Using Assurance Cases to Prevent Malicious Behaviour from Targeting Safety Vulnerabilities

Victor Bandur(✉), Mark Lawford, Sébastien Mosser, Richard F. Paige, Vera Pantelic, and Alan Wassyng

McMaster Centre for Software Certification, McMaster University, Canada
{bandurvp, lawford, mossers, paigeri, pantelv, wassyng}@mcmaster.ca

Abstract. We discuss an approach to modifying a safety assurance case to take into account malicious intent. We show how to analyze an existing assurance case to reveal additions and modifications that need to be made in order to deal with the effects of malicious intent aimed at safety critical applications, and where to make them.

1 Overview

In 2018 researchers in the McMaster Centre for Software Certification (McSCert) published a paper on safe and secure over-the-air software updates [2].

In the paper, we presented an assurance case (AC) we developed based on ISO 26262 [4] for functional safety, and on SAE J3061 [9] for cybersecurity. We were looking for a way to integrate safety and security. The top-level of that AC is shown in Figure 1, in a GSN-like notation.

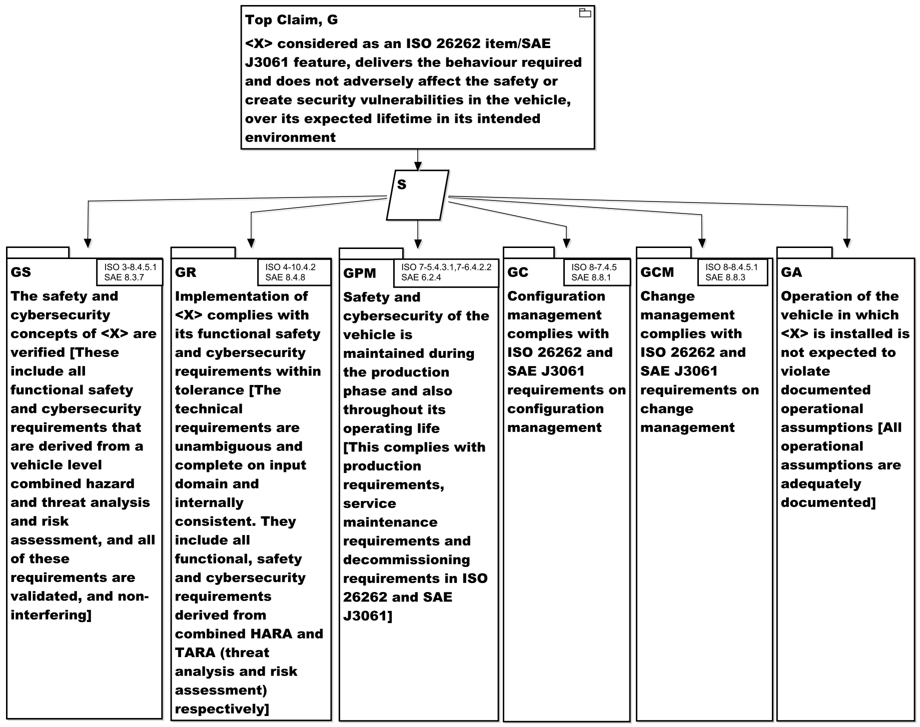
The approach was reasonably successful in that it helped us find a vulnerability in the implementation of the Uptane framework (see [1]). Later, we began to question our integration of safety and security as equals in the AC and in hazard analyses. The reason we began to doubt that strategy is captured in Figure 2.

The figure shows how malicious acts, including cybersecurity threats, can bypass safety mitigations built into a system.

From this grew the idea that we may be able to protect the safety of a system from malicious intent without a general cybersecurity analysis and mitigation. This is the first component of this research – explore using the safety argument to identify security concerns. Future work will include comparison with a more traditional approach that deals with safety and security as equals.

2 A Fast Food Coffee Cup Example

The following coffee cup example originated in McSCert, with an early version of the top-level of the AC published in 2020 [3]. We explored ACs that did not include software since we wanted to focus on safety (freedom from loss/harm), irrespective of the nature of the system. Everything we had analyzed up till then included software as a major component of the system. We chose this as



Top Claim, G

<X> considered as an ISO 26262 item/SAE J3061 feature, delivers the behaviour required and does not adversely affect the safety or create security vulnerabilities in the vehicle, over its expected lifetime in its intended environment

S

GS ISO 3-8.4.5.1 SAE 8.3.7

The safety and cybersecurity concepts of <X> are verified [These include all functional safety and cybersecurity requirements that are derived from a vehicle level combined hazard and threat analysis and risk assessment, and all of these requirements are validated, and non-interfering]

GR ISO 4-10.4.2 SAE 8.4.8

Implementation of <X> complies with its functional safety and cybersecurity requirements within tolerance [The technical requirements are unambiguous and complete on input domain and internally consistent. They include all functional, safety and cybersecurity requirements derived from combined HARA and TARA (threat analysis and risk assessment) respectively]

