now: int): real {
  GetSpeed(j,now) as real / Deceleration
}
function method TimeToCollision(j: int,now: int,c: CAM):
     real
  requires now > c.time
{
  var timediff := now - c.time;
  var d := Distance(c.position, GetPosition(j,now));
  var d_early := d + GetSpeed(j,now) * -timediff;
  var d_now := d + c.speed * timediff;
  (dist_now - dist_early) as real / timediff as real
}
function method SensorDistance(j: int,k: int,now: int):int
  ensures SensorDistance(j,k,now) = Distance(
      GetPosition(j,now), GetPosition(k,now))
```

Fig. 8: Dafny implementation of the `receiveCAM` algorithm.

data available to the CAM generation algorithm. The content of the CAMs may not reflect the physical truth when considering this attacker, however, this verification is out of the scope of analysing the generation of CAMs.

5.2.3 Receiving CAMs

Previously [2], we investigated a simple non-repudiation property, for `receiveCAM`, by including the sender's id as input and verifying that the sender of the received CAM matched the provided id. This was an abstract model of the use of digital signatures to verify the authenticity of the sender. However, it relies on a priori knowledge of who sent a message and does not prove that a receiver can verify the authenticity of the sender. We build upon our previous Dafny implementation of `receiveCAM` [2] and, rather than address the simple non-repudiation property that appeared in our previous work, we focus on the safety of initiating the brakes

in response to a received CAM. The revised implementation that we present here considers a DoS threat.

Fig. 8 contains our implementation of the `receiveCAM` algorithm which takes as input the id of the vehicle receiving the CAM (`j`), the sequence of CAMs that have been sent (`cams`), the current time (`now`), a real number variable representing the time to stop (`TTSFactor`), `HeadDist` to determine if the vehicles are travelling in the same direction and `SensDist` to indicate the distance between vehicles as measured by the vehicle's sensors. This method outputs a boolean flag that indicates whether the vehicle should brake.

We verify the safety property for this method on lines 5–8 of Fig. 8 with respect to when the vehicle should brake. Specifically, we loop through the sequence of received `cams` (lines 12–30) and if one is found which indicates that the originating vehicle is travelling in the same direction as the current one (line 21), then we brake if the CAM was recent enough or the sensors indicate that we should (lines 23–26). This condition was inspired by the identification of a DoS threat so we ensure that, if the received CAM is out of date (potentially due to jamming) then we rely on the physical sensors to indicate when braking should occur. For completeness, we provide the specifications for the helper functions used by `receiveCAM` (Fig. 8, lines 31–45).

An open question in software verification is in ensuring that the verified models faithfully capture the fully implemented systems. This *reality gap* is difficult to traverse and will almost always exist when devising abstract models of program behaviour [11, 3]. Since real world implementations of CAM should obey the specification in [5], we chose it as our starting point for modelling the protocol. To address the reality gap, the properties that we have verified are examined at runtime via the Generate Runtime Monitors step in Fig. 1.

5.3 Runtime Verification (RV)

In this paper, we apply RV on two fronts: first, we synthesise runtime monitors from existing properties that were formulated during static verification (§5.1 and §5.2); and second, we generate new properties and the corresponding runtime monitors for checking DoS attacks in the CAM protocol. We use the CAM logs from executions of a simulation of cars exchanging CAMs as input to the monitors for verification.

First, we show the corresponding RV automata that are synthesised from our existing static verification properties. Note that we do not apply these automata to an existing runtime environment, since the execution of the system itself is out of scope for this work and we have already defined models of the system for use with these properties. When the system is deployed, it is necessary to attach these monitors (via instrumentation). In the second case, we design new properties to be used in RV for DoS attacks and we briefly report the details of the simulation that we used as an execution environment for DoS monitors. If monitors detect violations, mitigations may be included as shown in Fig. 2.

5.3.1 Monitor Synthesis from Static Verification

Where we have identified specific threats and carried out static verification we have a formal model of the system upon which we can assess the required property. However, any such formal model can be incomplete since it is an abstraction of the final system. Thus, there may be actual behaviours of the real system that do not fall within this model. An obvious route is to deploy a runtime monitor to check the required property on each *actual* execution. Typically, this is easy to achieve by deriving a monitor from the formal property (often as an automaton) assessed within static verification.

