THE UNIVERSITY of York

**High Integrity Systems Engineering Group**

# Security Assurance Cases: Motivation and the State of the Art

**Report Number :** CESG/TR/2011/1

**Authors : Rob Alexander, Richard Hawkins, Tim Kelly**

**Issue :** 1.1 **Date :** 7 April 2011

High Integrity Systems Engineering
Department of Computer Science
University of York
Deramore Lane
York YO10 5GH

## ABSTRACT

Assurance cases are widely used in the safely domain, where they provide a way to justify the safety of a system and render that justification open to review. They capture how low-level evidence (such as software test results) relates to high-level safety claims (e.g. “the aircraft’s thrust reversers will not deploy during flight”). They record why we believed, at the time of evaluating the system, that the system was adequately safe.

Assurance cases have not been widely used in security, but there is guidance available and there have been some promising experiments. There are a number of differences between safety and security which have implications for how we create security cases, but they do not appear to be insurmountable. It appears that the process of creating a security case is compatible with typical evaluation processes, and will have additional benefits in terms of training and corporate memory.

We recommend that CESG experiment with security assurance cases by creating one for a small low-security product, in parallel with their standard evaluation. York will be able to support this experiment using the expertise we have gained with safety cases.

## VERSIONS

| Version | Date | Reason |
|---|---|---|
| 1 | 18/03/2011 | First release to CESG |
| 1.1 | 07/04/2011 | Minor wording corrections |

# TABLE OF CONTENTS

# 1 INTRODUCTION

An assurance case is a structured argument that some system has some properties we desire; that it is safe, or reliable, or secure against attack. Defining *safety* cases in particular, Kelly says *"A safety case should communicate a clear, comprehensive and defensible argument that a system is acceptably safe to operate in a particular context."* [1] - we can generalise this easily to properties other than safety.

The argument in an assurance case shows how a very high-level claim (e.g. "the system is adequately secure") is ultimately supported by detailed evidence of particular low-level properties (e.g. some statistical testing data targeting a key security requirement on one small software component). An assurance case will refer to a range of evidence items, and will show how different types of evidence (e.g. evidence of good process, results of manual design review, and results of component testing) combine to give us confidence in higher-level properties [2]. In other words, the assurance case captures the *rationale* of why the evidence we have produced (the results of the low-level analysis activities we have carried out) gives us reason to believe the high level claim (which is, ultimately, what we are interested in). Evidence on its own is never enough – all evidence provides only partial support for a given high-level claim, and making that connection between evidence and claims requires an act of subjective judgment. Assurance cases make that subjective reasoning explicit.

An assurance case does not replace *any* specific technique for analysis or for generating evidence. What it does do is show the connection between those techniques and the high level claims we want to make.

An assurance case provides a way of capturing assumptions about a system – these may describe a particular context, including a specific version of the system, a description of its environment (including particular threats present in that environment) and a description of how we will use that system.

Security and safety are open problems – the domain is open-ended and ultimately somewhat subjective. Security is worse in this respect, because of increased uncertainty about the capabilities and knowledge of attackers. At any time, new information may arise (about our system or about its environment) and thus we may need to review our reasoning about the safety or security of the system. Assurance cases provide a record of what our reasoning was.

In a sense, whenever we honestly claim that a system is acceptably safe or secure, we have an implicit assurance case; we have some mental model behind that claim that we could probably describe if asked. By making an *explicit* assurance case, however, we open up that mental model to review and criticism by others, and we record our reasoning so that others can learn from it (for example, if they want to make a change to the system they can use the assurance case to assess some of the impact).

Acceptance of an assurance case is not a mechanical process; it requires subjective assessment by a customer, regulator or evaluator. This does not mean that it is wholly arbitrary and idiosyncratic; we can have rigour in assurance cases by drawing on the work that philosophers (such as Toulmin [3]) have done on informal logic – see [4] and [5] for an explanation of this. Key techniques include assurance case patterns (to capture best practice in argument structure), systematic review processes, and the use of appropriate notations. The last of those is critical.

## 1.1 Assurance Case Notations

An assurance case can be expressed in natural language; in ordinary prose with no particular structuring mechanism. Kelly [1] gives this fragment of a safety case as a real-world example of this done well:

"The Defence in Depth principle (P65) has been addressed in this system through the provision of the following:

• Multiple physical barriers between hazard source and the environment (see Section X)

• A protection system to prevent breach of these barriers and to mitigate the effects of a barrier being breached (see Section Y)"

It is quite easy to follow the chain of reasoning here; easy to see why the two sub-claims address the principle mentioned, easy to guess what kind of information will be in the two other sections referenced, and easy to see how those referenced sections will support the claims here. Kelly follows this example, however, with the following one (slightly obfuscated for confidentiality, but again from a real case):

"For hazards associated with warnings, the assumptions of [7] Section 3.4 associated with the requirement to present a warning when no equipment failure has occurred are carried forward. In particular, with respect to hazard 17 in section 5.7 [4] that for test operation, operating limits will need to be introduced to protect against the hazard, whilst further data is gathered to determine the extent of the problem."

This is much harder to read, particularly because of the many cross-references. It is not possible to understand the implications and validity of the argument in this paragraph without reading and understanding the references. The references in turn may require further chasing. In a document that may be hundreds or thousands of pages long and spread over multiple volumes, this is an onerous task.