GPM ISO 7-5.4.3.1,7-6.4.2.2 SAE 6.2.4

Safety and cybersecurity of the vehicle is maintained during the production phase and also throughout its operating life [This complies with production requirements, service maintenance requirements and decommissioning requirements in ISO 26262 and SAE J3061]

GC ISO 8-7.4.5 SAE 8.8.1

Configuration management complies with ISO 26262 and SAE J3061 requirements on configuration management

GCM ISO 8-8.4.5.1 SAE 8.8.3

Change management complies with ISO 26262 and SAE J3061 requirements on change management

GA

Operation of the vehicle in which <X> is installed is not expected to violate documented operational assumptions [All operational assumptions are adequately documented]

Fig. 1. Top-Level of an Assurance Case for Over-the-Air Software Updates - from [2]

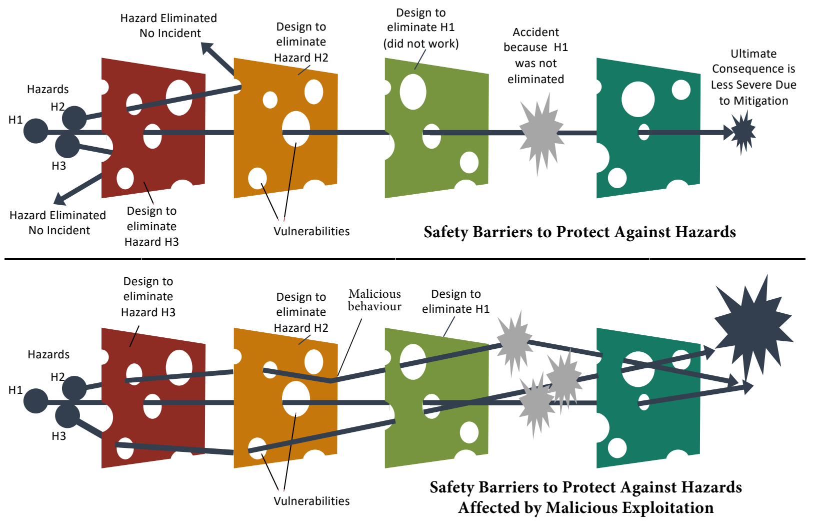

Hazard Eliminated
No Incident
Design to eliminate Hazard H2
Design to eliminate H1 (did not work)
Accident because H1 was not eliminated
Ultimate Consequence is Less Severe Due to Mitigation
Hazards
H2
H1
H3
Hazard Eliminated
No Incident
Design to eliminate Hazard H3
Vulnerabilities
Safety Barriers to Protect Against Hazards
Design to eliminate Hazard H3
Design to eliminate Hazard H2
Malicious behaviour
Design to eliminate H1
Hazards
H2
H1
H3
Vulnerabilities
Safety Barriers to Protect Against Hazards Affected by Malicious Exploitation

Fig. 2. Malicious Exploitation of Safety Vulnerabilities

the example to use for this analysis to highlight that we are interested in all malicious activities and are not restricted to cybersecurity. Examples of such cups are those deployed in fast food enterprises.