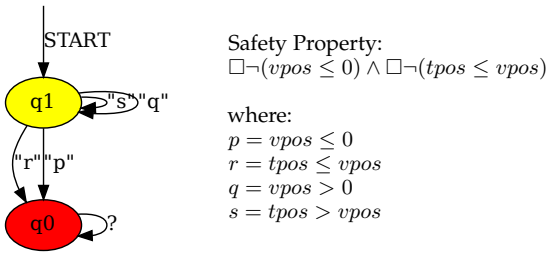

Fig. 9: Moore machine for safety property. Yellow states are inconclusive and red states are violation.

We use the LamaConv[5] tool, an existing Java library that converts temporal logic expressions into automata that can be used to generate runtime monitors. LamaConv is widely used to automatically generate runtime monitors from formal properties specified in LTL (for example [42, 43]). Since LTL does not support real-valued constraints, we manually convert our Promela property (presented in §5.1.1) by introducing symbols that abstract away the real-valued constraints. The automaton in Fig. 9 is automatically generated from the LTL safety property. Our monitor has an alphabet $\Sigma = \{p, q, r, s\}$, where events p and r are abstractions of the two comparisons in our original property, and the added q and s events are their respective opposites. This monitor remains inconclusive while observing q and s (these are the opposite of our bad states, making them good states that can be ignored), and returns a violation if either p or r are observed.

5.3.2 Runtime Verification for Denial of Service Attacks

Static verification of detailed runtime properties is often difficult. It requires a detailed model of the operating environment and the other actors. As this is rarely available, or feasible to verify, then an alternative is to deploy a runtime monitor to detect runtime properties. A simple example involves DoS attacks. Here we see either a vastly reduced response time (the effect) or a vastly increased number of messages arriving to a vehicle (the cause). We can deploy monitors to recognise these situations. For example, we might have expectations about message traffic and can encode this in a monitor. The monitor then *watches* the actual CAM traffic and flags a problem if the behaviour is significantly different to what is expected. The monitor used in this work simply recognises a divergent pattern of CAMs.

First, we introduce our simulation environment used to demonstrate the application of RV. VEINS, which simulates vehicle movement and is based on OMNET++, was utilised to conduct our simulations [44, 45]. In the simulation, a vehicle enters a single-lane road at the rate of one vehicle every 2.5–2.9s. Cars send messages every 100ms (`T_GenCam_Min`).

We investigated two parameters of the ECSDA digital signature scheme which can be used to verify the integrity

5. https://www.isp.uni-luebeck.de/lamaconv (Accessed 28/11/2023)

and authenticity of messages, and provide non-repudiation. Secp256k1 uses a $256\,\mathrm{bit}$ signature and Secp192k1 uses a $192\,\mathrm{bit}$ signature, additionally Secp256k1 provides greater security compared to Secp192k1 at higher space and computation costs. Each parameterisation takes a different amount of time to verify the signature (V_t). Secp256k1 takes $14\,\mathrm{ms}$ and Secp192k1 takes $8\,\mathrm{ms}$. These verification times were derived from profiling a vehicle's OBU [46], where the OBUs had an armv7l CPU with a clock speed of $792\,\mathrm{MHz}$. When a CAM is received it is put onto a queue. To emulate the verification time, the top of the queue is popped $V_t\,\mathrm{ms}$ after a CAM was added if the queue was empty, or $V_t\,\mathrm{ms}$ after the previous CAM was popped from the top of the queue.

We have performed experiments with maximums of 2 cars and 50 cars because we expect the number of cars to impact availability in terms of their ability to verify message authenticity. Due to the rate at which CAMs are sent and the time taken to verify a CAM signature, at some point a vehicle will be incapable of processing all of the messages that it receives in a $1\,\mathrm{s}$ window. For 2 cars, this should take a long time to occur or never happen. However, with 50 cars violations should be much more frequent.

Given this simulation environment, we want to verify at runtime an availability property from the receiver's perspective. That is, that the receiver can verify and process all of the messages that it receives. This will be violated when the time to verify the digital signatures of the received CAMs exceeds the processing speed that limits the number of messages that can be verified per second. More formally, given a message $queue$, we monitor that $queue \leq value$, and raise a warning whenever violation of this property is detected. The $value$ is determined by the number of CAMs per second that the vehicle's equipment can process and which signature scheme is being used. For Secp256k1 we monitor that $queue \leq 71$, i.e., the receiver can process up to 71 messages per second, while for Secp192k1 we monitor that $queue \leq 125$. Note that while this may be trivially verified statically (given a model of the environment), it is not possible to guarantee much in relation to DoS attacks, since this is dynamic behaviour that can only occur at runtime.