Structured argument notations, such as the Goal Structuring Notation (GSN) and Claims Argument Evidence (CAE) provide a way to enforce clarity on these key issues of one claim depending on another. The forthcoming GSN standard says *"The key benefit from using an explicit approach such as GSN to develop and document the arguments of any assurance case is that it can improve comprehension amongst the key project stakeholders (e.g. system developers, engineers, independent assessors and certification authorities). In turn, this improves the quality of the debate and discussion amongst the stakeholders and can reduce the time taken to reach agreement on the argument approaches being adopted."*

A point about terminology: in some communication with CESG, the term "Claims-Argument-Evidence" (CAE) has been used to refer to what we call "structured argument approach" (i.e. an argument from top-level claims down to evidence through a tree or network of intermediate claims). CAE is a specific structured argument notation, developed by the safety consultancy Adelard. Here, we use the alternative notation GSN [1, 4] because it is slightly richer - GSN provides a slightly larger number of symbols to learn, but the gain in expressive power is well worth the small amount of extra learning time. In particular, GSN is fully compatible with the new OMG Argumentation Metamodel (ARM) [6] (which is in turn used in the forthcoming revision of the software assurance standard ISO 15026). In contrast, converting from ARM to CAE loses some information.

Figure 1 shows an example goal in GSN. A goal is used to state a claim.

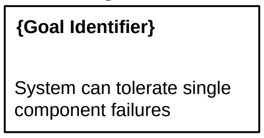

**Figure 1 - An Example Goal**

Where evidence is said to exist to support the truth of the claimed goal this can be documented by providing a *solution* in GSN. Figure 2 shows an example solution in GSN.

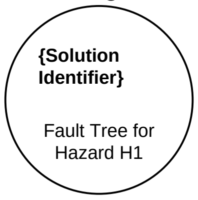

**Figure 2 - An Example Solution**

When documenting how *goals* are supported by *sub-goals* it can be useful to document the *reasoning step* – i.e. the nature of the argument that connects the goal to its sub-goals. This can be done in GSN by documenting the connecting argument *strategy.* Figure 3 shows an example strategy in GSN.

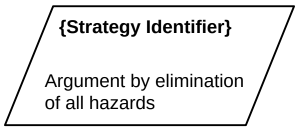

**Figure 3 - An Example Strategy**

When documenting a *goal* it can also be important to capture the *context* in which that claim should be interpreted. This is done in GSN by documenting context. Figure 4 shows an example context in GSN.

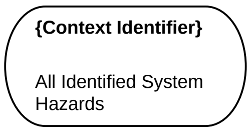

**Figure 4 - An Example Context**

Goals, strategies, solutions and context form the principal elements of GSN. Other element types exist – a full description can be found in [1].

When the elements of GSN are connected together they are said to form a “goal structure”. Figure 5 shows an example goal structure. Goal structures document the chain of reasoning in the argument (through the visible decomposition of claimed *goals* and description of argument *strategies*), how this argument is supported by evidence (through *solution*s) and clearly captures the *context* in which the claimed goals of the argument are being put forward.

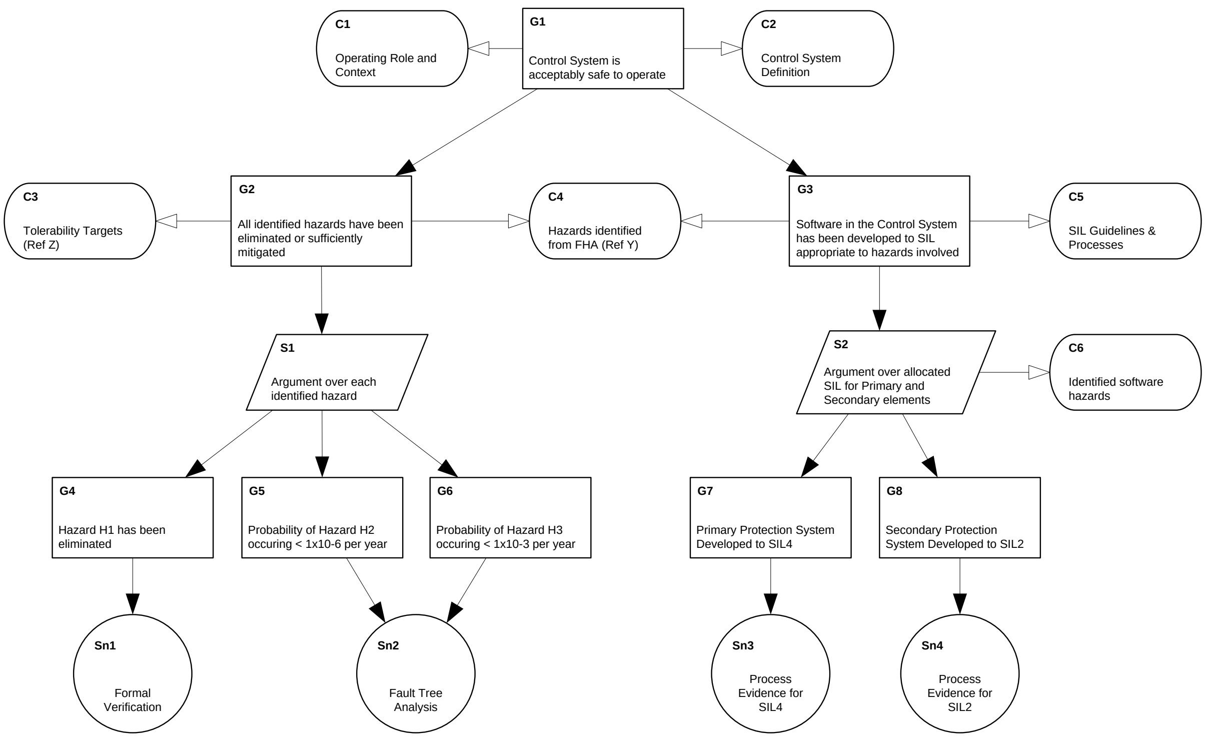
C1
Operating Role and Context
G1
Control System is acceptably safe to operate
C2
Control System Definition
C3
Tolerability Targets (Ref Z)
G2
All identified hazards have been eliminated or sufficiently mitigated
C4
Hazards identified from FHA (Ref Y)
G3
Software in the Control System has been developed to SIL appropriate to hazards involved
C5
SIL Guidelines & Processes
S1
Argument over each identified hazard
S2
Argument over allocated SIL for Primary and Secondary elements
C6
Identified software hazards
G4
Hazard H1 has been eliminated
G5
Probability of Hazard H2 occuring < 1x10-6 per year
G6
Probability of Hazard H3 occuring < 1x10-3 per year
G7
Primary Protection System Developed to SIL4
G8
Secondary Protection System Developed to SIL2
Sn1
Formal Verification
Sn2
Fault Tree Analysis
Sn3
Process Evidence for SIL4
Sn4
Process Evidence for SIL2

