Automating the Assembly of Security Assurance Case Fragments

Baoluo Meng(✉), Saswata Paul, Abha Moitra, Kit Siu, and Michael Durling

General Electric Research, Niskayuna, NY, USA
{baoluo.meng,saswata.paul,moitra,siu,durling}@ge.com

Abstract. This paper presents an approach and tools for automatic generation of security assurance case fragments using patterns for arguing the security of cyber physical systems. The fragments are generated using augmented Goal Structuring Notation (GSN) and can succinctly convey a system's resilience to cyber-threats specified in MITRE's Common Attack Pattern Enumeration and Classification (CAPEC). The GSN schema has been augmented with additional metadata that can be used for visually tracing back to component-level CAPEC threats from higher-level cyber security claims, enabling designers to easily locate flaws in a model when one or more claims cannot be substantiated. An implementation of the approach as a part of the Verification Evidence and Resilient Design in Anticipation of Cybersecurity Threats (VERDICT) toolchain has also been demonstrated along with a case study of a package delivery drone.

Keywords: Security assurance cases · Assurance case patterns · GSN · Security analysis of system architecture · Attack-defense tree · MITRE's CAPEC threats and NIST-800-53 controls

1 Introduction

The failure of safety-critical cyber physical systems can be catastrophic to life, property, or the environment [33]. Therefore, it is imperative that both software and hardware components are subjected to rigorous testing and certification procedures before being approved for use in safety-critical domains such as aviation, medicine, and automotive. However, with increasing complexity of systems, it becomes difficult to accurately and efficiently argue about system guarantees with respect to critical properties. One widely used approach for conveying system guarantees is the construction of *assurance cases* [5]. An assurance case is a structured argument that a system satisfies some desired safety, security, or reliability properties [4]. It is used to convey a compelling and comprehensive case for system guarantees to stakeholders, developers, engineers, and certifiers [20].

There are two main approaches for developing assurance cases – *process based*, which argues that a system has been developed adhering to certain process objectives such as RTCA-DO:178C [30], and *product based*, which argues that a system satisfies certain properties [21]. Several standards such as *Claim, Argument*

I. Habli et al. (Eds.): SAFECOMP 2021, LNCS 12852, pp. 101–114, 2021.
https://doi.org/10.1007/978-3-030-83903-1_7

and Evidence (CAE) [7], *Goal Structuring Notation* (GSN) [23], and *Structured Assurance Case Metamodel* (SACM) [35] exist for representing assurance cases. Traditionally, assurance cases have been used for arguing about the safety of systems. However, there has been recent interest in using them to argue about the security aspects of complex systems [9,29]. Although the fundamental concept of safety and security cases is similar, *i.e.*, using evidences to argue the validity of some claims, according to [4], there are some differences between the two. Security deals with the presence of an intelligent adversary, whereas safety analysis involves assigning probabilities to basic events (e.g., sensor failure), but the same is not possible for adversarial actions; and security-critical systems have to dynamically adapt to adversarial actions.

Model-based development uses tools and techniques to design and analyze domain-specific models of systems [6]. The *Verification Evidence and Resilient Design in Anticipation of Cybersecurity Threats* (VERDICT)[1] is a toolchain for incorporating cyber security resiliency analysis and recommendations in the system design process that are automated, scalable, provide rich feedback, specify trade-offs, and are easy to use by system architects [26,32]. It is available as a plugin for the Open Source AADL Tool Environment (OSATE) [16–18]. The *Model-Based Architectural Analysis* (MBAA) functionality of VERDICT analyzes the security of a system at the architectural level and mainly consists of two tools: STEM (Security Threat Evaluation and Mitigation) [28] and Soteria++ [31]. STEM identifies threats from the MITRE Corporation's *Common Attack Pattern Enumeration and Classification* (CAPEC) [1] and selects defenses from the National Institute of Standards and Technology's (NIST-800-53) *Security and Privacy Controls* [2]. Soteria++ constructs attack scenarios and highlights areas of the architecture where control measures are either missing or inappropriate, or insufficient rigor has been applied compared to the level of severity of the outcome. It generates attack-defense trees and analyzes them to determine cutsets and the likelihood of successful attacks of top-level events.