LamaConv was used to implement this monitor and its Java bindings [42] were used to run the monitor. A parser was used to feed logs obtained from the simulation into the monitor. The LTL property is simple, $\Box q$, which says it is always the case that we should observe q. The alphabet is defined by $\Sigma = \{p, q\}$, where p represents the event $queue > 71$ and q the event $queue \leq 71$ when using Sepc256k1 (125 for Sepc192k1). To test this monitor four log files from the simulation environment were used, one per signature scheme (Secp256k1 or Secp192k1) with fifty vehicles or two vehicles. One instance of the monitor was created per car. The results in Table 6 are as expected, no violations are detected with only two vehicles in the lane for either signature scheme. With fifty vehicles we observe violations in both signature schemes, with more in Secp256k1 due to its lower limit of messages that can be processed per second. The results also show that Secp256k1 had its first violation occur earlier in the simulation (at $30.2\,\mathrm{s}$) compared to Secp192k1 (at $48.4\,\mathrm{s}$) and that fewer vehicles had entered the simulation at the first violation for Secp256k1 (10) compared to Secp192k1 (16).

When violations are detected, monitors output error messages that contain the logged details of the event that caused the violation. For example: "*At time 30.28971822432 car with ID 0 violated the property, current queue of 72 is greater than the limit of 71. Message ID52 was sent by car ID 8*". This information can then be used to identify whether the violation was caused by an attack. Depending on the results of the monitors and static verification steps, mitigations may be required if the risk is not sufficiently managed.

Signature	Cars	Total Events	Violations (count and %)	
Secp256k1	50	286 337	271 214	94.7%
	2	3595	0	0 %
Secp192k1	50	290 343	195 614	67.3%
	2	4025	0	0 %

TABLE 6: RV results. Events observed is the total number of events observed by all monitors. Violations represent the total number of violations detected in the observed events.

6 Mitigations

Our verification revealed that the system was indeed vulnerable to attacks. In this section we discuss potential mitigations and associated implications for verification. This process is indicated in the bottom right of Fig. 2 when we assess Is risk managed for this use case and verification?

6.1 Mitigating Spoofing

Having identified that there is a potential issue if the vehicles in the platoon rely on CAMs alone for safety we are able to adapt our algorithms. We assume that each vehicle is fitted with a LIDAR or similar sensor that informs it of the distance to the vehicle in front of it. We adapted the protocol used by the middle and tail vehicles as follows:

- If the vehicle in front is closer than 10 units then reduce speed to 0.
- If the vehicle in front is further than 25 units away then increase speed to 5.
- Otherwise behave as in previous protocol.

We modified our Promela model to use this protocol and, with the attacker as in Fig. 4, they were unable to cause a crash by Spoofing as described in §5.1. The position information from the LIDAR detects imminent collisions and the vehicle slows down to safely avoid crashing. It is important to note that we did some basic calibration to identify the safe distances required for the initial positions of the vehicles and appropriate LIDAR thresholds. In more realistic models these would be set automatically. Next, we discuss the benefit of using both sensor data and CAMs.

6.2 Using CAMs and Sensor Data

There are two implementations to consider when vehicles perform local decision making. These are using: (i) only information received from CAMs and (ii) information received from sensors and CAMs. When performing decision making solely based on information received in CAMs, the digital signature needs to be verified to ensure the authenticity of the message. Typically, if a signature is invalid, the CAM will be rejected [5, §6.2.2.1]. However, information in CAMs may

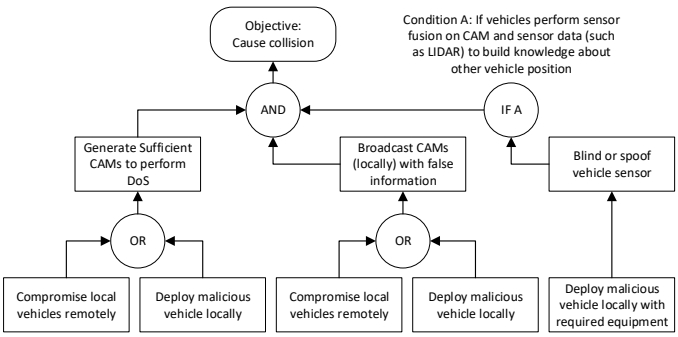

Fig. 10: An attack tree describing the multiple attacks needed to cause a collision by spoofing CAMs.