**Figure 5 - An Example Goal Structure**

### 1.2 Who Builds Assurance Cases?

It is typical, in safety, for the developer of a system to create a safety case for it and submit it to the customer or regulator for their assessment. In many safety regimes (for example, in the UK MOD one) this is required. If the vendor is not required to produce a safety case, it is possible to produce one as part of a safety evaluation or assessment process. In particular, a high-level assurance case can provide a strong tool for understanding how the safety efforts taken by a developer fit together to create a safe system.

## 2 SAFETY EXPERIENCE OF ASSURANCE CASES

Safety cases have been widely used in safety for 30 years. Safety cases are now mature and widely used; for example, they are required by the MOD for all equipment acquisitions (under Def Stan 00-56 [7]).

One benefit of producing a safety case, especially early in development, is that it can help engineers to think about where they need particular arguments, justification and evidence. It can also lead to early recognition of deficiencies in the system design or the process around it. For example, in one case study carried out by York, creating a safety case revealed a lack of traceability between high-level requirements and evidence about the implementation. (Similar results occurred in a security case example produced by Ankrum [8] – see section 3.1.2).

For safety cases to be effective, they have to be approached with intent to genuinely capture the safety (or not) of the system; failing that, they need to assessed by evaluators who are competent and willing to find flaws in them. A safety case produced and accepted as part of a box-ticking exercise will not be effective; if we roughly paraphrase the Nimrod report [9], one of its findings was that the Nimrod safety case had been such an activity. Kelly, in [10], identifies several types of ineffective safety case.

# 3 WHY SHOULD WE CONSIDER ASSURANCE CASES FOR SECURITY?

We explained in sections 1 and 2 that assurance cases are widespread in safety. In this section, we will review the potential for applying them in the security domain. In section 3.1 we look at whether (and how) security assurance cases can be created, in section 3.2 we discuss the differences between safety and security, and section 3.3 we look at the possible benefits from security cases, given typical security practices and current challenges. In section 3.4 we review some of the practicalities of moving to a security case approach.

## 3.1 Creating Security Assurance Cases

### 3.1.1 Methods and Guidance Available

There is already some published guidance on creating security cases. Basic advice is provided by Goodenough, Lipson and others (see [11] and [12]) as part of the “Build Security In” initiative in the USA. The advice there is not particularly detailed, but it is clear and practical. Their motive for proposing security cases is increasing complexity: they note that many people have responded to increasing system complexity by proposing an empirical, post-hoc approach (treat the developed system as a natural phenomenon, and assess its security after deployment by observing the number of security flaws uncovered). They reject this: they don’t see how it can provided the level of confidence we need for high-security systems. Their approach is aimed at vendors; they want vendors to instrument their development processes with evidence-generating activities, and use a security case to capture the result.

A more detailed process is provided by the SAFSEC standard [13], which was developed by Altran Praxis as a way to unify safety and security cases. In the SAFSEC approach, safety and security risks are treated equivalently – a combined set of mitigations is proposed and a single assurance case is produced that argues that the system will be safe *and* secure. The safety side is based on Def Stan 00-56, and the security side on release 2 of the Common Criteria. It is broadly compatible with similar work on safety-security unification by the Industry Avionics Working Group (IAWG) [14].

The SAFSEC standard provides a strong analogy between safety and security in an assurance case framework – it explains where safety-security commonality exists and provides a specific process to exploit that. As such, it could be followed directly to create a security case. It does not, however, address the cultural, epistemic and economic challenges that we discuss in Section 3.2.

### 3.1.2 Experience with Security Cases

We are not, as yet, aware of any large-scale application of security cases in practice. Obviously, given the sensitive content of security cases it is possible that some have been developed but not made public, but this is of little use to us here.