The conventional approach for constructing assurance case fragments is informal and manual, making them prone to errors, expensive and time consuming to design, and difficult to manage and evaluate for non-trivial systems [13]. Moreover, the informal nature of arguments is susceptible to logical flaws [5], making them unsuitable for certifying critical applications. This calls for the need of an approach that can holistically combine tools and techniques from formal model-based analysis to automatically generate assurance case fragments, for both safety and security, to aid in system certification. Contributions of this paper towards this end are:

- It augments the traditional GSN schema with metadata to present additional information that can be beneficial for system designers and certifiers.
- It presents security assurance case patterns for automatic instantiation of structured arguments using the augmented GSN schema. The patterns are designed to use domain-specific claims supported by VERDICT and domain-

[1] For more details, visit https://github.com/ge-high-assurance/VERDICT.

specific evidences generated by SOTERIA++ for arguing about both security and safety.
- It presents an approach for using the evidences generated by MBAA to construct security case fragments by considering component-level CAPEC threats and NIST-800-53 defenses.
- It demonstrates the effectiveness of the approach by presenting a case study using the model of an unmanned package delivery drone.

The rest of the paper is structured as follows: Sect. 2 describes the augmented GSN schema, security assurance case patterns, and the construction of security case fragments, by using evidences generated by MBAA; Sect. 3 presents a case study using the model of a package delivery drone; Sect. 4 discusses the key findings; Sect. 5 presents related work on assurance cases; and Sect. 6 ends the paper with a conclusion and future directions of work.

2 Contribution

2.1 Augmented Goal Structuring Notation (GSN)

The GSN schema has been widely used in safety case development. It provides a graphical representation of assurance case arguments by using principal elements—*goals*, *strategies*, *contexts*, *assumptions*, *justifications*, and *solutions*, which are arranged as nodes in a dependency graph. This work augments the traditional GSN to convey additional information to designers and certifiers. We incorporate additional node metadata to provide enhanced visual aid for flaw detection and for generating interactive assurance case fragments with tooltips and clickable links. The metadata also connects the assurance case fragments to the artifacts and evidences generated by VERDICT's MBAA functionality, creating a coherent connection between formal analysis and structured assurance arguments. This augmented GSN schema may also be useful in incorporating defeaters [8] where defeaters capture doubts and objections. The identification of defeaters can surface gaps in an assurance case; and assessment/evaluation of defeaters can strengthen an assurance case.

Traditionally, assurance cases have been designed to substantiate claims using evidences. However, in VERDICT, sometimes the evidences generated may show that a claim cannot be substantiated. Under such circumstances, a traditional GSN fragment cannot be created, but the information about the failure of a claim can still be useful to designers. This information can be used to generate "*incomplete*" GSN fragments which do not argue about the correctness of claims, but provide enhanced visual aid for easily detecting flaws in a model. To this end, we augment the GSN schema by specifying colors for the goal and strategy nodes. Below are the rules determining the color of nodes:

- If a solution fails to substantiate its parent node, then it will be colored red. Otherwise, it is colored green.
- If a goal or strategy node has at least one supporting red node, then it will be colored red. Otherwise, it is colored green.

2.2 Security Assurance Case Patterns

Table 1. A part of the assurance case pattern library for VERDICT.

GSN class	Predefined patterns
Goal	{} "is secure"
	{} "has been mitigated"
Strategy	"Argument: By validity of sub-goals"
	"Argument: By SOTERIA++ analysis of attack-defense trees"
	"Argument: All threats mitigated"
Context	{} "Properties"
	"All applicable threats are identified"
	"Acceptable likelihood ="{}, "Computed likelihood =" {}
	"Acceptable probability =" {}, "Computed probability =" {}
	"A condition and a target probability"
	"A condition and a target likelihood"
Solution	"SOTERIA++ minimum cutset for" {}
	"Evidence that" {} "is secure"

Assurance case patterns [22] are predefined templates that can be instantiated for constructing assurance cases. Patterns restrict the verbiage that can be used for the construction of assurance cases while being flexible enough for expressing structured arguments for a variety of systems. The use of patterns allows the development of consistent assurance case fragments and provides a formal structure for assurance case construction. They are composed of domain-specific constructs that depend on factors such as domain safety and security concerns, type of claims and evidences that can be used, and type of arguments [12].

In order to automatically generate assurance case fragments from the VERDICT toolchain, a library of patterns has been created. These patterns can be used to express structured arguments for claiming cyber-resilience by using evidences generated by SOTERIA++. The patterns allow the implementation to automatically collect information from an *annex*, which is a domain-specific language extension to AADL, and instantiate structured assurance case fragments in the augmented GSN schema. Table 1 shows a part of our assurance case pattern library for the different classes of GSN nodes. The strings inside "" are predefined and the patterns can be used to instantiate the arguments by automatically assigning parameters in place of the curly brackets {}. Using the patterns, it is possible to create claims like "actuation is secure" or "CAPEC-390 has been mitigated" to declare that the actuation component has been secured and that the threat of CAPEC-390 has been mitigated by the implemented defenses respectively. The patterns are specific to the VERDICT toolchain as they can only be used to express statements that make sense in the context

of the constructs that are used in VERDICT. *E.g.*, the *verdict* annex supports only two classes of requirements related to security –*mission requirements* and *cyber requirements*–so the patterns can only be used to instantiate requirements belonging to any of these two classes and not arbitrary ones.