```
1  method relaxedSendCAM(T_CheckCamGen: int, T_GenCam_DCC:
       int, j: int, start: int, max_msgs: int)
2    returns (msgs: seq<CAM>, now: int, c_T_CheckCamGen: int,
         c_T_GenCam_DCC: int)
3    requires start ≥ 0 ∧ max_msgs ≥ 0;
4    requires 0 < N_GenCamDefault ≤ N_GenCamMax;
5    ensures c_T_GenCam_DCC * |msgs| ≤ (now - start) ≤
       T_GenCamMax * |msgs|;
6    ensures |msgs| ≥ 2 ⟹ ∀ i: int • 1 ≤ i < |msgs| ⟹
       c_T_GenCam_DCC ≤ (msgs[i].time - msgs[i-1].time
       ) ≤ T_GenCamMax;
7    ensures |msgs| = max_msgs;
8  {
9   c_T_CheckCamGen := clamp(T_CheckCamGen,1,T_GenCamMin);
10  c_T_GenCam_DCC := clamp(T_GenCam_DCC,T_GenCamMin,
      T_GenCamMax);
11  msgs, now := sendCAM(c_T_CheckCamGen,c_T_GenCam_DCC,j,
      start,max_msgs);
12 }
13 function method clamp(x: int, min: int, max: int):int {
14   if x < min then min else
15   if x > max then max else x
16 }
```

Fig. 11: Dafny implementation of the `relaxedSendCAM` algorithm. This ensures availability if an adversary manipulates the inputs `T_CheckCamGen` and `T_GenCam_DCC`.

potentially be acted on if the vehicle's resources are fully consumed and will not be able to verify the signature of all CAMs in a timely manner. When vehicles perform decision making that considers additional data sources (e.g., LIDAR or camera data), the input to these sensors will also need to be spoofed or blinded for the attack to succeed.

Thus, in order to successfully cause vehicles to make decisions under this threat model then an adversary would either need to (i) generate sufficient CAMs that could not all be verified in a timely manner and (ii) blind or Spoof input to sensors to prevent them from detecting incorrect information in unverified CAMs and (iii) generate a CAM impersonating another vehicle with incorrect information within it (see Fig. 10). While the impact to a vehicle would be high (violation of safety), the likelihood of the attack is low and can be mitigated further by considering sensor fusion of multiple sources of input for autonomous decision making.

6.3 Relaxing `sendCAM` Pre-conditions

Other aspects of the CAM generation algorithm could be manipulated by an adversary. For example, the input parameters to `sendCAM` could be altered to be outside of the bounds required. To ensure availability under this threat model, we provide a `relaxedSendCAM` method that constrains the values of the incorrect parameters such that the parameters are valid. Fig. 11 shows `T_CheckCamGen` and `T_GenCam_DCC` constrained by `clamp` which returns a suitable value inside the range of [`min`, `max`]. The resulting implementation of `relaxedSendCAM` does not require any pre-conditions on `T_CheckCamGen` and `T_GenCam_DCC`.

An implementation of `relaxedSendCAM` is useful if adversaries could manipulate the inputs to the algorithm. Typically an adversary would gain access to the filesystem of the vehicle which could either be via a local (e.g., physical connection) or remote attack [47]. Depending on how the configuration of `relaxedSendCAM` is managed, attackers may need to escalate privileges to access the filesystem. For example, exploiting remotely may only provide access to a low privilege account that must be escalated to a higher privilege account to modify the CAM configuration file. Such an attack could be performed to prevent the vehicle from sending CAMs, but with `relaxedSendCAM` the impact of this attack would be mitigated as CAMs would still be generated within the required time bounds.

6.4 Availability Against DoS

The availability property that we have verified in the `sendCAM` method only ensures a vehicle is broadcasting CAMs at a suitable rate. The availability property from the perspective of receiving messages has been monitored via runtime verification. As a vehicle will not be able to verify all received CAMs in a timely manner in all circumstances, an availability property where a vehicle processes received messages would not be able to be maintained. Instead, messages may need to be prioritised according to some metric (such as allocating messages into zones and calculating a priority by distance, acceleration and velocity [48]). When the system enters this state, other tools such as PRISM [49] and STORM [50] for probabilistic verification may be useful in analysing the efficacy of such prioritisation techniques.

7 DISCUSSION

STRIDE threats are not equipped with a formal, logical definition. Such a definition would be especially useful for formal methods and the lack of one makes it difficult to accurately formalise properties related to the threats in the Formalise Threats step in Fig. 2. However, this methodology provides a series of steps to guide system designers or developers to focus their formalisation efforts on the most pertinent threats for a specific system/use case. By systematically applying our methodology to a large number of case studies we may be able to extract more general definitions of these threats that are useful from a formal methods perspective in future work. For example, formalisation of threats could aid with better automation of threat identification and modelling techniques.