There have been a number of security cases created in small-scale case studies. In [8], Todd Ankrum of MITRE briefly outlines a case study he did for the US National Security Agency. The system was a secure enclave project by the NSA’s research division, which used several authentication systems and provided an access log. The software had been developed to Common Criteria EAL 5, used formal methods, and extensive documentation was available including a vendor-supplied Security Target document and a NSA-supplied Protection Profile.

When Ankrum and his colleagues produced a security case for this system, they found a number of problems. Despite the obvious rigour with which the project had been developed and documented, it was not possibly to fully argue that the system met its security goals. First, although most security threats could be traced to requirements that mitigated them, at least one threat had no such requirements. Second, once the threat-requirements connection was explicitly made, it was not clear that the requirements for each threat sufficiently mitigated the threat; presumably the vendor's process had not made this relationship explicit, so it had not been apparent before. Finally, Ankrum's security case attempted to argue that all security enforcing functions had their dependencies met, and it became apparent that this was not true in all cases.

Lautieri and her colleagues at Altran Praxis describe a case study in [15] of a Command and Control system, for which they produced a combined safety and security argument. Their paper explains how they create a modular case for this system, and notes that the combination of safety and security was quite acceptable to certifying authorities that were only concerned with one of them; combining the two domains did not cause a communication problem.

Ankrum has also illustrated how an assurance case can capture the implicit argument in a standard that doesn't demand an assurance case. Specifically, in [8] he briefly outlines an argument that a product has achieved Common Criteria EAL 4, by starting with the claim "Product meets EAL 4" and arguing that it does everything the Common Criteria requires to support that claim. Ankrum's approach has some weaknesses; for example, it is primarily a process rather than product argument, and it is perhaps better described as a compliance case rather than a true security case. The process he gives for creating the case is perhaps over-mechanized; realistic security case creation will involve more subjective judgement than that. It has the value, however, of showing how non-assurance-case standards can be adapted.

## 3.2 The Differences Between Safety and Security

There is no question that safety and security are separated by different goals. The open question is whether the same means (in our case, assurance cases) can be used to demonstrate achievement of those goals. Cockram and Lautieri say in [16] that the two domains *can* be served by the same assurance case mechanism: both can use cause-effect models (e.g. fault trees or attack trees), both can derive requirements to mitigate the problems thus identified, and both can argue in an assurance case that those requirements are implemented and that they will achieve the mitigation that is needed. Lautieri et al [15] also give an example where separate requirements for safety and security were merged to create a single requirement that served the needs of both domains.

Beyond the basics discussed by Cockram and Lautieri, there are a range of theoretical and practical differences that we need to consider.

### 3.2.1 Theoretical Differences

The obvious difference between safety and security is the presence of an intelligent adversary; as Anderson [17] puts it, safety deals with Murphy 's Law while security deals with Satan's Law. Safety is mostly concerned with the predictable (or random) behaviour of the non-human, non-goal-seeking world, and with adaptive behaviour by humans (e.g. seeking to make their job easier) that is not aimed at reducing safety per se (although it often does as a side effect). Security has to deal with agents whose goal is to compromise systems. These attackers may be systematically probing a system for vulnerabilities (rather than acting randomly) and may realise when they have breached one defence and move to exploit that (whereas non-human phenomena cannot respond to such feedback, and nonmalicious humans may try to undo their actions if they realise they have bypassed a defence).

Another contrast is that a lot of effort in traditional system safety has gone into assigning probabilities to basic events (e.g. random failure of a valve) and computing the probabilities of accident stemming from combinations of those events. Assigning probabilities to the action of unknown intelligent adversaries is dubious – our uncertainty there is *epistemic* (we lack of knowledge) rather than *aleatory* (due to chance).

Safety does, however, already have to worry about epistemic uncertainty. This has become very apparent over the last 30 years or so as systemic faults (faults in the design that will cause failures under certain circumstances) have overtaken random failures as the cause of accidents. Partly, this is due to the great strides that system safety has made in handling random failures through component redundancy and architectures that support it. It has also come, however, from increasing use of software – there are methods for assigning failure probabilities to software (e.g. see Bishop [18]) but they do not have great credibility.

Increasingly, software safety is approached in terms of the evidence we can generate and the confidence it gives us that the software's behaviour will be safe in context. Where the link between some evidence and the claims it supports is inadequate, we have an *assurance deficit*. The significance of assurance deficits has been recognised in safety (e.g. see [19-20]), and they may be even more significant for security because of increased uncertainty about attacker's goals, capabilities and actual actions.

Because of this high uncertainty about attacker behaviour, it is common in security evaluation to take a variety of security measures that are not responses to specific threats. These measures instead provide a degree of protection against a whole class of attacks. For example, if one process has to be given temporarily elevated privileges, it is common to give it those privileges for only the minimum amount of time. In safety, there is concern about such generic "hardening" – a safety feature added without good rationale may merely be adding to the complexity of the system, and thus increasing our chance of introducing an error and not discovering it.

Well-justified defences against common errors are, however, common in safety. For example, safety-critical programming language subsets (such as MISRA C) often forbid dynamic memory allocation, thereby showing that errors from running out of heap space cannot occur. Similarly, they often forbid recursive calls to functions, allowing the size of the stack to be bounded and thus ruling out all stack overflow errors.