Figure 3 shows the top-level of the coffee cup AC. The top-level of the previ-

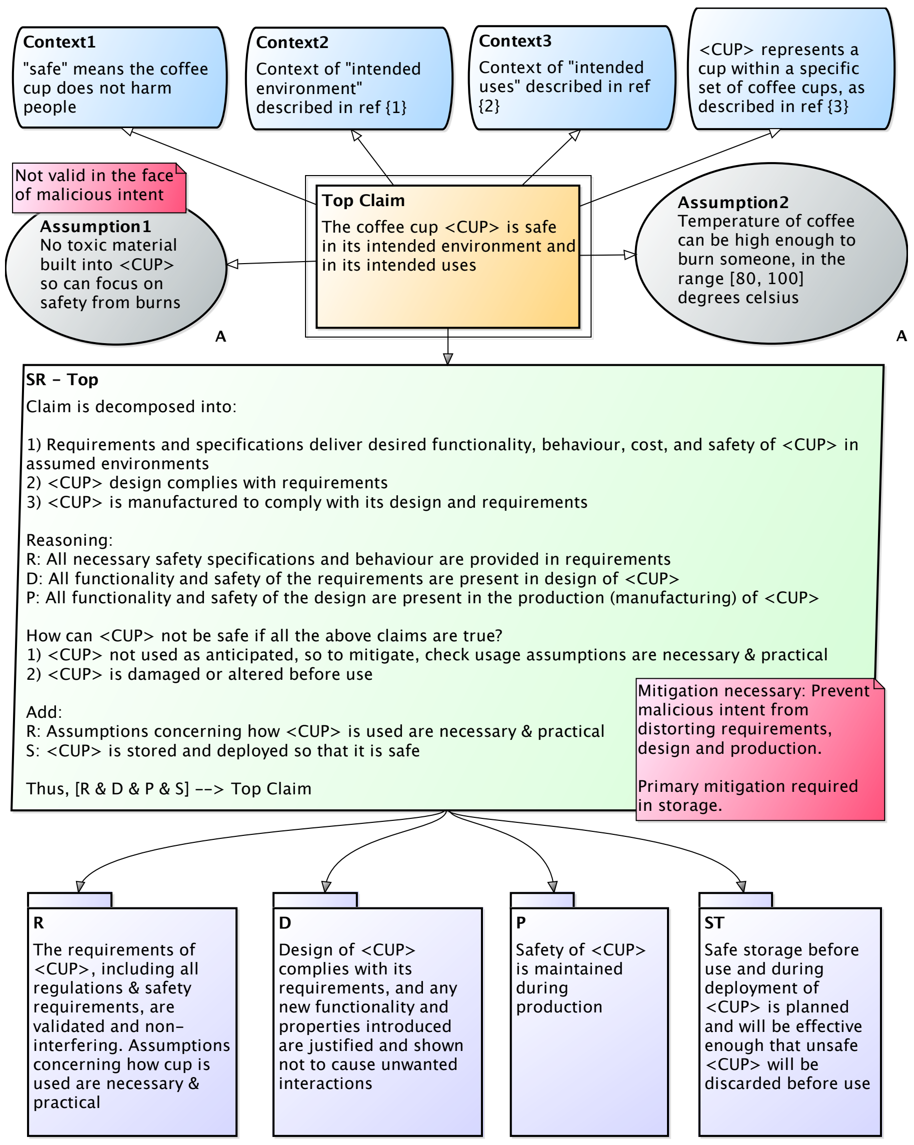

Context1
"safe" means the coffee cup does not harm people
Context2
Context of "intended environment" described in ref {1}
Context3
Context of "intended uses" described in ref {2}
<CUP> represents a cup within a specific set of coffee cups, as described in ref {3}
Not valid in the face of malicious intent
Assumption1
No toxic material built into <CUP> so can focus on safety from burns
A
Top Claim
The coffee cup <CUP> is safe in its intended environment and in its intended uses
Assumption2
Temperature of coffee can be high enough to burn someone, in the range [80, 100] degrees celsius
A
SR – Top
Claim is decomposed into:
1) Requirements and specifications deliver desired functionality, behaviour, cost, and safety of <CUP> in assumed environments
2) <CUP> design complies with requirements
3) <CUP> is manufactured to comply with its design and requirements
Reasoning:
R: All necessary safety specifications and behaviour are provided in requirements
D: All functionality and safety of the requirements are present in design of <CUP>
P: All functionality and safety of the design are present in the production (manufacturing) of <CUP>
How can <CUP> not be safe if all the above claims are true?
1) <CUP> not used as anticipated, so to mitigate, check usage assumptions are necessary & practical
2) <CUP> is damaged or altered before use
Add:
R: Assumptions concerning how <CUP> is used are necessary & practical
S: <CUP> is stored and deployed so that it is safe
Thus, [R & D & P & S] --> Top Claim
Mitigation necessary: Prevent malicious intent from distorting requirements, design and production.
Primary mitigation required in storage.
R
The requirements of <CUP>, including all regulations & safety requirements, are validated and non-interfering. Assumptions concerning how cup is used are necessary & practical
D
Design of <CUP> complies with its requirements, and any new functionality and properties introduced are justified and shown not to cause unwanted interactions
P
Safety of <CUP> is maintained during production
ST
Safe storage before use and during deployment of <CUP> is planned and will be effective enough that unsafe <CUP> will be discarded before use

Fig. 3. Top-level in Safety AC of Coffee Cup

ously developed safety AC is analyzed to determine which assumptions, claims and evidence could be affected by malicious intent. The red notes in the figure show the results of this initial analysis.

Most of the effects of malicious intent will be detected at lower levels of the AC. However, even at the top-level, Figure 3, one of the items of interest, if we consider security, is that the overall assumption that toxic material is not built into the cup may not be valid in the face of malicious intent. This, and other items, have implications for the analysis at the lower levels.

This is the first step in including the effects of malicious intent in the preservation of safety. Obviously, the better the AC, the easier it is to perform this analysis. In particular, relevant assumptions must be visible at this level, and the subclaims need to have enough detail to determine whether or not they are likely to be affected by malicious intent. However, even if we miss detecting a potential impact, we have later opportunities to do so.

The next step is to analyze the next lower level in the AC. Again, this is simply an exploration of which branches of the argument will be affected. Due to space limitations, we will focus only on the "ST" branch, shown in Figure 4. This was already determined to be of interest as highlighted in Figure 3. At this stage it is clear that malicious intent could impact every one of the four subclaims in ST. Storage is important in terms of safety, because weakened/damaged/altered cups may no longer comply with requirements. However, the safety AC does not consider malicious intent at all.