In this paper, we leveraged multiple verification approaches. This is becoming more common as systems increase in size and complexity to support the verification of different aspects using appropriate tools/techniques [11]. A limitation is in ensuring that multiple models of the same system are indeed representative of the system and are consistent with one another. This is important for both static and dynamic verification. Models used for static verification should be consistent with one another and be representative of the

implemented system. The models and properties verified at runtime should also be consistent with the static verification models. This work relies on expert opinion to ensure consistency between these artefacts and this is a common approach in assurance [51]. However, more systematic ways to compare and translate between models would be useful.

Notably, there is no end point in our methodology (Fig. 2) because the inclusion of runtime monitors allows the system to be continually monitored during execution. This is important for autonomous systems whose operating environment may be dynamic and unpredictable. Further, attackers are continually devising sophisticated attacks and our runtime monitors are used to identify potential attacks.

This paper focused on how violation of security properties impacts safety. We did not consider how violations of safety properties impact security. Since the safety process often involves a risk/hazard analysis process it might be worth exploring how these analyses could be linked.

8 Conclusions and Future Work

This paper builds on our methodology from [1] and presents a more detailed case study than originally provided in [2]. We demonstrate how cyber security threat analysis techniques can guide formal methods practitioners in verifying security properties, especially those that impact safety.

We carried out a STRIDE threat analysis of the CAM protocol for sending and receiving messages between vehicles and identified Spoofing and DoS attacks that may occur. We modelled Spoofing by specifying the behaviour of an attacker in our Promela model. DoS was considered statically via an availability property in the Dafny implementation of the algorithm for sending CAMs. DoS when receiving CAMs was also addressed by runtime monitoring.

Previously, we discussed the need for the use of integrated formal methods in the robotics domain and the example that we present here is no different [11]. By modelling the system at different levels of abstraction; system-level in Promela/SPIN and algorithm-level in Dafny, we were able to investigate and to verify properties related to STRIDE threat analysis. In particular, model-checking with Promela/SPIN is useful for examining high-level temporal properties. Conversely, theorem proving with Dafny allowed us to examine properties of our CAM protocol implementation.

Our use of Promela/SPIN and Dafny has been motivated by our familiarity with these tools and it is certainly the case that other formal methods may have been a better choice for our study. We intend to investigate this further in future work. Future analysis of CAM with various tools will likely provide a better understanding of which STRIDE properties should be checked using different kinds of formal methods. Along with (static) verification of the security properties, we showed how runtime monitors could be incorporated into our methodology to identify assumptions that have been violated or for conditions that are difficult to verify statically.

Importantly, although it could be useful, the individual formal analyses do not need to be combined as in holistic/compositional formal approaches [52–54]. Instead, formal methods focus security analysis on to specific areas/scenarios highlighted by cyber security analysis as being of *high risk*. Future work could involve proving that the independent formal models do, in fact, capture the same system. Additionally we focus on *formal* methods combined with security analyses. We could also apply non-formal verification techniques like simulation-based testing and real vehicle experiments to complement formal verification and further help provide assurances against security threats. For example [55] uses different types of verification that inform each other, to consider safety and reliability for a collaborative manufacturing task for a robot assistant. This work is a first step toward a detailed methodology of how STRIDE properties should be treated in formal verification.

Data Statement

Artefacts can be found at https://github.com/autonomy-and-verification/security-minded-verification.

Acknowledgments

Work supported by the Royal Academy of Engineering and UKRI via the Trustworthy Autonomous Systems Nodes on *Security* [EP/V026763/1] and *Verifiability* [EP/V026801/2] and the *FAIR-SPACE Hub* [EP/R026092].

References

[1] C. Maple, M. Bradbury, H. Yuan, M. Farrell, C. Dixon, M. Fisher, and U. I. Atmaca, "Security-minded Verification of Space Systems," in *IEEE Aeroconf*, 2020, pp. 1–13.

[2] M. Farrell, M. Bradbury, M. Fisher, L. A. Dennis, C. Dixon, H. Yuan, and C. Maple, "Using Threat Analysis Techniques to Guide Formal Verification: A Case Study of Cooperative Awareness Messages," in *Software Engineering and Formal Methods*. Springer, 2019, pp. 471–490.

[3] M. Luckcuck, M. Farrell, L. A. Dennis, C. Dixon, and M. Fisher, "Formal Specification and Verification of Autonomous Robotic Systems: A Survey," *ACM Computing Surveys*, vol. 52, no. 5, 2019.