Creating an assurance case can help you capture the rationale for such hardening, and thus distinguish between features that give a clear benefit, and features for which the benefit is uncertain. If you cannot justify the inclusion of a security feature (you cannot find a valid place for it in your argument), then you may be spending effort on measures with no actual security impact. Just like safety, all security is a trade-off, and every software feature has a cost (if nothing else, in the complexity of the resulting architecture and hence in the chance for an evaluator to miss a vulnerability).

Security evaluators often use static analysis for vulnerability signature detection – not to detect specific violations of known requirements, but rather to find "holes" in the system that an attacker might be able to exploit [21]. This is quite compatible with assurance cases, and can be incorporated into your argument as a complement to claims about dealing with identified threats. If you are adapting the SSEI software safety case patterns [22], this fits neatly into the place where you argue that any additional hazardous contributions at the code level have been identified. In other words, you argue that you have looked well enough for common low-level vulnerabilities that we can have adequate confidence in their absence. You can scale that "adequate" to a level that suits the security criticality of the system.

A final theoretical difference is that security-critical software often has to adapt quickly as attack patterns change [23]. This has implications for the use of assurance cases, because a case may have to change if the software changes. Even if it turns out that there is no impact on the case, the case maintainer will have to spend time reviewing the case to confirm this. Assurance cases can therefore increase the cost of making changes, and introduce delays. On the other hand, we need to understand the security impact of any changes we make; if we don't, we may fix one vulnerability but create a worse one in the process. If we have an assurance case, then we can use it to trace a low-level change up to its implications in terms of our high-level security claims.

### 3.2.2 Practical Differences

The previous section talked about the fundamental distinctions between safety and security – those that are likely to endure over time. In this section, we will look at the practical differences, which may be accidents of history; they cannot be ignored, but it may be possible to change them. A number of these distinctions were identified in a meeting between representatives of the safety and security communities in the UK – Carter reports the results of this in [23].

Perhaps the biggest practical concern is the process maturity of security-critical practice. In safety-critical software, even at lower criticality levels, mature development processes are the norm: developers use good processes for requirements management, test planning and configuration control. They monitor their processes for weaknesses, and correct them when they find them. In security-critical projects of an ordinary (commercial-grade) standard, this is not always the case. Requirements may be implicit (requiring evaluators to define their own security targets), testing may be unstructured (making it difficult to relate test schedules to specific requirements) and configuration control may be poor (making it difficult to relate review or test results to specific software versions).

The process problems in security-critical software are not necessarily unique to the domain; more likely, they are present because vendors have not had the economic incentives to do better. Poor process can make it difficult to produce an assurance case. It may be that the system is adequately secure, but because of poor process we cannot get adequate evidence of this. Similarly, it may be that we cannot build our case adequately because we cannot understand the security requirements. These are problems, certainly, but they are not problems with assurance cases: they are problems with the system being evaluated. If we cannot understand the system well enough to build an assurance case, then we are not in a position to say that it is secure. Problems with the assurance case may be the "mine canaries" that alert us to problems with the system.

It has been suggested that safety has more problems with requirements, security more with low-level defects in implementation (for example, this view was reported by Carter [23] and seems to be an assumption Lipson [12]). This could be a fundamental distinction, but it may just be a side-effect of immature processes in most security-critical software. Once your development process can reliably implement the requirements that you set out to implement, then getting the requirements right is the one place where problems can still manifest (as requirements engineering is ultimately unbounded, there will always be requirements errors). As process maturity improves in security vendors then this difference may go away.

It has been noted that there are some specific differences between software security standards and software safety standards, in terms of the development practices they demand. For example, King notes in [24] that an avionics system certified to the highest level of DO-178B [25] may not meet Common Criteria EAL 5 or above, because EAL 5 requires that the "developer mathematically prove the security properties of the [software]", whereas DO-178B does not. These are likely incidental, rather than fundamental differences, and agreement could be reached between standards (the inevitable politics aside).

Overall, we suggest that security evaluators remain aware of these practical differences, but experiment with assurance cases based on the safety case model, and see what obstacles they encounter in practice. There is a wealth of experience with assurance cases in the safety domain – where obstacles appear in security, there may already be known solutions.

A final note on practical challenges – safety-critical systems are becoming increasingly software-controlled and increasingly connected. As a consequence, there is an increasing threat of malicious outsiders causing safety-critical failures. If the security-critical software domain does not adopt assurance cases, then the safety-critical software industry will have to extend safety cases to include security. We will, therefore, need to resolve the problems above, and this will be easier if assurance cases are adopted by the security community as well.

## 3.3 Potential Benefits for Current Security Practice

### 3.3.1 Benefits for Handling Complexity

As noted earlier, the major role of assurance cases is to combine evidence from diverse analyses and show how they complement each other. For example, code review alone provides limited assurance – high assurance will require a range of complementary techniques targeted at fairly specific types of attack. This might include code review, static analysis of code-level requirements (e.g. pre- and post-conditions), static analysis for vulnerability signatures, and statistical or systematic tests.

Assurance cases provide a way to connect system-level properties (e.g. security of certain data) to low-level requirements and analyses. For example, it can be difficult for an evaluator to implicitly maintain the connection between the code in a given source file and the system-level goals to protect certain data or prevent denial of service.