2.3 Model-Based Architecture Analysis in VERDICT

One of the fundamental aspects of security, as discussed in Sect. 1, is that it is concerned with external adversarial actions that need to be mitigated. In this section, we will describe the model-based architecture analysis (MBAA) functionality of VERDICT, which can be leveraged to identify cyber vulnerabilities and mitigations, to calculate the likelihood of successful attacks, and ultimately to construct security cases.

Input to Model-Based Architecture Analysis. The input to MBAA is an AADL architecture model annotated with meta-level properties, defense properties, cyber relations and requirements, and mission requirements. Examples of meta-level properties are *componentType* and *connectionType*. Defense properties describe the rigor to implement a defense, which are enumerated type [0; 3; 5; 7; 9]. The implementation rigor is also known as *design assurance level* (DAL), with 0 being the lowest and 9 being the highest rigor. Relations and requirements are written in the *verdict* annex. A mission requirement describes a mission scenario for the system to fulfill, such as delivering a package to an intended location, which is supported by a combination of cyber and safety requirements. The success of mission requirements depends on the success of all supporting cyber and safety requirements. This paper will focus on cyber relations and requirements for security assurance cases. An example of a cyber relation and requirement is illustrated in Fig. 1.

Each cyber requirement is associated with a level of severity (Catastrophic, Hazardous, Major, Minor, and No Effect) corresponding to a quantitative acceptable level of risk (1e−9, 1e−7, 1e−5, 1e−3, and 1e−0, respectively) guided by the standard on Airworthiness Security Methods and Considerations DO-356A.

```
CyberRel "delivery_status_I" = delivery_cmd:I => delivery_status:I;
CyberReq {
    id = "CyberReq01"
    description = "The drone shall be resilient to loss
                   of ability to deliver a package to the
                   appropriate consumer location"
    condition = actuation_out:I or actuation_out:A
                or delivery_status:I or delivery_status:A
    cia = I
    severity = Hazardous
};
```

Fig. 1. A cyber relation and requirement specified in the *verdict* annex.

Note that each exponent of quantitative acceptable level of risk corresponds to a negative DAL value. Each cyber requirement is also tied to a security aspect of the overall system: *Confidentiality (C)*, *Integrity (I)* and *Availability (A)* listed in the "*cia*" field the cyber requirement. The success or failure of the cyber requirement relies on the logical "*condition*", which describes relevant CIAs of outputs of the system, and will be used in attack-defense tree analysis in SOTERIA++. Cyber relations describe how various possible threats associated with confidentiality, integrity, or availability propagate through components. The cyber relation in Fig. 1 tells that any threats affecting the integrity of the input *delivery_cmd* of the component will also affect integrity of the output *delivery_status*.

Analyzing the Security of System Architectures. The analysis of the security of system architectures replies on two components of VERDICT: Security Threat Evaluation and Mitigation (STEM) and SOTERIA++, which are described below.

Security Threat Evaluation and Mitigation (STEM). Security Threat Evaluation and Mitigation (STEM) [28] is a SADL-based semantic model with a set of rules to identify vulnerabilities and suggest defenses. SADL (Semantic Application Design Language) [10] is an English-like language based on Web Ontology Language (OWL) for building semantic models and authoring rules. STEM takes system architectural information annotated with properties as input, identifies possible threats from MITRE's CAPEC, and suggests mitigations from NIST-800-53 Security and Privacy Controls. There are a total of 61 Meta Attack Pattern CAPECs, but not all of them are relevant to embedded systems of interest to STEM. STEM incorporates 37 Meta Attack Pattern CAPECs that are relevant to an embedded system. Mitigations are linked to CAPECs so that controls are only suggested if they are useful in mitigating attacks that have a defined effect on the system under consideration. STEM encompasses three types of rules to identify CAPEC vulnerabilities, suggest NIST mitigations, and associate NIST mitigations with defense properties for attack scenarios. An example of a STEM rule to identify CAPEC-131 is shown in Fig. 2.