[4] J. Santa, F. Pereñíguez, A. Moragón, and A. F. Skarmeta, "Vehicle-to-Infrastructure Messaging Proposal based on CAM/DENM Specifications," in *IFIP Wireless Days*. IEEE, 2013, pp. 1–7.

[5] European Telecommunications Standards Institute, "Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service," Tech. Rep. ETSI EN 302 637-2, 2019, V1.4.1 (2019-04).

[6] W. Stallings, L. Brown, M. D. Bauer, and A. K. Bhattacharjee, *Computer Security: Principles and Practice*. Pearson, 2012.

[7] L. Kohnfelder and P. Garg, "The Threats to Our Products," Apr. 1999, Accessed: 2018-12-10. [Online]. Available: https://tinyurl.com/2p97xav3

[8] R. S. Ross, "Guide for Conducting Risk Assessments," National Institute of Standards and Technology, Tech. Rep., Sep. 2012, SP 800-30 Rev. 1.

[9] G. J. Holzmann, *The Spin Model Checker: Primer and Reference Manual*. Addison-Wesley, 2003.

[10] K. R. M. Leino, "Dafny: An Automatic Program Verifier for Functional Correctness," in *Logic for Programming*

Artificial Intelligence and Reasoning. Springer, 2010, pp. 348–370.

[11] M. Farrell, M. Luckcuck, and M. Fisher, "Robotics and Integrated Formal Methods: Necessity meets Opportunity," in *Integrated Formal Methods*. Springer, 2018, pp. 161–171.

[12] M. Fisher, *An Introduction to Practical Formal Methods Using Temporal Logic*. Wiley, 2011.

[13] M. Barnett, B.-Y. E. Chang, R. DeLine, B. Jacobs, and K. R. M. Leino, "Boogie: A Modular Reusable Verifier for Object-Oriented Programs," in *Formal Methods for Components and Objects*. Springer, 2005, pp. 364–387.

[14] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

[15] K. Havelund, G. Reger, and G. Rosu, "Runtime Verification Past Experiences and Future Projections," in *Computing and Software Science - State of the Art and Perspectives*. Springer, 2019, pp. 532–562.

[16] C. Snook, T. S. Hoang, and M. Butler, "Analysing Security Protocols using Refinement in iUML-B," in *NASA Formal Methods*. Springer, 2017, pp. 84–98.

[17] J. Whitefield, L. Chen, F. Kargl, A. Paverd, S. Schneider, H. Treharne, and S. Wesemeyer, "Formal Analysis of V2X Revocation Protocols," in *Security and Trust Management*. Springer, 2017, pp. 147–163.

[18] G. Vanspauwen and B. Jacobs, "Verifying Protocol Implementations by Augmenting existing Cryptographic Libraries with Specifications," in *Software Engineering and Formal Methods*. Springer, 2015, pp. 53–68.

[19] L. Huang and E.-Y. Kang, "Formal Verification of Safety & Security Related Timing Constraints for a Cooperative Automotive System," in *Fundamental Approaches to Software Engineering*. Springer, 2019, pp. 210–227.

[20] S. Schneider and R. Delicata, "Verifying security protocols: An application of CSP," in *Communicating Sequential Processes. The First 25 Years*. Springer, 2005, pp. 243–263.

[21] S. Schneider, "Formal Analysis of a Non-repudiation Protocol," in *Computer Security Foundations Workshop*, 1998, pp. 54–65.

[22] ——, "Verifying authentication protocols in CSP," *IEEE Trans. Software Engineering*, vol. 24, no. 9, pp. 741–758, 1998.

[23] F. Al-Shareefi, A. Lisitsa, and C. Dixon, "Clarification of Ambiguity for the Simple Authentication and Security Layer," in *Abstract State Machines, Alloy, B, TLA, VDM, and Z*. Springer, 2018, pp. 189–203.

[24] M. Kamali, S. Linker, and M. Fisher, "Modular Verification of Vehicle Platooning with Respect to Decisions, Space and Time," in *Formal Techniques for Safety-Critical Systems*. Springer, 2018, pp. 18–36.

[25] T. Kulik, P. W. V. Tran-Jørgensen, J. Boudjadar, and C. Schultz, "A Framework for Threat-Driven Cyber Security Verification of IoT Systems," in *Software Testing, Verification and Validation Workshops*, 2018, pp. 89–97.

[26] G. Pedroza, "Towards Safety and Security Co-engineering - Challenging Aspects for a Consistent Intertwining," in *Security and Safety Interplay of Intelligent Software Systems*. Springer, 2019, pp. 3–16.