When effective, assurance cases can focus evaluator attention on critical parts of the system at the expense of others. In a world of finite effort and growing software complexity, this is necessary. It is likely that a given evaluator already has some way of doing this – for example, they may concentrate primarily on security enforcing functions, rather than trying to review all of the code. Creating an assurance case allows you to put this in context and see how much assurance such a strategy really gives you.

If a vendor provides a security case, this focussing aspect allows you to treat the case as a summary of the vendor's security thinking – what they have concentrated on, and what they have not. It may be safe to assume that some vulnerabilities will appear in the aspects that they have not addressed, which means that an evaluator can concentrate on attacking those areas. This prioritisation would be difficult to do with only an implicit security case.

Where we encounter problems in assurance because of limited expertise, it may be possible to use an assurance case to "bracket" those problems ready to hand them over to a third-party expert. For example, we may have a portion of a software system that deals with radio communications according to a complex industry standard. As a specialist in software security evaluation, we may not be able to understand that part of the software well enough to assess its security. What we can do is make a number of claims about that component, claims that if true would allow us to support the overall security claims of the system. We can then ask a third-party domain expert to check those claims against the specialist software.

One tool for managing complexity in assurance is modular safety cases [26]. For example, if we have an operating system that is often used, which might create a security case module for it. This module would make certain claims about the operating system (for example, that there was definitely no way for a process to elevate its privileges without appropriate authorisation) subject to certain dependencies (for example, that all loaded device drivers could be trusted). Similarly, we might create modules for common hardware or for common support applications.

The modular case approach has been explored in other domains [16] with the aim of reducing certification update costs. Both the IAWG [14] and SAFSEC [13] processes support some degree of modularity. If an evaluator is creating assurance cases, they may be able to create modules for common OSs, hardware and software components, thus simplifying the process and reducing costs.

The security of a system obviously depends on the context that it is in. The assurance case for a system may need to change if the system's contexts changes, for example if the system needs to communicate with a new peer system. Thomas [27] raises this as a potential obstacle to the adoption of assurance cases. It is a genuine problem, although it is one faced in the safety domain as well, particularly as the complexity of safety-critical software increases. Modular assurance cases may help to deal with this challenge.

### 3.3.2 Benefits for Justifying Decisions

When an evaluator believes that a system is insecure, they can demand that the vendor generate additional evidence (e.g. run more tests), or they can demand changes to the design, or they can impose restrictions on how the system is used. If the evaluator is all-powerful then this is not a problem. Realistically, evaluators have to justify their position against claims by vendors that the change will be very expensive. If the problem is an objective vulnerability (something that can be seen e.g. in the code, and is easy to exploit) then this may be straightforward. If the concern is an assurance deficit – perhaps a lack of confidence that a certain requirement is satisfied – the justification needs to be rather more subtle. Assurance cases can help with such justification by linking analysis activities and claims about the system design to high-level security properties.

### 3.3.3 Compatibility with Typical Evaluation Processes

As noted in section 1.2, assurances cases can be created retrospectively. Indeed, if the vendor does not produce one then they can be created by an evaluator. This activity is not dissimilar to the case when an evaluator creates a security target document because the vendor did not supply one (or supplied an inadequate one). There are potential traps in this activity, particularly if the evaluator is unwilling to reach the conclusion that the system is not secure (see the Nimrod review [9] for an analogous incident in safety). The cost of creating a case retrospectively may be high, but doing so should also lead to some specific benefits: in creating a retrospective security case an evaluator creates a paper trail for their reasoning – they will capture a justification for why they think the system is adequately secure (or why it is not).

The first benefit of this that if we later need to re-evaluate the product (for example, after a major change or security incident), we know what the original evaluator did and can repeat only that which has been invalidated by the change.

A second benefit of such records is that they can provide a training tool and a mechanism of corporate memory. Junior evaluators can look at existing security cases to understand what kinds of security processes and evidence are acceptable to the evaluating organisation – they can see what was included and what was emphasized. Similarly, if an experienced evaluator encounters an unfamiliar type of system or an unfamiliar set of security requirements, they can look over past assurance cases for similar systems, and see what their peers did. There are existing techniques, such as assurance case patterns, for distilling this kind of information into a generic form.

The provision of a security case (whether by vendor or evaluator) may help coordinate teams of evaluators. The case could provide a central structure around which activity is organised – one evaluator could be assigned to each abstraction tier, or to a major component, or to a high-level requirement. They can also record their results in terms of the assurance case structure (so, for example, an evaluator working at the source code level may note problems there, and trace their significance back up to security claims made at the system level).

The costs and benefits of assurance cases are likely to vary with the level of security claimed. At lower levels, the assurance being sought may be relatively modest, which will make it easier to create a compelling security case. On the other hand, the product may be off-the-shelf, and when this is combined with low process maturity it may make it hard to create the case at all. As noted above, of course, if the evaluator requires good justification of security then inability to create a case is grounds for rejecting the product.

As we move to higher levels, the product is more likely to be bespoke and the evaluator is likely to be involved from an early stage. This may make it easier to justify a vendor-created security case, or one produced through vendor-evaluator collaboration. The risk at higher levels is that the case will need to claim a very high level of assurance, and this may be difficult to justify. As ever, if a security case cannot be made compelling, then it may well that the system is not secure.

