CHERI Standards Compliance (STONE)
10004570
Final Report
for open publication

Document Number	Issue	Date
D-RisQ/CHERI STONE/D1	1.0	28th Sep 2021

	Name	Date
Originator	Nick Tudor (D-RisQ)	

DISTRIBUTION		
Name	Establishment	Copy Number
Originator	D-RisQ Ltd	1

REVISION HISTORY		
Revision	Date	Detail
A		Document Creation
0.5	8th Sep 2021	Draft for comment
1.0	28th Sep 2021	First Issue

Table of Contents

1 Object of this Document

This document is the external deliverable on CHERI Standards Compliance (STONE). It provides an opinion on the application of the CHERI architecture software (firmware) in aerospace safety and security applications. The assessments were under taken against DO-178C [1] and DO-326A [2]. We assessed the software against DO-178C and DO-326A because they are exemplars of best practice and consequently might help provide insight to improvements to development in software that could be developed for the new CHERI architecture.

2 References

Identifier	Reference
[1] DO-178C	RTCA, Software Considerations in Airborne Systems and Equipment Certification, Issue C, RTCA DO-178C, 13th December 2011. Also known as EUROCAE ED-12C
[2] DO-326A	RTCA, Airworthiness Security Process Specification, RTCA DO-326A June 2014. Also known as EUROCAE ED-202A
[3] DO-356A	RTCA, Airworthiness Security Methods and Considerations, RTCA DO-356A June 2018. Also known as EUROCAE ED-203A.
[4] DO-355	RTCA, Information Security Guidance for Continuing Airworthiness', RTCA DO-355 June 2014. Also known as EUROCAE ED-204.
[5] CS-25	Certification Specifications and Acceptable Means of Compliance for Large Aeroplanes CS-25
[6] Woodruff et al.	The CHERI capability model: Revisiting RISC in an age of risk. Proceedings of the 41st International Symposium on Computer Architecture (ISCA 2014). Minneapolis, MN, USA.
[7] Wilkes & Needham	The Cambridge CAP Computer and Its Operating System. Elsevier, 1979.
[8] Foster et al.	An Introduction to the FLEX Computer System. Malvern, UK: Royal Signals and Radar Establishment, 1979.
[9] NIST Apple	*CVE-2014-1266.* Retrieved from https://nvd.nist.gov/vuln/detail/CVE-2014-1266
[10] The Verge	The dangers behind Apple's epic security flaw. Retrieved from https://www.theverge.com/2014/2/24/5442576/inside-apples-epic-security-flaw
[11] NIST Heartbleed	*CVE-2014-0160.* Retrieved from https://nvd.nist.gov/vuln/detail/CVE-2014-0160
[12] DO-330	RTCA, Software Tool Qualification Considerations, RTCA DO-330, January 2012. Also known as EUROCAE ED-215.
[13] DO-333	RTCA, Formal Methods Supplement to DO-178C and DO-278A, RTCA Jan 2012. Also known as EUROCAE ED-216.

Identifier	Reference
[14] TF-A Assessment	An Assessment of Trusted Firmware-A Firmware against DO-178C, Software Safety Ltd /D-RisQ Ltd, September 2021
[15] SCP Assessment	An Assessment of System Control Processor Firmware against DO-178C, Software Safety Ltd /D-RisQ Ltd, September 2021

3 List of acronyms / definitions

Term	Definition
API	Application Programming Interface
BSD	Berkeley Standard Distribution
CAP	Capability Protection
CHERI	Capability Hardware Enhanced RISC Instructions
DSbD	Digital Security by Design
FAA	Federal Aviation Administration
FVP	Fixed Virtual Platform
iAPX	Intel Advanced Performance Architecture
IIS	Internet Information Services
ISA	Instruction Set Architecture
ISCF	Industrial Strategy Challenge Fund
MC/DC	Modified Condition/Decision Coverage
MCP	Manageability Control Processor
NIST	National Institute of Standards and Technology
RISC	Reduced Instruction Set Computer
RSRE	Royal Signals and Radar Establishment
SCP	System Control Processor
SHA	Secure Hash Algorithm
SLOC	Source Line of Code
SQL	Structured Query Language
SSL	Secure Sockets Layer
TF-A	Trusted Firmware-A
TLS	Transport Layer Security

4 Introduction

The CHERI project has produced an Instruction Set Architecture which provides a means for managing memory safety through the introduction of 'capabilities'. The Morello programme run by Arm is producing an example, prototype hardware board with a firmware stack to enable use of the hardware. This firmware will need to be assessed as part of any use in a regulated industry. As the clearest and most mature set of standards are in aerospace and support airworthiness certification, it was decided to use these as the basis for assessment of the Morello software in regard to both safety and security. Since 1992, the airworthiness of software has been focussed on safety using RTCA/DO-178B/C [1] but increasingly the security aspects of airworthiness has been more of a concern and RTCA/DO-326A [2] and related documents often has to be met. The project could not assess every aspect of the available firmware. The 2 sets of Morello software selected for assessment using standard aerospace practices were Trusted Firmware-A and SCP. The approach has been to use the Federal Aviation Agency (FAA) Job Aid for assessment of the software against DO-178C. There is no equivalent for DO-326A, so an informal compliance assessment will be undertaken. Additionally, the software available in the Morello Fixed Virtual Platform (FVP) was to be used in experiments.

5 Introducing the Assessors

Nick Tudor is the CEO of D-RisQ Ltd, a company based in Malvern that is focussed on the development and use of automatic formal methods based verification technologies. Nick has been involved in a wide range of safety and security critical projects for over 20 years following a successful career in the RAF as Officer Engineer. He was a key contributor in the development of DO-178C and the associated suite of documents with an emphasis on the development of the new Formal Methods Supplement to DO-178C (DO-333). He has led a number of projects in aerospace, autonomous systems (in maritime - surface and subsea - air and nuclear decommissioning), automotive, cyber-security and medical devices. Additionally, he has been contracted to provide assessment of software for compliance to DO-178C for various government and commercial entities and has been called upon to assist in solving issues in software development in the rail sector among others. It was decided that the effort required for this project would require assistance. D-RisQ contracted Mr Dewi Daniels (MD Software Safety Ltd) to assist in the project. Dewi has had a long career in safety critical software in various companies and on projects in aerospace, rail and other sectors. He was also one of the key contributors to the development of DO-178C. Both Nick and Dewi were invited as 2 of only 4 UK nationals onto the advisory panel post the issue of the DO-178C. Together they regularly provide training in DO-178C to individuals, companies and government agencies.