Further analysis at the child level of the argument reveals additions required to the AC. An example is shown in Figure 5. In this figure we see that safety considerations are not adequate to prevent against malicious action. Some of the mitigation for malicious action could have been included in safety mitigation, but it would be costly and was not thought to be necessary, based on risk. In particular, the check on toxic substances when malicious intent was not included, assumed that toxic substances could find their way into the cup material only if the cup was stored in an environment in which this could happen. To protect against malicious intent requires much more surveillance as to how the packages of cups are stored and whether or not anyone interferes with a package.

The resulting modified AC is shown in Figure 6. Note that:

- The *security* argument can be appended to the *safety* argument on a local level
- Localized like this, we can analyze the argument for potential interaction between safety and security – this is not necessarily trivial or even easy, and may require analysis completely separate from the AC itself
- We can construct patterns for these arguments (see below)

3 Patterns for Arguments that Add Security to Safety

3.1 Add Effect of Malicious Intent to Safety Argument

Since our plan in general is to develop a safety AC and then add security, an obvious initial pattern for this is to simply add conjunctive security claims to

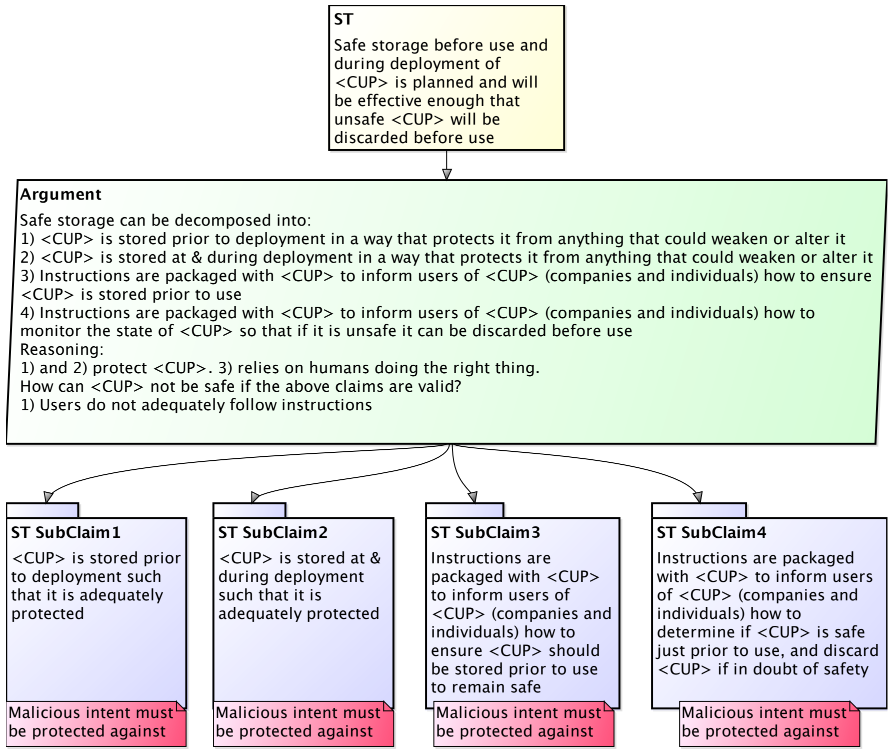

ST
Safe storage before use and during deployment of <CUP> is planned and will be effective enough that unsafe <CUP> will be discarded before use
Argument
Safe storage can be decomposed into:
1) <CUP> is stored prior to deployment in a way that protects it from anything that could weaken or alter it
2) <CUP> is stored at & during deployment in a way that protects it from anything that could weaken or alter it
3) Instructions are packaged with <CUP> to inform users of <CUP> (companies and individuals) how to ensure <CUP> is stored prior to use
4) Instructions are packaged with <CUP> to inform users of <CUP> (companies and individuals) how to monitor the state of <CUP> so that if it is unsafe it can be discarded before use
Reasoning:
1) and 2) protect <CUP>. 3) relies on humans doing the right thing.
How can <CUP> not be safe if the above claims are valid?
1) Users do not adequately follow instructions
ST SubClaim1
<CUP> is stored prior to deployment such that it is adequately protected
Malicious intent must be protected against
ST SubClaim2
<CUP> is stored at & during deployment such that it is adequately protected
Malicious intent must be protected against
ST SubClaim3
Instructions are packaged with <CUP> to inform users of <CUP> (companies and individuals) how to ensure <CUP> should be stored prior to use to remain safe
Malicious intent must be protected against
ST SubClaim4
Instructions are packaged with <CUP> to inform users of <CUP> (companies and individuals) how to determine if <CUP> is safe just prior to use, and discard <CUP> if in doubt of safety
Malicious intent must be protected against

Fig. 4. Storage Branch in Safety AC of Coffee Cup

the existing safety claims – as in Figure 6. This works in general, and it is simple. However, there are serious drawbacks to this approach. The most critical drawback is that this results in localized silos – one for safety and another for security. Based on our previous experience, this is something we had decided to avoid. First, there is obvious duplication in the different branches if we do this, which impacts maintenance of the AC. More importantly, there is less chance of analyzing interactions. Separating safety and security exacerbates the problems that GSN-like notations have in coping with cross-cutting concerns.