## 3.4 Practical Aspects of Moving to Security Cases

As noted in section 3.1.2, there are no public descriptions of major applications of security cases. It is therefore difficult to say what the costs and timescales will be. However, we can draw on experiences of moving from procedural safety standards to safety case approaches. The major need is for expertise: at least the evaluators, and maybe the vendors as well, will need to learn to produce and assess assurance cases. This expertise does not need to be spread uniformly throughout the population, but a sufficient number of staff will need to be skilled at judging assurance cases and fluent in the GSN notation. There are published accounts of moving from a prescriptive standard to a safety case approach, and from moving from a textual case to a GSN case. For example, see Chinneck et al [28].

The move to assurance cases does not always have a major disruptive effect on the products being assured, or on the techniques used to evaluate them. In the first instance, assurance cases merely provide a way to relate what is already done to the goals that are already held (although those goals may previously have been implicit). Their impact will not, for example, be comparable to moving a vendor’s software development from procedural to object-oriented. What they *may* do, in terms of disruptive influence, is reveal that there are weaknesses in the products or in the evaluation of them. That is, after all, the point of an assurance case: it should allow a third party to assess whether the system has the properties that are claimed for it. When an assurance case does reveal problems, then evaluators and vendors may, of course, decide that they need to change their processes and tools. *That* may be disruptive.

If an evaluator already produces a security target document, then an assurance case is partly an extension of that - it maps threats onto requirements, and requirements onto justification of their adequacy and evidence that they have been implemented. If the vendor already follows the requirements of the Common Criteria, particularly the refinement structures required by the higher EALs, then there may be a clear mapping onto the existing SSEI safety case patterns (which use a similar structure of refinement through ‘tiers’ of development).

In terms of notation, you can create GSN diagrams using ordinary drawing software (including the tools in Word as a last resort) but for full-sized cases you will want a special-purpose tool. Perhaps the most widely used one is ASCE (http://www.adelard.com/web/hnav/ASCE/), which is provided on a commercial basis with support, and has been used to produce and manage large safety cases.

## 4 KEY QUESTIONS AND ANSWERS

**Q: Why do we need special-purpose notations such as GSN and CAE? Why can't we just present assurance cases in free-form text?**

You can present assurance cases in ordinary prose, but experience in the safety world is that these are often difficult understand – see the example in section 1.1. Structured notations make it much easier to find problems in the *structure* of the argument (and, indeed, to realise when the structure is the problem).

**Q: I already use [some analysis method e.g. code review/static analysis tools/attack trees]. Why should I do assurance cases instead?**

Assurance cases are not a way to do analysis – they're the way to summarise the results of multiple analyses and record how those results relate to higher-level claims (ultimately, to "the system is adequately secure"). They capture the *rationale* for why the results of the analyses support our high-level requirements and goals, and the context for this support (for example, the assumptions and scope of any models used).

**Q: If a vendor gives me a codebase and an associated assurance case, why should I believe that the case genuinely applies to the code?**

It's certainly possible to have an assurance case that is structurally valid, but not sound because the evidence it is based on does not exist or is not what the case claims it is. For example, a vendor could claim that they did statistical testing using $10^5$ test cases, when in fact they only did $10^3$ tests, or no tests at all.

As a first step, where there is any systematic or semi-formal design we can check the structure against that – if there is a GSN strategy that says "argument over all software modules", then we can check that it has one child goal for each module in the software design. In other words, a misleading assurance case may incriminate itself by making easily-refutable claims.

Dealing with misleading claims about evidence (as in the testing example above) is more difficult. The principal weapon here is audit – if there is suspicion that some evidence is not what it seems, the evaluator should either try to reproduce it (e.g. if they have source code, build environment, and the test scripts), or to at least witness it being reproduced at the vendor's site.

The assurance case itself can guide what we choose to audit – we can use it to work out which test sets are most significant for our top-level security claims. We can extend this with a sampling method – we could audit a series of evidence items, starting with the most assurance-significant and moving steadily onto the less significant. With each test set that passes our audit, our confidence in the trustworthiness of the overall evidence grows. Conversely, if we find some evidence that does not meet its description, then we know that the case has a problem.

Any evaluation activity applied to the source code or executable will have an implicit audit effect. If we find a vulnerability through code review, and the vendor's safety case claims analysis activities that should have found that vulnerability, then we have good reason to suspect the evidence from their analysis.

Trustworthiness of evidence is an issue in safety too, but it can be managed, provided that the regulator in question has the authority and technical ability to genuinely evaluate the safety case. For an illustration of where this did *not* occur, consider the MOD's handling of the Nimrod safety case (see [9]) – ultimately the safety case was accepted by a group of MOD employees who did not have the time and technical skills to review what they were presented with. The MOD had engaged an independent safety assessor (QinetiQ), but it did not act adequately in this role.

**Q: Don't safety arguments rely on probabilities of events?**

Good safety arguments invoke probabilities of failure for e.g. random electronic failures, but do not assign probabilities to systemic faults in software. In this respect, the situation in modern safety-critical projects is not dissimilar to that of security-critical ones.

## 5 CONCLUSIONS

We (York) believe that adopting security assurance cases will be beneficial to CESG, and will increase the rigour of the security evaluation that you do. We believe they will help you scale your efforts as the complexity of software increases. We believe that they will have benefits in training, in corporate memory, and in your ability to justify your responses to vendors.