6 Why D-RisQ is Interested in CHERI

6.1 D-RisQ Background

The Founders and team in D-RisQ all have a background in high integrity software and systems covering both safety and security. The basis of all our work has been in the exploitation and use of Formal Methods, particularly in building tools for our own use as well as licencing them to 3rd parties.

6.2 Introduction to Formal Methods

“Program testing can be used to show the presence of bugs, but never to show their absence” is probably the most famous quote by Edsgar Dijkstra, a Dutch mathematician and widely acknowledged as the author of the term ‘Formal Methods’. His point being that the program only is shown to work under the conditions of the test; should the conditions change, then other tests are needed to show that the program still works as expected. An alternative to test is proof which is the exploitation of the mathematical discipline of Formal Methods. If the program specification can be described in a mathematical manner, then it can be proven to work without the need for test. Indeed, Alan Turing in 1949[1] produced a paper in which he stated that he did not need to test the program he had written since he could prove that it would always work. While the term Formal Methods has been around since the late 1960’s, until this century, it was very much an academic discipline with only a few industrial examples. With the development of more powerful and accessible computing as well as evolution of the mathematics, automation has become possible, thus removing much of the need for deep knowledge of the techniques.

6.3 Adoption

With the combination of cheaper memory and faster processing, the mathematics behind FM could accelerate. Indeed, since the very costly Intel floating point bug of 1998, FM have increasingly been used in a wide variety of both hardware and software developments. In 2011, RTCA and EUROCAE published an update to the aerospace certification standard DO-178C/ED-12C and alongside this published the Formal Methods Supplement to DO-178C/ED-12C in reaction to a rise in the use of FM in aerospace applications. Since then, the likes of Google, Facebook, Amazon, Microsoft have all invested heavily

[1] Jones, C.B.: Turing's 1949 paper in context. In: Kari, J., Manea, F., Petre, I. (eds.) Computability in Europe 2017. Springer, LNCS, vol. 10307, pp. 21{41 (2017).

in recruiting FM expertise. Within D-RisQ we have also taken advantage of the changing FM landscape, indeed, we are initiating some of these changes.

6.4 Automation

The power of FM can be best be exploited through automation, otherwise dependence will be on highly skilled individuals. Indeed, the very definition of FM means that it can be done by a machine, hence the ultimate aim is automation. The usefulness of the output is a key consideration, so when the verification fails to provide a proof, the automation must provide results that can be meaningful and useful to a typical developer. The approach adopted by D-RisQ has always been to exploit FM where it is practical to do so and not at the cost of delivery of capability. Where FM are used, we automate their use so that results are repeatable, auditable and wherever possible, with the intention that non-experts can use the technologies. The main criteria we use for the use of FM is can we save time by their use, either in the initial development or/and because we will need to use the technique again in further iterations. In this way we avoid the subjectivity of reviews and do not need to develop swathes of test cases that can also be incomplete or in error. The use of FM has the side benefit that it makes specifications much more complete, precise and crucially, verifiable from the outset. When all these benefits are combined the result is a much more efficient way of working. It means we can produce highly reliable results in short timescales while also tackling very large, complex problems. In independent industrial scale blind trials of one of the D-RisQ technologies showed that it was feasible to make 80% savings over traditional techniques such as review and test by typical users.

6.5 Tools

The tool suite developed by D-RisQ is centred on assured development of software. This includes both safety and security and compliance to standards, such as DO-178C, ISO 26262, IEC 61508, etc. As such the premise behind CHERI appeals to the ethos we have within D-RisQ: software should only do that which is required; and nothing else. More on our tools and other developments can be found on our website: www.drisq.com.

7 A Short History of Capability Architectures

The University of Cambridge is developing a capability architecture called CHERI (Capability Hardware Enhanced RISC Instructions) [6]. CHERI is based on pioneering work carried out at by Maurice Wilkes and Roger Needham at the University of Cambridge in the 1970s on capability architectures, culminating in the creation of the Cambridge CAP computer [7]. Another early capability machine was the FLEX computer system developed at the Royal Signals and Radar Establishment (RSRE) [8].

Pointers are inherently insecure and a well-known source of security vulnerabilities. In a capability architecture, pointers are replaced by references to protected objects. These references are called capabilities. Capabilities can only be created using privileged instructions, which may be executed only by the operating system kernel or by some other privileged process authorized to do so. This greatly increases security, eliminating many common security vulnerabilities such as the exploitation of buffer overflow, array bounds violations or dangling references.

In the early 1980s, it looked as if capability architectures would become mainstream, with the introduction of the Intel iAPX 432 in 1981. However, early microprocessors were relatively slow, and the market at that time turned out to value the higher performance of conventional microprocessors such as the Motorola MC68000 and Intel's own 80286 over the greater security offered by the iAPX 432. The iAPX 432 was a commercial failure and was discontinued in 1986.

Capability architectures mitigate many common security vulnerabilities, such as buffer overflow. The first documented exploitation of a buffer overflow was by the Morris worm in 1988. Other examples of computer worms that exploited buffer overflows were the Code Red worm, which attacked computers running Microsoft's Internet Information Services (IIS) web server in 2001 and the SQL Slammer worm which attacked computers running Microsoft SQL Server in 2003.

In 2021, desktop computers, laptop computers, smartphones, and even smart home devices such as thermostats and smart lights can take advantage of cheap and powerful microprocessors with large amounts of memory. Given the pervasiveness of connectivity, such as the Internet of Things, focus is increasingly turning from performance to security. It is therefore hoped that novel architectures such as CHERI will result in greatly increased security.

It is therefore hoped that the time is now right for capability architectures, that CHERI will be a commercial success and that it will result in greatly increased security .

Arm is leading a research program called Morello, which is partially funded by the UK government's Industrial Strategy Challenge Fund (ISCF) as part of the Digital Security by Design (DSbD) program. The main output of DSbD will be a technology platform prototype called the Morello board, which is based on the CHERI architecture. As part of the Morello program, Arm is developing a software stack that is intended to be an exemplar of secure software running on the Morello board. While the overall aim of the DSbD project is not to fix bad software practices, this could be a long term side effect. This is because as new software is developed using poor practices it would more often fail to behave as needed as the capabilities intervened. This would probably result in rising costs for developers to fix problems that should have been foreseen, thus gently encouraging a better way of developing software. Note, however, that this is a long term possibility.