To avoid this, we should explore alternatives. Readers will have noticed that our *Strategy* nodes contain brief reasoning as to what subclaims and/or evidence are necessary to support the parent claim (sometimes included in Justification nodes in GSN). We believe the search for alternative claim structure starts here.

3.2 Re-analyze the Local Argument

In the example in Figure 6, the *Default safety plus security argument* is:

- <CUP> is stored so that moisture, pressure and toxic substances do not harm it

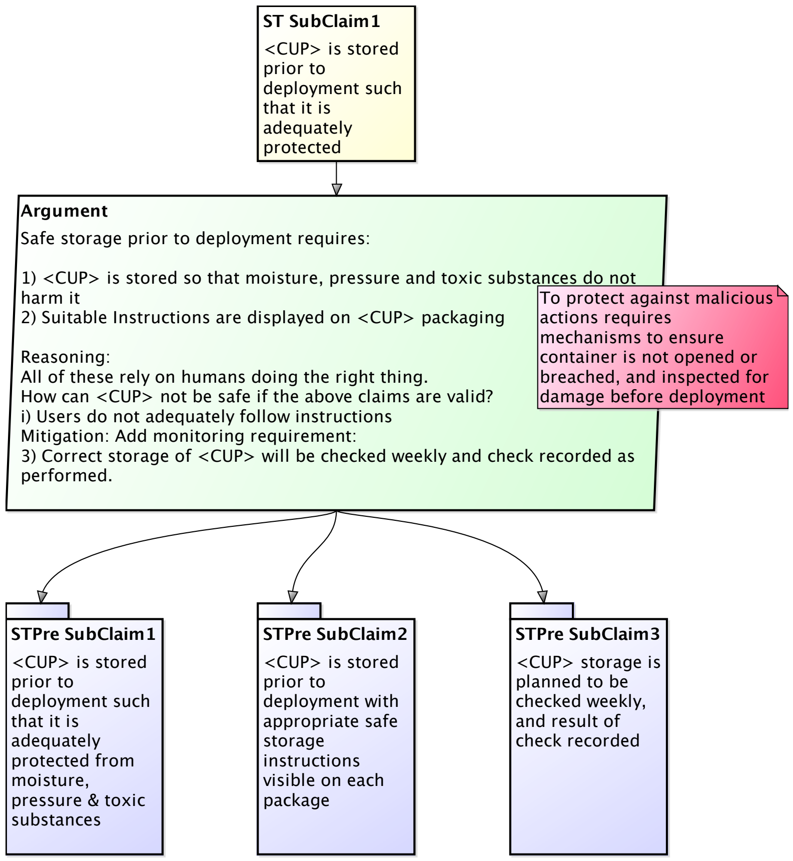

ST SubClaim1
<CUP> is stored prior to deployment such that it is adequately protected
Argument
Safe storage prior to deployment requires:
1) <CUP> is stored so that moisture, pressure and toxic substances do not harm it
2) Suitable Instructions are displayed on <CUP> packaging
Reasoning:
All of these rely on humans doing the right thing.
How can <CUP> not be safe if the above claims are valid?
i) Users do not adequately follow instructions
Mitigation: Add monitoring requirement:
3) Correct storage of <CUP> will be checked weekly and check recorded as performed.
To protect against malicious actions requires mechanisms to ensure container is not opened or breached, and inspected for damage before deployment
STPre SubClaim1
<CUP> is stored prior to deployment such that it is adequately protected from moisture, pressure & toxic substances
STPre SubClaim2
<CUP> is stored prior to deployment with appropriate safe storage instructions visible on each package
STPre SubClaim3
<CUP> storage is planned to be checked weekly, and result of check recorded

Fig. 5. Storage Pre-Deployment Branch in Safety AC of Coffee Cup

- Suitable instructions are displayed on <CUP> packaging
- Correct storage of <CUP> will be checked weekly and check recorded as performed
- Access to storage is restricted, and monitored
- Stored packages are protected from tampering
- People with access to storage are vetted

We notice immediately that there is a simple difference between the safety and security arguments: safety clauses refer to <CUP> and a security clause to packages of cups. The focus on packaging is vital to the security argument. The focus on <CUP> was natural in the safety argument and did not raise any concerns. However, the argument could easily have referred to packages of cups if necessary. With this in mind, and looking for ways to integrate this better, an *improved argument* may be:

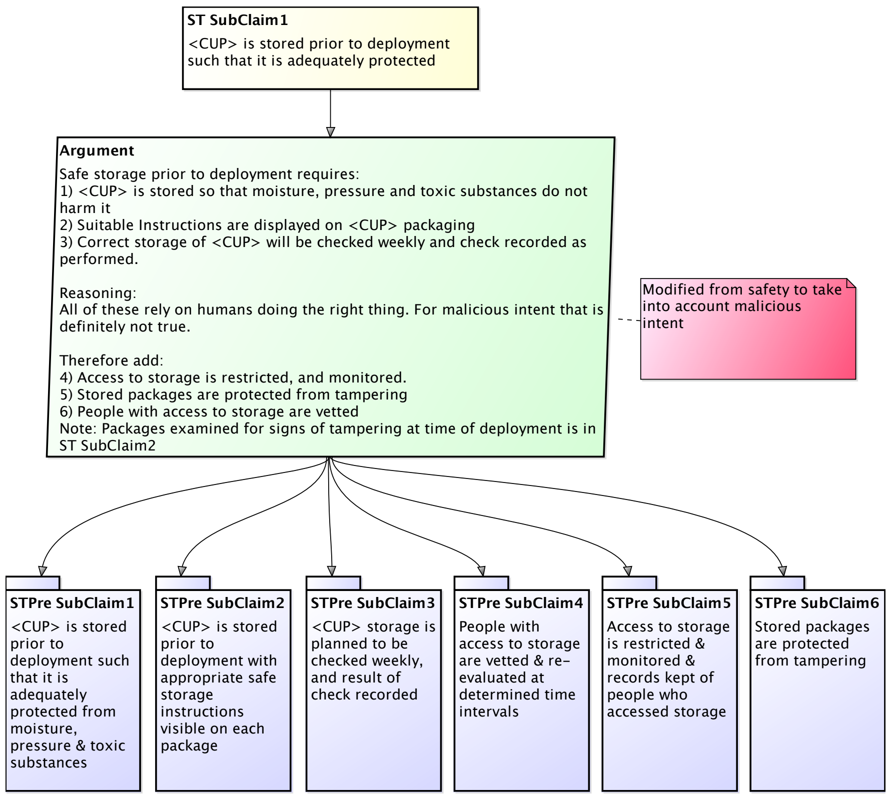

ST SubClaim1
<CUP> is stored prior to deployment such that it is adequately protected
Argument
Safe storage prior to deployment requires:
1) <CUP> is stored so that moisture, pressure and toxic substances do not harm it
2) Suitable Instructions are displayed on <CUP> packaging
3) Correct storage of <CUP> will be checked weekly and check recorded as performed.
Reasoning:
All of these rely on humans doing the right thing. For malicious intent that is definitely not true.
Therefore add:
4) Access to storage is restricted, and monitored.
5) Stored packages are protected from tampering
6) People with access to storage are vetted
Note: Packages examined for signs of tampering at time of deployment is in ST SubClaim2
Modified from safety to take into account malicious intent
STPre SubClaim1
<CUP> is stored prior to deployment such that it is adequately protected from moisture, pressure & toxic substances
STPre SubClaim2
<CUP> is stored prior to deployment with appropriate safe storage instructions visible on each package
STPre SubClaim3
<CUP> storage is planned to be checked weekly, and result of check recorded
STPre SubClaim4
People with access to storage are vetted & re-evaluated at determined time intervals
STPre SubClaim5
Access to storage is restricted & monitored & records kept of people who accessed storage
STPre SubClaim6
Stored packages are protected from tampering

Fig. 6. Storage Pre-Deployment Branch in Safe & Secure AC of Coffee Cup

- Packages containing <CUP>s are stored in a restricted and monitored storage area, in such a way that moisture, pressure, and toxic substances cannot harm the <CUP>s by accident or malicious action
- People with access to the storage area are vetted for access
- Suitable instructions are displayed on packages containing <CUP>s and cannot be removed or altered without leaving a noticeable trace
- Correct storage of packages containing <CUP>s will be checked for damage and/or tampering weekly, by at least two people, and the check immutably recorded as performed with names of the checkers (durations and number of people are placeholders here in lieu of more detailed analysis).

This results in the modified claim structure shown in Figure 7.

This shows how we can integrate safety and security at a particular level in the AC. A simple process view of it, for a particular claim decomposition, is:

1. Analyze the safety argument to identify where malicious intent affects safety, and add that to the argument (in the example this is shown in Figures 3, 4, 5 and resulted in Figure 6)