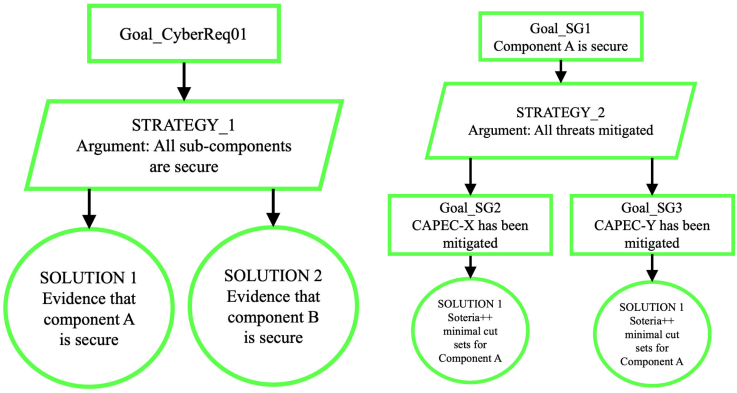
```
Rule Vul-CAPEC-131-1
if oneOf(componentType of a Subsystem,
  Software, SwHwHybrid, SwHumanHybrid, Hybrid)
and insideTrustedBoundary of a Subsystem is true
and dataReceivedIsUntrusted of the Subsystem is true
then applicableCM of the Subsystem is CAPEC-131A-ResourceAvailability.
```

Fig. 2. An example of a STEM rule to identify CAPEC-131 Resource Leak Exposure.

The SOTERIA++ Tool. Once STEM identifies all *applicable* CAPEC attacks and *implemented* NIST controls for components or connections of an architecture model, the information is fed into SOTERIA++ [31] for further analysis. The *severity* field of a cyber requirement defines the top-level cyber security goal of a system, which corresponds to a quantitative acceptable level of risk. The acceptable level of risk indicates that the likelihood of successful attack should be less than or equal to one failure for every 10^d hours of system operation, where d is a DAL value. SOTERIA++ analyses the system to determine whether it satisfies the top-level goal or not. Starting with the *condition* field of a cyber requirement, it back-traces each atomic *output:CIA* expression via cyber relations and connections to construct an attack-defense tree. The non-leaf node of an attack-defense tree is a logical (AND, OR, or NOT) operator; the leaf node is either an attack or a mitigated attack. Note that the NOT operator can only be applied to defenses, and is used for convenience to convert the DAL of defense to a likelihood (10^{-DAL}). An attack-defense tree is used to compute the likelihood of a successful attack against a system. It represents possible ways to violate a cyber requirement for a system in which an adversary is attempting attacks given the implementation rigors of defenses. This is the reason we compute likelihoods and not probabilities. Worse case is assumed for attacks. As a result, the likelihood of attack is always 1. This then leads the designer to focus on applying rigor to the defense implementation in order to lower the likelihood of an attack. The implementation is a defense profile, which is a conjunction or disjunction set of defenses. The likelihood of a conjunction set of defenses (i.e., all defenses are required to mitigate an attack) is the maximum of the defense DALs; the likelihood of a disjunction set (i.e., any defense in the profile by itself can mitigate an attack) is the minimum. The intuition behind the min/max calculation is to encourage the designer to focus on the most rigorous defense. The likelihood of an attack is calculated by doing a minimum of the attack (which is always 1) and the likelihood of the set of defenses. An attack-defense tree is said to be mitigated if the calculated likelihood of defenses is less than or equal to the severity level assigned for the top-level goal for a system; thus the cyber requirement corresponding to the attack-defense tree is satisfied.

2.4 Security Assurance Case Construction

To construct meaningful and accurate security assurance case fragments, it is important to ensure that there is a succinct argument that shows how a cyber requirement is supported by the mitigations suggested by STEM. However, a cyber requirement can be dependent upon several components and each component can be vulnerable to multiple CAPEC threats. Embedding all the assurance information for a cyber requirement into a single security case fragment may involve a GSN with hundred of nodes, making the fragment difficult to comprehend and visualize in physical or even digital form. Therefore, a security case fragment for every cyber requirement is decomposed into two classes of fragments:



(a) A requirement-level fragment (b) A component-level fragment

Fig. 3. Sample structures of a requirement-level and a component-level security case fragment.

- *Requirement-level fragments* (Fig. 3a) - There is a single requirement-level fragment for a cyber requirement whose root goal claims that the requirement has been satisfied. The solution nodes provide evidences for the security of every component that affects the requirement. Clicking on a solution node causes associated component-level fragments to be displayed. The root goal is supported by a strategy that argues that all sub-components are secure.
- *Component-level fragments* (Fig. 3b) - Each solution node for a requirement-level fragment is a separate component-level fragment whose root goal claims that the component is secure. The root goal has a sub-goal for every CAPEC threat that the component is vulnerable to, each of which claims that the threat has been mitigated. The sub-goals are directly supported by solution nodes which are evidences generated by SOTERIA++. The root goal uses a strategy which argues that all threats have been mitigated. The solution node for a CAPEC threat shows the computed and acceptable likelihoods for the threat, and the implemented NIST defenses when hovered upon.