8 The Assessment Process

8.1 DO-178C

8.1.1 Software and Verification

DO-178C/ED-12C is a document suite that provides guidance on the evidence requirements for software in certification. For large aircraft, this is an 'Acceptable Means of Compliance' to CS-25 [5]. DO-178C sets out over 70 objectives that may need to be met by a systems development, depending on the integrity level for the system. These objectives can be augmented by demand from the regulator or as suggested by the Applicant. The objectives are selected by software Level and reduce in number from A to D. There are also requirements for independence in various objectives and these also change with software level; the more critical the system. the more independence is required. In all cases there is a need for independence for Software Quality Assurance. DO-178C/ED-12C is a 'requirements-based document'. This means that it expects evidence that the software requirements have been developed properly from system requirements, i.e., that the development actually implements these software requirements with evidence supplied through verification activities meeting objectives. Requirements cover every aspect of functional, safety, security, performance and anything else that needs to be delivered by the software development. It expects proper configuration control, the involvement of independent software quality assurance, the production of plans, records and other documentation as well as a well constituted change control and authorisation process. In summary, it expects the developer to be able to describe what the software is intended to do, show that it does it and nothing else.

8.1.2 Audit

In the 1990's, the FAA produced a series of documents that formalised the process of assessment of airborne software as part of the certification process, known as the 'Job Aid'. It was based on the use of DO-178B (the EUROCAE version is referred to as ED-12B). It has 4 Stages of Involvement (SOI) that cover Planning (SOI#1), Development (SOI#2), Verification (SOI#3) and a final assessment (SOI#4). The intention was that the guide was a more standardised approach to assessment with the aim of bringing consistency across a wide range of avionics software. The assessment guide was intended to be used by licenced assessors (called Designated Engineering Representatives - DERs) who had to have regular training. Unfortunately, the use of the guide had become somewhat abused and has now been withdrawn by the FAA but exists in many areas as the standard approach. As such it was not updated to cater for the changes made by the refresh to DO-178C. The guide needs to be used with care, can be interpreted based upon experience of software engineering as well as the use of DO-178C and therefore judgement is required; it is not a tick list. Given that the Job Aid is a standard approach and we

have considerable experience of DO-178C coupled with over a decade of assessment on various safety critical projects, this was the approach adopted.

8.1.3 Evidence Based on Process

The audit looks for evidence produced as a by-product of undertaking software engineering starting and centred on clear requirements. While no particular life cycle is needed, there will be expected to be requirements for the software, design, coding and integration including the compilation. It is expected that there will be evidence that the transformation throughout development is verified to be complete, accurate and consistent using standards for requirements, design and code as well and maintenance of configuration control. This is simply good software engineering practice and it is therefore reasonable to expect to see such practice being followed by a programme seeking to reduce the opportunity for cyber-attack, such as CHERI/Morello. At different integrity levels, various objectives have to be met with some requiring independent views (by people or process/tools). The outcome from the assessment is an independent view that what was written in the requirements is verified to be in the Executable Object Code, with no extra code (known as 'dead code'), that will only do what is required and, crucially from both a safety and security perspective, nothing else. The audit starts with Plans (special focus on the Software Development and Verification Plans, as well as Configuration Management and Quality (or process) Assurance Plans)[2] and standards (for Requirements, Design and Code) in SOI#1. The audit process is looking for evidence of compliance to DO-178C (actually, simply good software engineering practice). The next step is looking for evidence that the development is being undertaken in accordance with the plans (or updated plans), that requirements are clear consistent, traceable and can be used in verification; this is SOI#2. The design and software architecture as well as coding is proceeding correctly, that reviews and analyses are being conducted and corrective actions are being recorded with changes being managed in accordance with the approved plan. The verification and remaining aspects of design and code are assessed in SOI#3 to check that the development is producing the required evidence with coverage of requirements, design and code structure being undertaken. The final SOI#4 audit wraps up various archiving and retrieval processes, assesses completeness of the development among a few other items, such as ensuring all audit actions are complete. Done properly, the use of DO-178C will save both time and money; the opposite is regrettably often true.

8.2 DO-326A and Associated Documents

Since the mid-2000's, the RTCA SC-216/EUROCAE WG-72 committees have been working on the production of a suite of documents related to airworthiness and security of systems from a cyber perspective. To date there are 3 documents covering design (DO-326A/ED-202A) [2], assessment (DO-356A/ED-203A) [3]and on-going maintenance (DO-355A/ED-204A) [4] with updates and more supporting documents planned.

While this is a relatively new set of process-oriented documents and the guidance to certifiers is still evolving, security has always been a part of the regulators remit. The rather large amount of guidance covers the scope of cyber-security including risk assessment, threat conditions, effectiveness, architecture, measures, verification, techniques for provision of evidence and maintenance of security over time. The 3 documents give guidance for systems cyber security considerations. While there is a link to software, they are very much focussed at the systems level. The focus from a regulatory perspective will also be focussed on CS-25.1309 (for large aircraft) among a number of other relevant areas related to systems.

[2] In aerospace, it is also expected that a Plan for the Software Aspects of Certification (PSAC) will also be developed as the overview of how 'certification' of the software will be achieved. It is the outline or parent plan that is used to communicate to the assessor.

9 The Assessments – DO-178C

9.1 The Audit Subjects

We conducted separate audits on the Trusted Firmware-A (TF-A) and Systems Control Processor using the FAA Job Aid. This paragraph is intended to highlight the areas that would fall short of compliance as well as highlight areas of good practice. As the software was never intended to meet the needs of a formal assessment such as this, it was to be expected that a number of areas would fall short. However, this report can indicate where gaps could be filled in should this be required in future. We held short meetings with Arm specialists on two occasions to seek clarification on a number of technical points. In these meetings a number of issues were discussed ranging for our motivation for the project to the pros and cons of 'open' development. Our thanks to Arm for being available and willing to help in such frank and helpful manner.

9.2 A Typical Audit?

This was not a typical audit. The approach normally would rely upon the Applicant to provide a lot of material to the auditor prior to an on-site meeting, either by provision or by request. The material would have been reviewed and a preliminary view made prior to a meeting where material can be discussed, expanded upon and an agreed determination of compliance made. A desk review for each SOI can be a number of weeks and a meeting typically lasts between 2-3 days. While the material available in the various repositories was useful, we had no reasonable means of requesting further information. We managed a number of short video calls but no opportunity to have a meaningful meeting on any of the SOI questions. This was never intended to be part of the project due to lack of time and funding, let alone an agreed certification context (ie an actual aeroplane project). As such, it is to be expected that there would be shortcomings in the audit, but that potentially useful outcomes may result.