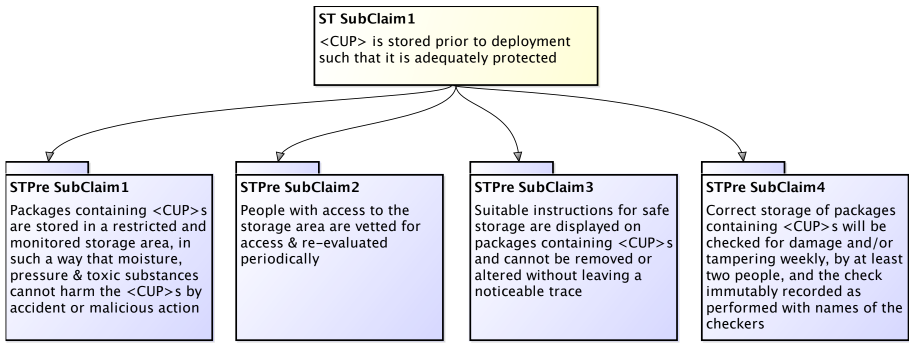

ST SubClaim1
<CUP> is stored prior to deployment such that it is adequately protected
STPre SubClaim1
Packages containing <CUP>s are stored in a restricted and monitored storage area, in such a way that moisture, pressure & toxic substances cannot harm the <CUP>s by accident or malicious action
STPre SubClaim2
People with access to the storage area are vetted for access & re-evaluated periodically
STPre SubClaim3
Suitable instructions for safe storage are displayed on packages containing <CUP>s and cannot be removed or altered without leaving a noticeable trace
STPre SubClaim4
Correct storage of packages containing <CUP>s will be checked for damage and/or tampering weekly, by at least two people, and the check immutably recorded as performed with names of the checkers

Fig. 7. Protecting Against Malicious Exploitation of Safety Vulnerabilities

2. Analyze wording in the reasoning clauses (the *Default argument*, above) to facilitate integrating safety & security
3. Restructure argument based on revised argument (the *Improved argument*, above)

3.3 Two Fundamental Assurance Patterns

There are two structural patterns that we feel are necessary at the lower levels of the AC in all safety and security ACs. The first pattern is based on an old idea that, for adequate safety (and security) assurance, we need a tripod of claims supported by evidence. This tripod is based on 3-Ps, Product/Process/People, as shown in Figure 8. For example, SubClaim1 in the right part of Figure 9 should be supported by claims that all relevant hazards, including mitigations are documented, that the hazard analysis method was appropriate, and that the team that performed the hazard analysis was competent to do so.

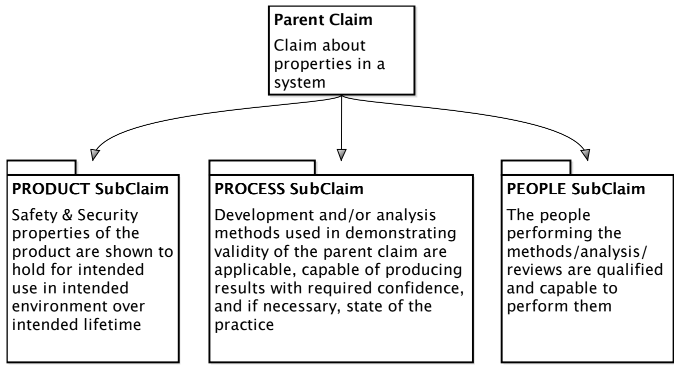

Parent Claim
Claim about properties in a system
PRODUCT SubClaim
Safety & Security properties of the product are shown to hold for intended use in intended environment over intended lifetime
PROCESS SubClaim
Development and/or analysis methods used in demonstrating validity of the parent claim are applicable, capable of producing results with required confidence, and if necessary, state of the practice
PEOPLE SubClaim
The people performing the methods/analysis/ reviews are qualified and capable to perform them

Fig. 8. Tripod Pattern of Product/Process/People

The second pattern is used to deal with a never-ending problem in assuring safety and security – completeness. This pattern is simple, but difficult to implement sufficiently well. It is shown in the left part of Figure 9 with an example to its right. SubClaim2 is the added component of the argument.

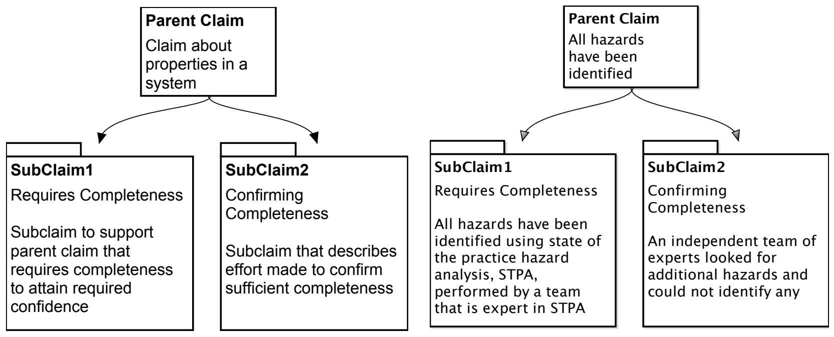

Parent Claim
Claim about properties in a system
SubClaim1
Requires Completeness
Subclaim to support parent claim that requires completeness to attain required confidence
SubClaim2
Confirming Completeness
Subclaim that describes effort made to confirm sufficient completeness
Parent Claim
All hazards have been identified
SubClaim1
Requires Completeness
All hazards have been identified using state of the practice hazard analysis, STPA, performed by a team that is expert in STPA
SubClaim2
Confirming Completeness
An independent team of experts looked for additional hazards and could not identify any