3 Case Study: A Delivery Drone

To demonstrate the effectiveness of our approach, we use the AADL model of an unmanned package delivery drone. The architecture of the delivery drone is provided in Fig. 4 and its AADL model is publicly available on Github (see footnote 1). The drone has been designed to perform tasks that have certain associated mission and cyber requirements. In VERDICT, cyber requirements are independent of each other while mission requirements are dependent on a combination of cyber requirements. The requirements for the delivery drone have been provided by security domain experts and have been specified in the *verdict* annex of the AADL model of the drone.

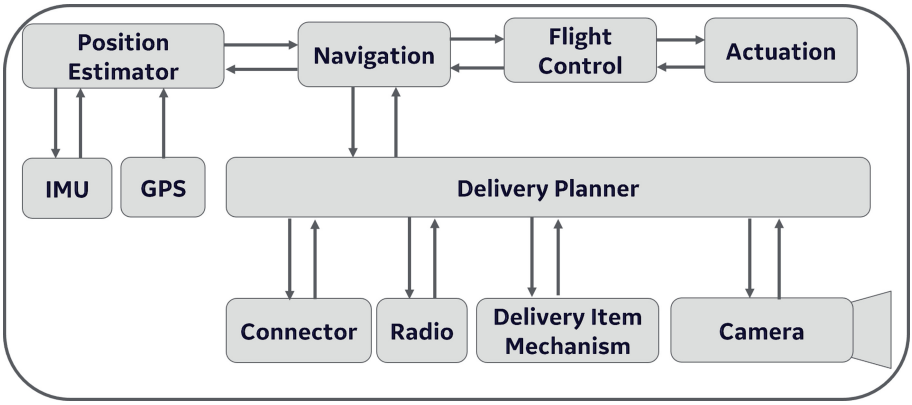


Fig. 4. The delivery drone system architecture.

Consider cyber security requirement CyberReq01 in Fig. 1, which states that the *"The drone shall be resilient to the loss of ability to deliver a package to the appropriate consumer location."* To fulfill this requirement, it has been determined by a security expert that the integrity and availability of *actuation_out* of *actuation* and *delivery_status* of *delivery item mechanism* component shall not be compromised. Moreover, the consequence of failure is hazardous, which corresponds to an acceptable likelihood of failure 1e−07. To declare a system to be secure in the context of VERDICT, we need to show that all applicable CAPECs to its subcomponents have been sufficiently mitigated by the implemented defenses in the system. Specifically, all the subcomponents propagating threats that could affect the integrity and availability of those outputs have been secured. The AADL model annotated with properties is then fed into the VERDICT tool for analysis. A requirement-level security case fragment for CyberReq01 is returned as shown in Fig. 5.

It is evident from the figure that CAPEC threats to the *actuation* component are not all mitigated. When clicking on the solution node for the *actuation* component, a component-level fragment is displayed as Fig. 6, which shows all applicable CAPEC threats to the component and that the specific CAPECs have not been mitigated. In this case, actuation is susceptible to CAPEC-390 (Bypassing Physical Security), which has not been sufficiently mitigated, as the computed likelihood for a successful CAPEC-390 attack is 1e−05, which is greater than the acceptable likelihood 1e−07. In other words, the defense (NIST-SE-3 System Access Control) has only been implemented to the design assurance level of 5, which is not sufficient to mitigate CAPEC-390 to avoid hazardous consequences. The minimal custset evidence computed by Soteria++ can be viewed by hovering over or clicking on the solution node. In practice, what happens at this point is that the security expert works with the system design team to upgrade the defense. Upgrading the defense means revisiting the list of required design activities and performing additional activities to gain credit for a higher