9.3 Trusted Firmware-A (TF-A) Assessment

9.3.1 Use of TF-A

This firmware is intended for use by the manufacturers of Arm processors and typically it is used without or with only minor modification. It therefore has had considerable usage over many years. TF-A provides a secure enclave for security related code. This enclave could also be used in safety related partitioning. It is strongly separated from the main CPU execution threads. The team that supports this firmware is very experienced and the Git repository provides a reasonable amount of documentation.

9.3.2 Resources

Trusted Firmware-A (TF-A) is an open-source project to provide a reference implementation of secure world software for Arm processors. The TF-A project can be found at https://www.trustedfirmware.org/projects/tf-a/. The code is stored in a git repository at https://git.trustedfirmware.org/TF-A/trusted-firmware-a.git/. The tests are stored in another git repository at https://git.trustedfirmware.org/TF-A/tf-a-tests.git/.

9.3.3 Plans and Development Standards

The TF-A team makes no claim that TF-A complies with DO-178C. It is therefore not surprising, for example, that there is no Plan for Software Aspects of Certification (PSAC). This is a planning document required by DO-178C that sets out how the software development will comply with the objectives of DO-178C. However, there is no Software Development Plan or Software Verification Plan and there are no Requirements or Design Standards but there are Coding Style and Coding Guidelines documents. There is a Software Configuration Management Plan (called the 'Contributor's Guide, Project Maintenance Process), but no Software Quality Assurance Plan and hence no planned QA activities could be identified. There was also no Tool Qualification Plan, so any reliance on the output of tools for certification credit would be suspect. The lack of plans and standards makes it difficult to find material that would help with an assessment and provide confidence that the development was, and would continue to be, carried out in a sound manner. From a safety perspective and, for that matter, a security one too, there is no identification of the hazards that need to be mitigated through the development. This means

that although subsequent claims for higher security might be made, there is a lack of specific justification for how this might have been achieved.

9.3.4 Development

There are no explicit software requirements which presents a number of issues with trust in the software. While the intended function can be inferred from the documents describing the Application Programming Interface (API), individual 'requirements' are not labelled and there is no traceability from the 'requirements' to the source code. The trust issues this presents are as follows:

a. Without an explicit set of statements of what the software is to do, but more fundamentally, what it is not to do and what the behaviour is to be when (not 'if') something goes wrong, it is difficult to then find evidence to match these requirements. It is the case that with some effort, the intended function can be inferred from the descriptions of the API functions, but this can be subjective and hence is insufficiently rigorous.
b. Without requirements, tracing to functionality in either design and the code to be sure that future verification activities are aligned to the correct artefacts and make suitable claims is difficult.
c. If there are no explicit statements of required, not required and error behaviour, it is somewhat subjective to claim that any test or other verification activity has provided any assurance of behaviour.
d. Without bi-directional traceability from requirements, through design to code and to the relevant verification or test cases, it is not simple to claim the right result against the right code.
e. Without bi-directional traceability it is not possible to measure coverage of requirements or structure of the code potentially leading to not only unverified aspects of code that is needed but opens the possibility for extra code for which there is no requirement or verification. The lack of traceability introduces the risk that required functionality could have been implemented incorrectly or not at all, making TF-A vulnerable to defects like the one reported in Apple Secure Transport, which failed to check the signature in a TLS Server Key Exchange message [9][10] and hence potentially introducing both safety and security vulnerabilities. While the CHERI architecture is effective at preventing buffer over-reads, such as the one that caused the Heartbleed bug [9][11], it cannot mitigate the omission of intended behaviour, which caused the Apple Secure Transport vulnerability.

In line with many such open-source projects, the focus is very much on coding and it is therefore left to developers and code reviews to determine how much (if any) test evidence is required to justify that the code satisfies its intended function. This approach is very weak from an assurance perspective, safety or security. Furthermore, it means an over-reliance on experience and knowledge which is not easily transferred as people move on. This leads to cost and time issues with the support of TF-A because it is harder to maintain without either the experience or the need to understand quite detailed code with little description of the intention. However, it is to the TF-A project's credit that a separate test team has been established to develop bare metal tests to exercise TF-A. That said, while there are 15 maintainers and 119 code owners listed in the TF-A design document, there are only 9 maintainers listed in the TF-A test document. The test documentation admits that the tests do not cover 100% of the TF-A code and it appears that the structural coverage achieved by the TF-A tests has not been measured.

9.3.5 Design

The approach taken by DO-178C is to have as many opportunities for error detection as possible. It is typically expected that a design step after the requirements phase is undertaken before coding begins. This sets out the architecture and the low-level requirements[3,4] (or modules) to be coded, how they interface with each other, timing and other performance detail.

[3] DO-178B introduced the term 'low-level requirements' to mean modules of code or other such interpretations of sections of code used within the software architecture. This historic term was unfortunately required to be used in DO-178C for backwards compatibility. Design covers software architecture and the modules contained therein also known as Low level Requirements.

[4] An informal definition of low-level requirements (and architecture) is the description from which source code can be developed without further information.

This step seems to be largely missing. The problem then becomes the lack of orthogonal verification activities whereby 'black box' tests could be carried out of the code against the requirements (noted – largely missing) and 'white box' testing whereby code is verified against the design. The lack of formal design means that the project loses the opportunity for black box testing.

9.3.6 Independence

It is well known that using an independent person or tool to assist with verification is highly productive and it is good to see that all contributions must be independently reviewed. It is also good to see that the Coverity static analysis tool assists with the enforcement of the MISRA C coding rules, though it was not clear which rules specifically apply. As stated above, there do not appear to be any guidelines for sufficiency of verification.

9.3.7 Verification

Verification in DO-178C means either Review, Analysis or Test. Reviews are manual, subjective activities that typically have a checklist to aid with consistency. Analysis[5] is meant to be 'static analysis' but is often mis-used in the context of 'analysis of the results' which is actually a review. Test is the dynamic exercise of code to demonstrate presence of correct behaviour. Of course, as Dijkstra is famous for saying, "testing can be used to show the presence of bugs, but never to show their absence". The completeness of the verification is measured in 2 ways: coverage of requirements and coverage of the code structure.