[27] G. Pedroza, L. Apvrille, and D. Knorreck, "AVATAR: A SysML Environment for the Formal Verification of Safety and Security Properties," in *New Technologies of Distributed Systems*, 2011, pp. 1–10.

[28] F. Aarts, J. De Ruiter, and E. Poll, "Formal Models of Bank Cards for Free," in *Software Testing, Verification and Validation Workshops*. IEEE, 2013, pp. 461–468.

[29] J. De Ruiter and E. Poll, "Formal Analysis of the EMV Protocol Suite," in *Theory of Security and Applications*. Springer, 2011, pp. 113–129.

[30] J. Sheffield and J. Lemétayer, "Factors associated with the software development agility of successful projects," *International Journal of Project Management*, vol. 31, no. 3, pp. 459–472, 2013.

[31] European Telecommunications Standards Institute, "Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Definitions," Tech. Rep. ETSI TR 102 638, 2009, V1.1.1 (2009-06).

[32] C. Johnson, L. Badger, D. Waltermire, J. Snyder, and C. Skorupka, "Guide to Cyber Threat Information Sharing," National Institute of Standards and Technology, NIST Publication 800-150, 2016.

[33] European Telecommunications Standards Institute, "Intelligent Transport Systems (ITS); Security; Threat, Vulnerability and Risk Analysis (TVRA)," Tech. Rep. ETSI TR 102 893, 2017, V1.2.1 (2017-03).

[34] M. Bradbury, C. Maple, H. Yuan, U. I. Atmaca, and S. Cannizzaro, "Identifying Attack Surfaces in the Evolving Space Industry Using Reference Architectures," in *IEEE Aeroconf*, 7–14 March 2020.

[35] Q. Do, B. Martini, and K.-K. R. Choo, "The role of the adversary model in applied security research," *Computers & Security*, vol. 81, pp. 156–181, 2019.

[36] J. McDermott and C. Fox, "Using Abuse Case Models for Security Requirements Analysis," in *Computer Security Applications*. IEEE, 1999, pp. 55–64.

[37] I. Leontiadis, G. Marfia, D. Mack, G. Pau, C. Mascolo, and M. Gerla, "On the Effectiveness of an Opportunistic Traffic Management System for Vehicular Networks," *IEEE Trans. Intelligent Transportation Systems*, vol. 12, no. 4, pp. 1537–1548, 2011.

[38] B. Langenstein, R. Vogt, and M. Ullmann, "The Use of Formal Methods for Trusted Digital Signature Devices," in *Florida Artificial Intelligence Research Society Conf.* AAAI Press, 2000, pp. 336–340.

[39] European Telecommunications Standards Institute, "Intelligent Transport Systems (ITS); Security; Trust and Privacy Management," Tech. Rep. ETSI TS 102 941, 2019, V1.3.1 (2019-02).

[40] N. Beckmann and M. Potkonjak, "Hardware-Based Public-Key Cryptography with Public Physically Unclonable Functions," in *Information Hiding*. Springer, 2009, pp. 206–220.

[41] A. Pnueli, "The Temporal Logic of Programs," in *Foundations of Computer Science*. IEEE, 1977, pp. 46–57.

[42] A. Ferrando and R. C. Cardoso, "Towards Partial Monitoring: It is Always too Soon to Give Up," in *Formal Methods for Autonomous Systems*. Open Publishing Association, 2021, pp. 38–53.

[43] H. Kallwies, M. Leucker, and C. Sánchez, "General Anticipatory Monitoring for Temporal Logics on Finite Traces," in *Runtime Verification*. Springer, 2023.

[44] C. Sommer, R. German, and F. Dressler, "Bidirectionally coupled network and road traffic simulation for improved IVC analysis," *IEEE Trans. Mobile Computing*, vol. 10, no. 1, pp. 3–15, 2010.
[45] A. Varga and R. Hornig, "An overview of the OMNeT++ simulation environment," in *Simulation Tools and Techniques for Communications, Networks and Systems*, 2008.
[46] C. Maple, M. Bradbury, M. Elsden, H. Cruickshank, H. Yuan, C. Gu, and P. Asuquo, "IoT Transport and Mobility Demonstrator: Cyber Security Testing on National Infrastructure," Univ. Warwick, UK, Tech. Rep., 2019.
[47] S. Nie, L. Liu, and Y. Du, "Free-fall: Hacking Tesla From Wireless To CAN Bus," in *Black Hat USA*, 2017, https://tinyurl.com/2ckbux4c.