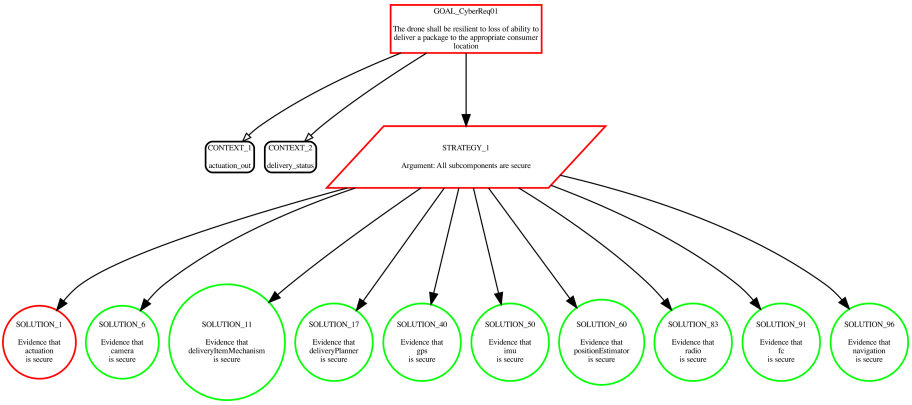


Fig. 5. The requirement-level security fragment for CyberReq01.

assurance level. Activities include traditional software assurance items such as configuration management, requirements management, requirements verification, and testing. The list also includes specific security assurance activities such as penetration testing and vulnerability assessment. These activities are incorporated into the System Engineering design process of the drone. The system designer can then update the defenses from design assurance level 5 to 7, and then run the new model again with VERDICT to show that the system is secure.

4 Discussion

Security cases allow developers to easily detect security flaws in the design of security-critical systems, thereby enabling them to go back to the design phase and include security features that were initially missed. They also allow capturing the rationale behind adding more stringent security features that give a clear benefit and those for which the benefit is not very obvious [4]. While developing the capability to generate security cases from the VERDICT toolchain, we considered the best ways to enable end-users to easily detect and isolate security issues in the design of a system. One of the unique aspects of the security case generation feature of the VERDICT toolchain is the ability to trace back to the source of potential security issues by visual inspection of the colored branches of the GSN fragments. This not only saves time but also makes it easy for users who are not experienced with the intricacies of the toolchain to understand the cause of a problem and provide specific feedback to address the same.

Developing security cases is fairly complicated as they have to consider the high uncertainties involved with how an intelligent adversary may attack a system. Therefore, they should be designed in a way so as to indicate the cause of possible security flaws with as much specificity as possible. This is because sometimes, adding a feature to avoid one security flaw may lead to the creation of another flaw that was originally absent. Therefore, an accurate error-tracing

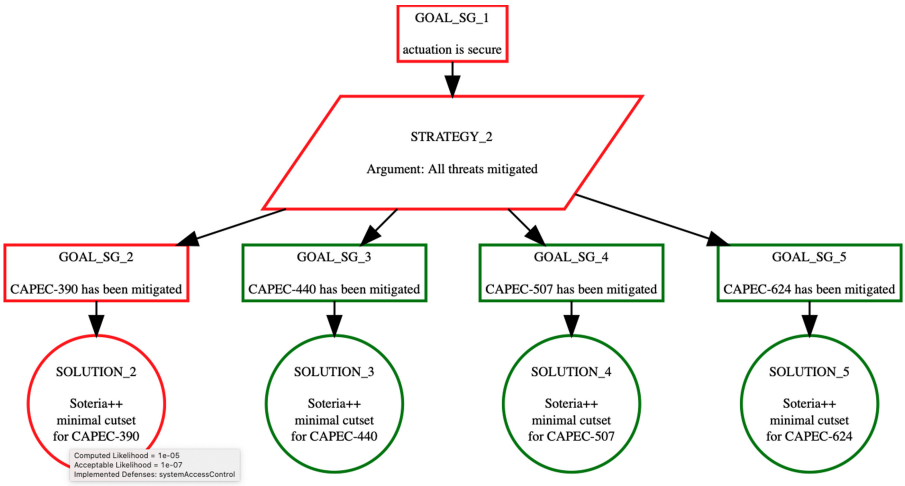


Fig. 6. The component-level security fragment for the *actuation* component.

capability is an important factor in security cases. Automated security case patterns should be designed in such a way so that they can provide enough information to capture the essence of an issue. Modularity is important as it allows the security case fragments to be inspected and reviewed independently, reducing the cognitive workload on the users. This also allows the cases to be aggregated into larger fragments as necessary when trying to analyze a system from varying levels of granularity. Another important aspect to keep in mind while designing automatic security cases is to limit the amount of additional information that developers have to include in the specification. This helps to keep the specifications concise and free from additional content that is not critical to the design.

5 Related Work

Denney *et al.* [12–15] present AdvoCATE, a toolset for automatic creation and management of safety cases. Resolute [19] is an assurance case language and tool where users manually formulate claims and the rules for justifying them. Meng *et al.* [25] formalize queryable safety assurance case in SADL language and use evidence from fault tree analysis to substantiate claims. Assurance cases have traditionally been used for arguing about system safety. Existing work on security cases has primarily investigated the efficacy of using assurance cases for arguing about security. Alexander *et al.* [4] provide a detailed analysis of the benefits of using assurance cases for arguing about system security concerns. They point out the fundamental differences between safety and security concerns that make it necessary to consider external adversarial threats in security cases. Agudo *et al.* [3] investigate the development of security cases by mapping different stages of the system development life-cycle to the structure of the security cases.