The absence of clearly defined requirements and traceability presents a problem in the measurement of coverage of requirements by test cases that also lack trace information. The structural coverage achieved by the TF-A tests has not been measured, although the test documentation admits that the tests do not cover 100% of the TF-A code. The audit was able to find several API functions that are never invoked from the TF-A tests. For example, function `plat_is_my_cpu_primary` is never invoked. It is not known what percentage of the TF-A code has been exercised by the TF-A tests.

DO-178C requires statement coverage at Level C, decision coverage at Level B and modified condition/decision coverage (MC/DC) at Level A. Structural coverage analysis is useful because:

1. It ensures that all the code has been exercised.
2. It establishes the thoroughness of the test suite.
3. It can help to find dead code and unintended code.

In other words, to ensure that all requirements are correctly implemented through design into the code and that all the code that is present is supposed to be there and does nothing else. There are many open source and commercial tools available that measure structural code coverage. It is recommended that TF-A adopts one of these tools so they can determine what percentage of the TF-A code is exercised by the TF-A tests and to ensure there is no unreachable code. See Appendix B Example Vulnerability Detectable by Coverage Analysis for an example of why this approach would be useful and enhance confidence in security.

9.3.8 Software Quality Assurance

According to DO-178C, Software Quality Assurance (SQA) provides confidence that the software life cycle processes produce software that conforms to its requirements by assuring that these processes are performed in compliance with the approved software plans and standards. DO-178C requires SQA to be applied with sufficient independence and authority to ensure corrective action. There is no SQA involvement in the TF-A project. The maintainers are responsible for ensuring the integrity of TF-A. It would add confidence if the TF-A project were subject to SQA oversight to provide an independent assessment of the software life cycle processes and their outputs.

---

[5] Analysis is actually meant not just static analysis but specifically to allow the use of formal methods for which there is a specific Supplement DO-333 (ED-216) [13]. This was set out in section 6.3 of DO-178B and is repeated in DO-178C.

9.3.9 Tool Qualification

DO-178C requires a software tool to be qualified if DO-178C processes are eliminated, reduced, or automated by the use of the tool without its output being verified. The reason that tool qualification is needed is to assure the user and regulator that the tool provides the relevant assurance of its outputs and therefore can be used to claim appropriate certification credit. The tool qualification is carried out in accordance with DO-330 [12], which is also published as ED-215. Tool qualification does not have to be onerous. DO-330 defines five Tool Qualification Levels (TQL-1 to TQL-5). Most software verification tools are TQL-5, the least rigorous level. Since TF-A makes no claim for compliance with DO-178C, it is not surprising that no tools have been qualified. However, it is noted that the test suite includes tests to verify the core features of the test framework.

9.3.10 International Standards

TF-A is intended to be a reference implementation of secure world software and to form the foundations of a Trusted Execution Environment (TEE). However, it does not claim compliance with any cybersecurity standard or indeed any best practice with the exception being the claim to use MISRA C guidelines. It would be useful if the TF-A documentation were to identify relevant cybersecurity standards and discuss how TF-A could be extended to comply with those standards.

9.3.11 Conclusion to TF-A Assessment

As open-source projects go, TF-A is much better than most in terms of the compliance to DO-178C. However, it falls short in a number of crucial areas that can be easily remedied should the desire be there. Indeed, some of these shortfalls identified in this report would probably save developers time and money and remove the over-reliance on knowledgeable, experienced personnel. The detailed results for the 3 stages of audit carried out under this project can be found in [14][15] and a summary of the results can be found at Appendix A.

9.4 System Control Processor (SCP) Assessment

9.4.1 Use of SCP

The SCP development for Morello is simply an example of how to develop firmware for the application. Arm licensees that develop the hardware will often develop their own SCP as this provides a differentiator for their own processor implementation. This provides not only a marketing differential with their own Intellectual Property but also the ability to flex processor performance as needed by their customer base. It is therefore not intended to be used as provided in the open repository. However, there are instances where some developers are using SCP without alteration. As SCP is not intended for wide use there is less of an imperative to provide documentation and other support as this would mean significant wasted effort.

9.4.2 Configuration Management

SCP Firmware is an open-source project to provide a software reference implementation for the System Control Processor (SCP) and Manageability Control Processor (MCP) components found in several Arm Compute Sub-Systems. The master repository for the SCP Firmware project is at https://github.com/ARM-software/SCP-firmware. The Morello project has created a second repository at https://git.morello-project.org/morello/scp-firmware. The relationship between these two repositories is unclear. The contents are not the same, yet both readme.md files are the same. It was not possible to documentation specific to the Morello repository. Following the instructions in the user_guide.md file in the Morello repository results in the code being fetched from the master repository rather than the Morello repository, which was confusing. The master repository is at version 2.8, while the Morello repository is at release 1.2 (although no documentation relating to release 1.2 could be found). It is suggested that the relationship between the two repositories needs to be defined more clearly.

9.4.3 Plans and Development Standards

The SCP Firmware team makes no claim that SCP Firmware complies with DO-178C. It is therefore not surprising, for example, that there is no Plan for Software Aspects of Certification (PSAC). This is a planning document required by DO-178C

that sets out how the software development will comply with the objectives of DO-178C. However, there is no Software Development Plan or Software Verification Plan and there are no Requirements or Design Standards but there are Coding Style and Coding Guidelines documents. There is an issue tracker at https://git.morello-project.org/morello/scp-firmware/-/issues, but it contains no issues. The main issue tracker is at https://github.com/ARM-software/SCP-firmware/issues, which contains 8 open and 2 closed issues. It is suggested that baseline management should be improved and a Software Configuration Management Plan written. There is no Software Quality Assurance Plan and hence no planned QA activities could be identified. There was also no Tool Qualification Plan, so any reliance on the output of tools for certification credit would be suspect.

The lack of plans and standards makes it difficult to find material that would help with an assessment and provide confidence that the development was, and would continue to be, carried out in a sound manner. From a safety perspective and, for that matter, a security one too, there is no identification of the hazards that need to be mitigated through the development. This means that although subsequent claims for higher security might be made, there is a lack of specific justification for how this might have been achieved.

9.4.4 Development

It is difficult to identify a development process. There are no explicit software requirements, although the Doxygen generated descriptions of the APU functions could be considered to be the requirements. The expectation from contributors is set out in contributing.md and hence, informally, could be called a software life cycle. The software development environment is described in the user_guide.md. However, the issues identified in para 9.3.4 are the same or worse for SCP and are not repeated here. There does not appear to be a formal release process; user_guide.md assumes any user will just fetch the latest copy of the master branch. As such deployment is at the users discretion/risk and amended and verified as they need. Arm make no claim for 'correctness' of this code.