Fig. 9. Completeness Pattern and Example Instantiation

4 Related Work

There has been some related work on providing integrated assurance of safety and security. Some of this work focuses on considering security concerns within specific safety analyses, e.g., security-aware HAZOPs [8] or secure FTA [7]. Other approaches include security-aware STPA [11], or modeling approaches such as UMLsec [6] or security profiles associated with SysML, or the security annexe for AADL. Of additional note is SSAF [5], which attempts co-assurance of security and safety with checkpoints for ensuring no conflicts.

5 Conclusion

We are advocating that an effective way of dealing with the effect of security on the safety of a system, is to (surprisingly) first conduct the safety analysis and develop a safety Assurance Case Template (see [10]), which is constructed prior to system development. Then, by analyzing the safety AC for vulnerability to malicious intent, it is possible to integrate security assurance into the AC. It is important to note that the techniques we presented above apply equally well to cybersecurity threats as they do to the software-free coffee cup, and that this is not a full security analysis. It results in an AC that analyzes safety primarily, but takes into account how malicious activity can adversely affect the safety of the system. We believe the security "story" starts with a careful analysis of the assumptions and reasoning at every level of the safety AC though this technique does not apply to those argument segments in the AC that are confined to the application of a safety process, such as safety-only hazard analysis. We also reuse patterns that were developed for safety ACs, that are just as useful when malicious intent is introduced. Based on this initial security focused exploration of the safety AC, we believe it is possible to develop a systematic approach, based on an initial safety AC, to analyze the potential for malicious attacks on safety critical systems. Again, this is planned future work.

We do not completely agree with the view that says malicious behaviour is no different from inadvertent behaviour, and that if we cope with inadvertent behaviour it will be sufficient. We believe there is a real difference. For instance, we have a requirement that protects against inadvertently spilling the coffee. One mitigation is to design a lid that has safeguards built in so that accidentally knocking over the cup will not result in a spill. However, malicious behaviour could potentially damage the safeguard mechanism and not be obvious to a user. Inspecting the safety AC with respect to how malicious activity can adversely affect the safety argument helps to identify this "security" concern.

References

1. Uptane: Securing Software Updates for Automobiles. https://uptane.github.io/, accessed: 2023-04-25
2. Chowdhury, T., Lesiuta, E., Rikley, K., Lin, C.W., Kang, E., Kim, B., Shiraishi, S., Lawford, M., Wassyng, A.: Safe and secure automotive over-the-air updates. In: SAFECOMP 2018. pp. 172–187. Springer (2018)
3. Chowdhury, T., Wassyng, A., Paige, R.F., Lawford, M.: Systematic evaluation of (safety) assurance cases. In: Computer Safety, Reliability, and Security: 39th International Conference, SAFECOMP 2020, Lisbon, Portugal, September 16–18, 2020, Proceedings 39. pp. 18–33. Springer (2020)
4. ISO: 26262: Road vehicles-Functional safety. International Standard ISO/FDIS (2018)
5. Johnson, N., Kelly, T.: Devil's in the detail: Through-life safety and security co-assurance using SSAF. In: Romanovsky, A.B., Troubitsyna, E., Bitsch, F. (eds.) SAFECOMP 2019. LNCS, vol. 11698, pp. 299–314. Springer (2019)
6. Jürjens, J.: Umlsec: Extending UML for secure systems development. In: Jézéquel, J., Hußmann, H., Cook, S. (eds.) UML 2002. LNCS, vol. 2460, pp. 412–425. Springer (2002)
7. Kordy, B., Piètre-Cambacédès, L., Schweitzer, P.: Dag-based attack and defense modeling: Don't miss the forest for the attack trees. Comput. Sci. Rev. **13-14**, 1–38 (2014)
8. Macher, G., Sporer, H., Berlach, R., Armengaud, E., Kreiner, C.: SAHARA: a security-aware hazard and risk analysis method. In: Nebel, W., Atienza, D. (eds.) DATE 2015. pp. 621–624. ACM (2015)
9. SAE Vehicle Electrical System Security Committee, et al.: SAE J3061-Cybersecurity Guidebook for Cyber-Physical Automotive Systems. SAE-Society of Automotive Engineers (2016)
10. Wassyng, A., Singh, N.K., Geven, M., Proscia, N., Wang, H., Lawford, M., Maibaum, T.: Can product-specific assurance case templates be used as medical device standards? IEEE Design & Test **32**(5), 45–55 (2015)
11. Young, W., Leveson, N.G.: Systems thinking for safety and security. In: Jr., C.N.P. (ed.) ACSAC '13. pp. 1–8. ACM (2013)