Vivas *et al.* [34] present a similar approach for integrating security cases with the security engineering phase of development in which the argument structure of a security claim represents the structure of the system development phase. Poreddy and Corns [29] argue the security of a generic avionics mission controls system by analyzing the potential threats to the mission control computer and constructing arguments for tangible claims in the GSN form. The AMASS Tool Platform [11] provides a novel holistic approach to support assurance and certification for Cyber Physical Systems. Yamamoto and Kobayashi [24] propose the creation of security assurance cases for arguing the security of mobile architecture models.

We improve upon existing work by creating domain-specific patterns to automatically generate security case fragments using MITRE's CAPEC threats and NIST's cyber defenses. The toolchain implementation also provides advantages such as automatic pattern instantiation, automatic generation of assurance case fragments, user-defined argument instantiation, integration and management of formal method tools based evidences, modularity of fragments, and easy management of fragments. XML artifacts are generated for the fragments that can be consumed by other tools for assimilation of lower-level fragments to create higher-level fragments, archiving, and further ontological and formal analysis.

6 Conclusion and Future Work

This paper presented an approach for constructing security case fragments using an augmented GSN schema that can present useful information to designers even if one or more claims cannot be substantiated. The fragments can be automatically generated using assurance case patterns without requiring any manual formulation or specification from the users. The security case fragments consider the component-level CAPEC vulnerabilities of a system to argue about the cyber requirement level claims. An implementation of the approach was presented as a functionality of the VERDICT toolchain and a case study was presented to show the effectiveness of the approach. A potential application of the tool would be to assemble security cases to certify systems towards cyber-security standards such as ASTM-F3286 Standard Guide for Cybersecurity and Cyberattack Mitigation.

One potential future direction would be to extend STEM to include more threats and defenses that target at other areas such as network communications, or integrate with tools like CyVAF [27] that can cover a broader range of threats and defenses. Also, we would like to investigate how the evidence generated by other formal methods tools can be used for constructing security assurance case fragments. This would involve expanding the current assurance case pattern library to accommodate the verbiage of other tools.

Acknowledgement. Distribution Statement "A" (Approved for Public Release, Distribution Unlimited). This research was funded by the Defense Advanced Research Projects Agency (DARPA). The views, opinions and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

References

1. Common Attack Pattern Enumeration and Classification (CAPEC) (2017). https://capec.mitre.org
2. Security and Privacy Controls for Information Systems and Organizations (2017)
3. Agudo, I., Vivas, J.L., López, J.: Security assurance during the software development cycle. In: Proceedings of the International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing, pp. 1–6 (2009)
4. Alexander, R., Hawkins, R., Kelly, T.: Security Assurance Cases: Motivation and the State of the Art. The University of York, York (2011)
5. Bagheri, H., Kang, E., Mansoor, N.: Synthesis of assurance cases for software certification. In: Proceedings of the International Conference on Software Engineering (2020)
6. Basir, N., Denney, E., Fischer, B.: Deriving safety cases for hierarchical structure in model-based development. In: Schoitsch, E. (ed.) SAFECOMP 2010. LNCS, vol. 6351, pp. 68–81. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15651-9_6
7. Bloomfield, R., Netkachova, K.: Building blocks for assurance cases. In: 2014 IEEE International Symposium on Software Reliability Engineering Workshops, pp. 186–191. IEEE (2014)
8. Bloomfield, R., Rushby, J.: Assurance 2.0: A manifesto (2020)
9. Cheah, M., Shaikh, S.A., Bryans, J., Wooderson, P.: Building an automotive security assurance case using systematic security evaluations. Comput. Secur. **77**, 360–379 (2018)
10. Crapo, A., Moitra, A.: Toward a unified English-like representation of semantic models, data, and graph patterns for subject matter experts. Int. J. Semant. Comput. **7**(03), 215–236 (2013)
11. De La Vara, J., Parra, E., Ruiz, A., Gallina, B.: The amass tool platform: an innovative solution for assurance and certification of cyber-physical systems. In: Joint 26th International Conference on Requirements Engineering: Foundation for Software Quality Workshops, Pisa, Italy, vol. 2584. CEUR-WS (2020)
12. Denney, E., Pai, G.: A methodology for the development of assurance arguments for unmanned aircraft systems. In: 33rd International System Safety Conference (ISSC 2015) (2015)
13. Denney, E., Pai, G.: Automating the assembly of aviation safety cases. IEEE Trans. Reliab. **63**(4), 830–849 (2014)
14. Denney, E., Pai, G.: Tool support for assurance case development. Autom. Softw. Eng. **25**(3), 435–499 (2017). https://doi.org/10.1007/s10515-017-0230-5
15. Denney, E., Pai, G., Pohl, J.: AdvoCATE: an assurance case automation toolset. In: Ortmeier, F., Daniel, P. (eds.) SAFECOMP 2012. LNCS, vol. 7613, pp. 8–21. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33675-1_2
16. Feiler, P.: The Open Source AADL Tool Environment (OSATE). Technical report, Carnegie Mellon University (2019)
17. Feiler, P.H., Gluch, D.P.: Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language. Addison-Wesley, Boston (2012)
18. Feiler, P.H., Gluch, D.P., Hudak, J.J.: The architecture analysis & design language (AADL): An introduction. Technical report, Carnegie Mellon University (2006)
19. Gacek, A., Backes, J., Cofer, D., Slind, K., Whalen, M.: Resolute: an assurance case language for architecture models. ACM SIGAda Ada Lett. **34**(3), 19–28 (2014)