Certainly, there will be costs to adopting assurance cases, and there is an element of risk here. Assurance cases are widely used in safety, but there has been limited used in security thus far. In order to get benefits from assurance cases, CESG will need to develop significant in-house expertise, and find solutions for assurance case challenges that are unique to the security world. CESG will not be the only organisation doing this, however, and much more guidance on security assurance cases will appear over the next few years.

We suggest that you proceed by creating an assurance case for a small project at Baseline security level. You should carry this out in parallel with your standard evaluation activity, so as not to impair your evaluation, but you should allow interaction between the case creators and the regular evaluators so that you can benefit from any insights that the assurance case provides. This will obviously be a retrospective rather than vendor-supplied security case, but if you do adopt assurance cases you will be working with retrospective cases for some time. York can provide consultancy to help with this activity, but it is important that CESG staff do the bulk of the work in order to build in-house skills. The most useful role for York would be initial training and periodic review.

## 6 ACKNOWLEDGEMENTS

The authors would like to thank Andrew Rae and Patrick Graydon for their help in preparation of this report.

## 7 REFERENCES

[1] Kelly T. Arguing Safety - A Systematic Approach to Managing Safety Cases [PhD Thesis]: University of York; 1998.

[2] Hawkins R, Kelly T. A Structured Approach to Selecting and Justifying Software Safety Evidence. Proceedings of the 5th IET System Safety Conference, Manchester, 2010.

[3] Toulmin S. The Uses of Argument: Cambridge University Press; 1958.

[4] Kelly T, Weaver R. The Goal Structuring Notation - A Safety Argument Notation. Proceedings of the Dependable Systems and Networks 2004 Workshop on Assurance Cases, 2004.

[5] Bishop P, Bloomfield R, Guerra S. The future of goal-based assurance cases. Proceedings of the Workshop on Assurance Cases - supplemental Volume of the 2004 International Conference on Dependable Systems and Networks, 2004.

[6] Object Management Group. Argumentation Metamodel (ARM). 2010. http://www.omg.org/spec/ARM/

[7] MoD Interim Defence Standard 00-56 Issue 4 - Safety Management Requirements for Defence Systems. Ministry of Defence; 2007.

[8] Ankrum TS, Kromholz AH. Structured Assurance Cases: Three Common Standards. 2005. www.asq509.org/ht/action/GetDocumentAction/id/2132

[9] Haddon-Cave C. The Nimrod Review. 2009.

[10] Kelly T. Are 'Safety Cases' Working? Safety Critical Systems Club Newsletter. 2008;17:31-3.

[11] Goodenough J, Lipson H, Weinstock C. Arguing Security - Creating Security Assurance Cases. 2007. https://buildsecurityin.us-cert.gov/bsi/articles/knowledge/assurance/643-BSI.html

[12] Lipson H, Weinstock C. Evidence of Assurance: Laying the Foundation for a Credible Security Case. 2008. https://buildsecurityin.us-cert.gov/bsi/articles/knowledge/assurance/973-BSI.html

[13] Dobbing B, Lautieri S. SafSec Methodology: Standard. 3.1 ed, Altran Praxis; 2006.

[14] Short Study: SafSec Coherence. Industrial Avionics Working Group; 2007.

[15] Lautieri S, Cooper D, Jackson D. SafSec: Commonalities Between Safety and Security Assurance. Thirteenth Safety Critical Systems Symposium, Southampton, 2005.

[16] Cockram TJ, Lautieri SR. Combining Security and Safety Principles in Practice. 2nd IET International Conference on System Safety, London, 2007.

[17] Anderson R. Security Engineering: A Guide to Building Dependable Distributed Systems. 2nd ed: John Wiley & Sons; 2008.

[18] Bishop P. SILs and Software. Adelard and Centre for Software Reliability, City University.

[19] Hawkins R, Kelly T. Software Safety Assurance - What Is Sufficient? Proceedings of the 4th IET System Safety Conference, London, UK, 2009.

[20] Hawkins R, Kelly T, Knight J, Graydon P. A New Approach to Creating Clear Safety Arguments. Proceedings of the Safety-critical Systems Symposium, 2011.

[21] Gutgarts PB, Temin A. Security-critical versus safety-critical software. IEEE International Conference on Technologies for Homeland Security (HST), Waltham, MA 2010.

[22] Hawkins R, Kelly T. A Systematic Approach for Developing Software Safety Arguments. Proceedings of the 27th International Systems Safety Conference, 2009.

[23] Carter A-L. Safety-Critical Versus Security-Critical Software. British Computer Society; 2010.

[24] King T. Two different realms: RTOS support for safety-critical vs. security-critical systems 2009. http://www.vmecritical.com/articles/id/?4031

[25] RTCA. DO-178B: Software Considerations in Airborne Systems and Equipment Certification. 1999

[26] Kelly T. Using Software Architecture Techniques to Support the Modular Certification of Safety-Critical Systems. Proceedings of the Eleventh Australian Workshop on Safety-Related Programmable Systems, Melbourne, Australia, 2005.

[27] Thomas R. Software Assurance Using Structured Assurance Case Models. Journal of Research of the National Institute of Standards and Technology. 2010.

[28] Chinneck P, Pumfrey D, Kelly T. Turning Up the HEAT on Safety Case Construction. Proceedings of the Twelfth Safety-critical Systems Symposium, 2004.