9.4.5 Design

The design step is not simple to discern because the documentation is auto-generated from the source code. While this makes traceability easy, the problem is the lack of apparent intent. The link between requirements, what is intended, to how to build the code is missing and reverse justifying the design from the code does not mean that the design reflects what was intended. The problems with lack of explicit requirements, as noted earlier, combined with the reverse engineered from code design means that the value of any testing is questionable.

9.4.6 Independence

Independence is well known to bring a good level of assurance whether by another person or through the use of a tool. All submissions are reviewed by the maintainers, though the contributor could be a maintainer. The enforcement of independence would be of benefit.

9.4.7 Verification

The only test procedures in the git repository relate to the framework code which can be found in the git repository under framework\test. These tests are well-written and informally appear do a good job of normal range and robustness testing of the framework code. These test procedures run on the host and verify the test framework code. The framework tests are laid out logically and are easy to understand. However, there are no guidelines on how to write tests, the rationale for how the test cases were selected is not explained, the test procedures do not contain any comments and the test procedures do not trace to any requirements. The structural coverage achieved by the framework tests has not been measured. It is not known what percentage of the framework code has been exercised by the framework tests.

There are no test procedures in the git repository that test the SCP firmware itself, either on the host or on the target. According to David A. Wheeler's SLOCCount, SCP Firmware contains a total of 381,512 Source Lines of Code (SLOC), but the test directory contains only 5,291 SLOC.

The absence of clearly defined requirements and traceability presents a problem in the measurement of coverage of requirements by test cases that also lack trace information. The structural coverage achieved by the framework tests has not been measured which means there is no detection of issues related to dead code, exercise of all the code and the thoroughness of the test suite. There are many open source and commercial tools available that measure structural code coverage. It is recommended that SCP Firmware adopts one of these tools so that, when they have developed tests for the SCP Firmware (not just the framework code), they can determine what percentage of the SCP Firmware code is exercised by the SCP Firmware tests and to ensure there is no unreachable code. See Appendix B Example Vulnerability Detectable by Coverage Analysis for an example of why this approach would be useful and enhance confidence in security.

9.4.8 Software Quality Assurance

According to DO-178C, Software Quality Assurance (SQA) provides confidence that the software life cycle processes produce software that conforms to its requirements by assuring that these processes are performed in compliance with the approved software plans and standards. DO-178C requires SQA to be applied with sufficient independence and authority to ensure corrective action. There is no SQA involvement in the SCP Firmware project. The maintainers are responsible ensuring the integrity of SCP Firmware. It would add confidence if the SCP Firmware project were subject to SQA oversight to provide an independent assessment of the software life cycle processes and their outputs

9.4.9 Tool Qualification

DO-178C requires a software tool to be qualified if DO-178C processes are eliminated, reduced, or automated by the use of the tool without its output being verified. The tool qualification is carried out in accordance with DO-330 (RTCA, Inc., 2011), which is also published as ED-215 (EUROCAE, 2012). Tool qualification does not have to be onerous. DO-330 defines five Tool Qualification Levels (TQL-1 to TQL-5). Most software verification tools are TQL-5, the least rigorous level. Since SCP Firmware makes no claim for compliance with DO-178C, it is not surprising that no tools have been qualified.

9.4.10 International Standards

Although SCP Firmware is intended to be a software reference implementation for the System Control Processor (SCP) and Manageability Control Processor (MCP) components found in several Arm Compute Sub-Systems, it does not claim compliance with any cybersecurity standard. It would be useful if the SCP Firmware documentation were to identify relevant cybersecurity standards and discuss how SCP Firmware could be extended to comply with those standards.

9.4.11 Conclusion to SCP Firmware Assessment

It is understood by the audit team that SCP Firmware is only provided as a reference and that it is normally expected that each licensee will develop their own SCP Firmware to provide their own performance differentiation. It is therefore to an extent understandable that effort could well be wasted in development of such a reference set of code. The team that have developed the SCP Firmware appear to have followed typical industry practice where independent scrutiny is not required. The focus is very much on code rather than test or documentation. However, there are a number of practices that could be improved including the development of requirements in order to be able to link to the code and to tests that are claimed to verify the code. This will enable future users within Arm, as well as those that are using SCP Firmware unmodified, to have greater confidence in the code from both a performance as well as a security perspective.

10 The Assessment – DO-326A, et al

10.1 Compliance

To date there is no standard guide for compliance assessment for DO-326A and associated documents and, given the subject matter and the security developments in the sector, it is unlikely that one will be developed. The approach adopted has therefore been developed for this assessment based on how the documents can be applied and therefore the applicability to CHERI/Morello. As noted in para 8.2 the focus is on whole aircraft with supporting evidence from the relevant systems

development and assessment. Without such context, there is little point in going through the whole of the document suite looking for evidence as it will simply be absent. The relevant part of the whole document set to consider is what would CHERI bring to security that is currently of concern to developers and regulators. This then focusses on a few Objectives within the entire document suite.

10.1.1 Security Assurance

There are 2 aspects identified with regard to assurance within DO-326A: Security Development Assurance and Security Effectiveness Assurance:

- Security Development Assurance seeks to assure that security measure perform as intended.
- Security Effectiveness Assurance seeks to assure that security measures (and architecture) is free of known and unacceptable exploitable vulnerabilities.

From an assurance perspective, both are supported by CHERI. Firstly, the intent of CHERI is to remove exploitation options from the deployed hardware, so supports Security Effectiveness Assurance. The vast majority of effort in academic papers, etc. will provide sufficient evidence that the measures perform as intended. However, there is more to this as CHERI does not defend against every vulnerability, so it is for the developer of the software to provide assurance that the code deployed in the application is also well found and itself is free from exploitation opportunities. This is an important point. The CHERI documentation makes bold claims about strong vulnerability mitigation. Nevertheless, we have identified a real-world example (the Apple TLS bug) that would not have been mitigated by the CHERI architecture. Furthermore, both TF-A and SCP Firmware seem to be particularly susceptible to this kind of defect due to the lack of requirements traceability and structural code coverage. Fortunately, in aerospace, it is typically the case that for applications requirements, design and especially code standards are chosen to avoid unforeseen issues in execution. However, the main concern is that the Executable Object Code, i.e., the code that is installed and flown is free from vulnerability. This requires an understanding of how the code will execute and this means understanding any RTOS and firmware that is supplied to ensure that the behaviour of the code is well understood and can be shown to be free from issues that would cause the code to exhibit unforeseen characteristics. As can be seen from the DO-178C assessments in this report, there are some concerns over TF-A and especially SCP Firmware.