20. Graydon, P.J.: Formal assurance arguments: a solution in search of a problem? In: 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, pp. 517–528. IEEE (2015)
21. Guerra, S., Sheridan, D.: Compliance with standards or claim-based justification? The interplay and complementarity of the approaches for nuclear software-based systems. In: Proceedings of the Twenty-Second Safety-Critical Systems Symposium, Brighton, UK (2014)
22. Hawkins, R., Clegg, K., Alexander, R., Kelly, T.: Using a software safety argument pattern catalogue: two case studies. In: Flammini, F., Bologna, S., Vittorini, V. (eds.) SAFECOMP 2011. LNCS, vol. 6894, pp. 185–198. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24270-0_14
23. Kelly, T., Weaver, R.: The goal structuring notation-a safety argument notation. In: Proceedings of the Dependable Systems and Networks 2004 Workshop on Assurance Cases, p. 6. Citeseer (2004)
24. Kobayashi, N., Morisaki, S., Yamamoto, S.: Mobile security assurance for automotive software through ArchiMate. In: You, I., Leu, F.-Y., Chen, H.-C., Kotenko, I. (eds.) MobiSec 2016. CCIS, vol. 797, pp. 10–20. Springer, Singapore (2018). https://doi.org/10.1007/978-981-10-7850-7_2
25. Meng, B., et al.: Towards developing formalized assurance cases. In: 2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC), pp. 1–9 (2020). https://doi.org/10.1109/DASC50938.2020.9256740
26. Meng, B., et al.: VERDICT: a language and framework for engineering cyber resilient and safe system. Syst. **9**(1) (2021). https://doi.org/10.3390/systems9010018. https://www.mdpi.com/2079-8954/9/1/18
27. Meng, B., Smith, W., Durling, M.: Security threat modeling and automated analysis for system design. SAE Int. J. Transp. Cyber Privacy **4** (2021). https://doi.org/10.4271/11-04-01-0001
28. Moitra, A., Prince, D., Siu, K., Durling, M., Herencia-Zapana, H.: Threat identification and defense control selection for embedded systems. SAE Int. J. Transp. Cyber. Privacy **3** (2020)
29. Poreddy, B.R., Corns, S.: Arguing security of generic avionic mission control computer system (MCC) using assurance cases. Proc. Comput. Sci. **6**, 499–504 (2011)
30. RTCA-DO: 178c: Software considerations in airborne systems and equipment certification (2011)
31. Siu, K., Herencia-Zapana, H., Prince, D., Moitra, A.: A model-based framework for analyzing the security of system architectures. In: 2020 Annual Reliability and Maintainability Symposium (RAMS), pp. 1–6. IEEE (2020)
32. Siu, K., et al.: Architectural and behavioral analysis for cyber security. In: 2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC), pp. 1–10. IEEE (2019)
33. Sommerville, I.: Software Engineering (2011). ISBN-10 137035152, 18
34. Vivas, J.L., Agudo, I., López, J.: A methodology for security assurance-driven system development. Requir. Eng. **16**(1), 55–73 (2011)
35. Wei, R., Kelly, T.P., Dai, X., Zhao, S., Hawkins, R.: Model based system assurance using the structured assurance case metamodel. J. Syst. Softw. **154**, 211–233 (2019)