[48] S. Banani and S. Gordon, "Selecting Basic Safety Messages to Verify in VANETs using Zone Priority," in *Asia-Pacific Communication Conf.* IEEE, 2014, pp. 423–428.
[49] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker, "PRISM: A Tool for Automatic Verification of Probabilistic Systems," in *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2006, pp. 441–444.
[50] C. Dehnert, S. Junges, J.-P. Katoen, and M. Volk, "A STORM is Coming: A Modern Probabilistic Model Checker," in *Computer Aided Verification*. Springer, 2017, pp. 592–600.
[51] H. Bourbouh, M. Farrell, A. Mavridou, I. Sljivo, G. Brat, L. A. Dennis, and M. Fisher, "Integrating formal verification and assurance: an inspection rover case study," in *NASA Formal Methods*. Springer, 2021, pp. 53–71.
[52] C. B. Jones, "Tentative Steps Toward a Development Method for Interfering Programs," *ACM Trans. Programming Languages and Systems*, vol. 5, no. 4, pp. 596–619, 1983.
[53] R.-J. Back, "A Calculus of Refinements for Program Derivations," *Acta Informatica*, vol. 25, no. 6, pp. 593–624, 1988.
[54] C. Morgan, K. Robinson, and P. Gardiner, *On the Refinement Calculus*. Springer, 1988.
[55] M. Webster, D. Western, D. Araiza-Illan, C. Dixon, K. Eder, M. Fisher, and A. G. Pipe, "A Corroborative Approach to Verification and Validation of Human–Robot Teams," *Robotics Research*, vol. 39, pp. 73–99, 2020.

Marie Farrell is a Royal Academy of Engineering Research Fellow in the Department of Computer Science at the University of Manchester. Her research focuses on using and combining formal methods to reason about and provide certification evidence for robotic systems deployed in hazardous environments. She is secretary for the working group developing the IEEE P7009 Standard on Fail-Safe Design for Autonomous Systems.

Matthew Bradbury is Lecturer in Cyber Security at Lancaster University. He received his PhD in Computer Science from the Department of Computer Science at the University of Warwick in 2018. His research interests include the security, privacy and trustworthiness of resource-constrained and distributed Cyber Physical Systems, including wireless sensor networks, transportation systems and space systems.

Rafael C. Cardoso is a Lecturer at the University of Aberdeen. He obtained his MSc (2014) and PhD (2018) in Computer Science from PUCRS, and BSc in Computer Science from Centro Universitário Franciscano (2012). His research interests include multi-agent systems and multi-agent planning, as well as integrating formal verification in software development.

Michael Fisher is Royal Academy of Engineering Chair in Emerging Technologies in the Department of Computer Science at the University of Manchester. He co-chairs the IEEE Technical Committee on the Verification of Autonomous Systems, and he sits on standards boards related to *sustainable robotics*, *ethical robot deployment*, and *fail-safe design of autonomous systems*.

Louise A. Dennis is a Senior Lecturer at the University of Manchester where she leads the Autonomy and Verification Group. Her research focuses on the development and verification of autonomous systems, with specific interest in issues related to transparency and ethics. She is conference coordinator for the ACM Special Interest Group in AI.

Clare Dixon is a Professor of Computer Science in the Department of Computer Science at the University of Manchester. Prior to joining Manchester she was a Professor of Computer Science in the Department of Computer Science at the University of Liverpool, working as an academic at Liverpool from 2001-2020. Her research interests include formal verification and temporal and modal theorem-proving techniques, in particular applied to robotics and autonomous systems.

Al Tariq Sheik is a Doctoral Researcher at WMG, University of Warwick, and concurrently serves as a Research Assistant at the Alan Turing Institute. His research focusses on threat modelling and risk assessment, and adaptive security within cyber-physical systems and organisational infrastructures.

Hu Yuan is a Lecturer in Cyber Security at Kingston University. He received his PhD from the University of Warwick (2016). His research focuses on the security and privacy aspects of IoT, including internet of bio-nano things, vehicular communication networks, user behaviours identification and space systems. He was the leading researcher for the IoT Transport and Mobility Demonstrator.

Carsten Maple is Professor of Cyber Systems Engineering in WMG at the University of Warwick, where he is the Director of Research in Cyber Security. He is Principal Investigator at the EPSRC/GCHQ Academic Centre of Excellence in Cyber Security Research, leads various projects on the security of CAVs and is a fellow of the Alan Turing Institute and member of the ENISA CARSEC Expert Group.