10.1.2 Refutation

Regardless of arguments of exploitation options being removed, there remains an objective O3.2 from DO-356A “Refutation tests are performed to evaluate the exposure of vulnerabilities in the security environment and to challenge the vulnerability evaluation.” to ensure that object code – both as an output from software and hardware development – has been tested to be secure. While CHERI provides some level of assurance that such tests may pass, there remains the need to carry them out because coders may still construct vulnerabilities within the code that CHERI cannot protect against.

10.2 Coding

The ultimate protection against vulnerabilities remains with the software developer and this starts with requirements. If it is not clearly defined what the code is intended to do and what threats it is expected to meet, then it is rather difficult to provide objective evidence that any system, CHERI enabled or not, is safe/secure to fly. The use of a good code subset with well-defined behaviour, a sound means for checking compliance to that subset and good programming practices cannot be substituted solely by a back stop provided by capabilities.

It is often claimed that modern software is reliable despite not having any software requirements and that software requirements are therefore unnecessary; Linux is often cited as an example. It is pointed out there is no ‘Software Requirements Specification’ for Linux, yet the Linux kernel has turned out to be extremely reliable. However, the Linux kernel was designed to implement the same API as the Unix kernel. The Unix kernel is extremely well-documented, especially in the `man` pages and the Unix Programmer’s Manual. There also exist several reference implementations, including the original Bell Labs Unix and the Berkeley Standard Distribution (BSD) Unix. It is therefore contended that while the Linux

kernel does not have a specific document titled 'Software Requirements Specification', it does nevertheless have a very clear specification.

Furthermore, the Linux development has well controlled practices with many independent views on the robustness of releases. This tends to strengthen the argument that Linux is robust and possibly secure, but unless other open-source projects mirror the good practices, some of which are highlighted in this report, then they cannot claim to be the 'same as Linux'. All of this requires a lot of effort by many people and therefore is not cheap. Furthermore, many may have noticed the stability and security of other non-open source operating systems, such as Windows. Over the past 2 decades, increasing adoption of good software engineering practices have been incorporated. Indeed, the likes of Google, Amazon, Facebook, etc have had recruitment campaigns for formal methods specialists which has enabled for more robust software to be developed for more quickly than perhaps would otherwise have been able.

Capabilities won't stop bad programming practices, at least in the short term. Capabilities will stop, quite possibly literally, bad programs from executing. While this might encourage better programming from those inclined to do so, the vast majority of the eco-system may not worry as it changes little from their perspective: they're only worried about making the program 'work' and are not generally worried about the 'never' behaviours. This is how security vulnerabilities are made and subsequently exploited. The lack of a real understanding of how to develop code that has demonstrated robustness is of concern and reflects our experience of many open-source projects.

11 Use of the Fixed Virtual Platform

The Fixed Virtual Platform (FVP) was accessed and downloaded with the intent to run some experiments using some D-RisQ applications. The idea was to run either a real time application (our assured decision-making software application for autonomous systems) or one of our tools called Kapture®. While the download of the FVP went well, there were a number of issues that meant we could not run the intended experiments. Firstly, the current status of the CHERI/Morello programme is only focussed on the Arm-A that has no real time support. While this was a fundamental shortfall in our ability to conduct any 'real-time' experiments, the second option had yet more issues. We aimed to use one of the 2 operating systems that are being worked on that take advantage of capabilities, the Android and CHERIBSD projects. The Android version is not complete enough for us to be able to use, so was ruled out. The CHERIBSD was downloaded and booted, but immediately crashed. After multiple attempts to identify the problem, which failed, this experiment was also abandoned. The obvious reason for this being that if we could not load an operating system then we would not be able to run Kapture. However, additionally, we also did not have the resource or funding to re-target Kapture to a non-Windows environment (or even a non-Linux one). While considerable time/effort was expended, we have no useful results. It can be said though, that any project that is not targeting a Windows environment will result in poor take up (let alone a Linux release) in the server/laptop market.

12 Conclusions

The aim of the CHERI project and Morello development is to attempt to reduce the opportunity for cyber-attack. As such, it should be expected that good software engineering practice would have been followed to develop the firmware. The assessment sought evidence of good software engineering practice by using an internationally recognised independent process used for the safety assessment of aerospace software; the FAA Job Aid. This looks for documented evidence to meet objectives as described in RTCA DO-178C, the de-facto software standard for aerospace. The 2 elements of firmware examined using the FAA Job Aid were Trusted Firmware-A (TF-A) and System Control Processor (SCP). A further assessment against DO-326A and associated standards for airworthiness and system security. Finally, the Fixed Virtual Platform was downloaded with the intention of running some experiments.

The desk review undertaken by the audit team was looking for a description of the required functionality ('requirements'), a software design, source code combined with tracing information that linked them together and to verification activities such

as documented reviews, tests or static analyses that provided assurance that the code does what is expected and, crucially, nothing else. Furthermore, the audit was looking for evidence of configuration management, Software Quality Assurance and the production and use of various internal standards for how to unambiguously describe required functionality, design and code.

There were marked differences in the outcomes of the DO-178C compliance audit between TF-A and SCP, but also a number of similarities. Both had a lack of clearly defined requirements and no definable design step. Trace information was lacking or absent and while there is a coding guideline, it was not clear how compliance to the guidance was assured. Tools, where mentioned were not qualified, which means there is no documented evidence that their use is understood and can be relied upon. The major differences were based on the intended use of the code. TF-A is intended to be used largely unaltered by licensees of the Arm product, whereas SCP is an example of how to develop firmware by hardware developers. TF-A did have descriptions of some of the intended functionality and to an extent it was possible to deduce the link between this and code and tests. Accordingly, SCP has little documentation and consequently there is much lower confidence in the code. It is understood that there are clients that are using SCP firmware unaltered.

From a security and airworthiness perspective, the internationally accepted approach is documented in DO-326A and associated publications. There is no standard guide for its use and assessment of products developed. There was little to be gained by a full assessment against the standard so the focus was on the 3 objectives that were the most relevant. These focus on the security assurance gained from the development and the effectiveness of the techniques deployed as well as a set of refutation tests which challenge the vulnerability evaluation and seek to ensure that the code is secure. When combined with the outcome of the safety assessment using DO-178C, there are some deficiencies. While there are coding guidelines based upon MISRA-C, there appears to be no technique for assuring compliance. Furthermore there is no analysis of coverage of either the requirements, which are weak, nor the code structure. Code structure coverage analysis can be used to reveal code that is not required and through a short review examples of this were found in the audit. Code that has no function may lead to exploitation.

CHERI gives a substantial improvement in security compared with a conventional instruction set architecture (ISA). However, while the CHERI architecture is effective at preventing some classes of vulnerability, such as the buffer overflow that caused the Heartbleed bug [11], it cannot mitigate other vulnerabilities, such as the dead code that caused the Apple Secure Transport vulnerability[9]. The two components of the Morello software stack that we sampled are vulnerable to defects like the one reported in Apple's Secure Transport platform.

Should application developers wish to adopt CHERI, there will need to be some changes to software development practices in order to take best advantage of this functionality. However, for those who do not wish to change coding practices in particular, the only incentive will be frustration whenever capabilities intervene by, for example, stopping programmes. This is likely to be found when code is already deployed as typically it is only though diligent robustness testing that such issues are found, something that is not likely if developers do not wish to change their ways. It is therefore suggested that firmware developed for CHERI should also adopt a more rigorous approach.

The full assessments are too large to be included in this summary report. These can both be made available from D-RisQ on request. The references are: "An Assessment of Trusted Firmware-A v1.0 – CHERI STONE D2" and "An Assessment of System Control Processor Firmware v1.0 – CHERI STONE D3".

13 Acknowledgments

The project benefitted from funding from the Innovate UK Digital Security by Design programme for which we are grateful. The staff contacted at Arm were very helpful and open and we thank them for their assistance.

14 Disclaimer

The views expressed in this report are those of the authors based upon the open information available and an incomplete assessment in isolation of an actual use case. No proprietary information was sought or used as part of the production of this report. The views do not necessarily reflect those of any other body in industry or government and cannot be used as evidence of compliance or otherwise of the software to aerospace or any other standards.

Appendix A SOI Audit Results

This Appendix summarises the audit results for both of the firmware assessed using the FAA Job Aid. There are 4 Stages of Involvement (SOI) to a typical audit: SOI#1 – Planning, SOI#2 – Development, SOI#3 – Verification and SOI#4 Final. It should be noted that the Job Aid is a guide to auditors and covers many types of software which is why there are a number of aspects that 'Not Applicable'. Those that remain 'Not Evaluated' is because there was insufficient information to make a determination. The detailed results have been made available to the developers and can be obtained from the author on application. There was no assessment against SOI#4 because as there was no project to archive, finalise test results, etc, it was not practicable to undertake this exercise.

TF-A Audit Result Summary

This is a summary of the results of the assessment of Trusted Firmware-A against DO-178C using the FAA Job Aid. The full report is at [14].

SOI#1 – Planning

Not Compliant	48	22%
Compliant	27	13%
Not Applicable	54	25%
Partial	63	29%
Not Evaluated	0	0%
Blanks	23	11%
Total	**215**	100%

SOI#2 – Development

Not Compliant	36	11%
Compliant	46	14%
Not Applicable	100	30%
Partial	120	36%
Not Evaluated	8	2%
Blanks	26	8%
Total	336	100%

SOI#3 – Verification

Not Compliant	41	24%
Compliant	15	9%
Not Applicable	34	20%
Partial	36	21%
Not Evaluated	25	14%
Blanks	23	13%
Total	174	100%

SCP Audit Result Summary

This is a summary of the results of the assessment of Trusted Firmware-A against DO-178C using the FAA Job Aid. The full report is at [15].

SOI#1 – Planning

Not Compliant	50	23%
Compliant	14	7%
Not Applicable	65	30%
Partial	62	29%
Not Evaluated	1	0%
Blanks	23	11%
Total	**215**	100%

SOI#2 – Development

Not Compliant	33	10%
Compliant	19	6%
Not Applicable	150	45%
Partial	103	31%
Not Evaluated	5	1%
Blanks	26	8%
Total	336	100%

SOI#3 – Verification

Not Compliant	59	34%
Compliant	25	14%
Not Applicable	40	23%
Partial	27	16%
Not Evaluated	0	0%
Blanks	23	13%
Total	174	100%

Appendix B Example Vulnerability Detectable by Coverage Analysis

The following is an example security vulnerability that would not have been trapped by the approach adopted by the development of the firmware in Morello. It requires well written requirements as well as trace information through design into code and appropriate developed and traceable test cases. Structural coverage analysis is useful because:

1. It ensures that all the code has been exercised.
2. It establishes the thoroughness of the test suite.
3. It can help to find dead code and unintended code.

A serious bug was reported in Apple's TLS implementation in 2014 [9][10]. This created a vulnerability in all devices using iOS and MacOS.

The code read as follows:

```
1              if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
2                      goto fail;
3              if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
4                      goto fail;
5                      goto fail;
6              if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
7                      goto fail;
8              ...
9  fail:
10             SSLFreeBuffer(&signedHashes);
11             SSLFreeBuffer(&hashCtx);
12             return err;
```

Note the two `goto fail` lines in a row at lines 4–5. The indentation is misleading — the code will always jump to the end from that second goto at line 5. In that case. `err` will contain zero because the SHA1 update operation was successful and so the function will return success even if the signature was invalid. This was a subtle but serious vulnerability that allowed a man-in-the-middle attack to spoof SSL servers.

This defect could have been detected had there been detailed software requirements to define the intended behaviour, which is that the function should return success if the site has a valid security certificate, an appropriate error code if there is no security certificate or if it is invalid, and test cases traced to those requirements. Detailed software requirements such as these are called low-level requirements in DO-178C.

Measuring statement coverage would also have detected this particular defect because it would have shown that the line of code `if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)` cannot be reached (due to the erroneous `goto fail` on the previous line). However, structural coverage cannot always find the absence of intended function.

There are many open source and commercial tools available that measure structural code coverage. It is recommended that SCP Firmware adopts one of these tools so that, when they have developed tests for the SCP Firmware (not just the framework code), they can determine what percentage of the SCP Firmware code is exercised by the SCP Firmware tests and to ensure there is no unreachable code.