D33.1 Addendum to CEM

Project number:	318353
Project acronym:	EURO-MILS
Project title:	EURO-MILS: Secure European Virtualisation for Trustworthy Applications in Critical Domains
Start date of the project:	1st October, 2012
Duration:	36 months
Programme:	FP7/2007-2013

Deliverable type:	Report
Deliverable reference number:	ICT-318353 /D33.1
Activity and Work package contributing to the deliverable:	Activity 3 / WP 33
Due date:	M36
Actual submission date:	5th October, 2015

Responsible organisation:	TCS
Editor:	TCS (Romain Bergé) TSYS (Igor Furgel)
Dissemination level:	PU
Revision:	1.0

Abstract:	Refinement of the CEM for high assurance. Software composition.
Keywords:	Refinement, CEM, application of attack potential, attack method, work units.

Editor

Romain BERGE, TCS

Igor FURGEL, TSYS

Contributors

Thomas BEN, Cyril PROCH (TCS)

Viola SAFTIG, Tobias WAGNER (TSYS)

Disclaimer

The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 318353.

This document has gone through the consortium's internal review process and is still subject to the review of the European Commission. Updates to the content may be made at a later stage.

Executive Summary

This document is an Addendum to the current version of the [CEM] document.

The [CEM] document is used by any ITSEF in order to perform evaluation of TOE submitted.

However, the [CEM] is a generic document that is technologic agnostic.

Hence, the aim to this document is to provide additional information to the [CEM] as a form of suggestion.

This information could be used to perform the evaluation of a MILS system or MILS component.

First, this document studies the current status of various technological domains with regards to the Common Criteria standard.

In the second part, this document develops, like for many other technological domains, the “Attack Potential”. It is a refinement for the MILS technology of the five essential criterion of an attack: elapsed time, expertise of the attacker needed, knowledge of the TOE needed, windows of opportunity needed and equipment required for the attack. Like for the Smartcard domain or the the Point of Interaction domain, this document suggest values and definition to be used for an evaluation of a MILS system. This section is written in a form that is compatible to the JIL consortium.

Later, this document suggests an “Attack method” for the MILS domain. The aim is to suggest attack path that could be used to perform the evaluation of a MILS system. In addition, this section suggests the JIL quotation for each attack. The JIL quotation is a standardized way of evaluating the resources needed and the complexity of an attack.

In the fourth section, this document suggests a refinement of the CEM document in the context of MILS system. Indeed, CEM is not complete and some work units to be performed are not defined. This section suggests an interpretation of the missing work units of the CEM in the context of MILS domain.

Finally, this document deals with the concept of Composition. This concept is well defined in the case of Hardware based technology. However, this concept has never been pushed to the Software layer. This section suggests a methodology that allows performing an evaluation of a software platform (like an Operating System) and then taking benefit of this certification to compose with applications.

This document is compatible with CC v3.

Contents

Figures:

Diagrams:

Tables:

Chapter 1 Addendum to CEM for a Separating Kernel

The chapter recalls the current available technical domains recognized by international organization such as JIL.

1.1 Supporting documents for technical domains

1.1.1 Introduction

This document contains the collective knowledge of EURO-MILS members about evaluation of Separation Kernel-based MILS systems, including the attacks and their rating in terms of resistance against a certain attack potential. Therefore, it must not be disclosed to the public or potential attackers. This document is only to be released to partners of EURO-MILS project.

The document must be protected during transmission and storage.

The Common Criteria Recognition Agreement (CCRA) provides for international recognition (by CCRA members) of CC certificates up to EAL2 (+ ALC_FLR.1) assurance level.

For Higher assurance level (EAL4 + ALC_FLR.1) the CCRA asks for the establishment of Common Criteria (CC) communities for each type of Target Of Evaluation (TOE) to develop Collaborative Protection Profiles (cPP). Those cPP will be composed of PPs (as defined today) but also including a set of supporting documents to refine the CEM for this particular type of product. The set of documents under the cPP will be developed by Technical Communities composed of Certification Bodies (CB), developers and laboratories.

SOG-IS agreement allows a recognition of high EAL certificates (up to EAL7) by restricted SOG-IS (European) members for well-defined and mutually agreed SOG-IS Technical Domains. But the CEM does not propose any guidance for the interpretation of evaluation methodology, evaluation efforts, and type of attacks on AVA_VAN.5.

How under these conditions a CC evaluation on Separating Kernel with a high assurance level (greater than EAL4) can be performed and recognized internationally? The SOG-IS approach is to create supporting documents specifically to cover the SOG-IS IT-Technical domains high EAL specificities.

The Euro-MILS initiative could address the two constraints described above by setting-up a MILS community and providing a set of documents supporting the high level EAL7 evaluation of Separating Kernel MILS (Protection Profile ([PP]) products.

This document provides guidance on the evaluation methodology for higher assurance components not covered by [CEM] and on attack methods that have to be considered in a Separating Kernel evaluation. This document describes what can be expected from the high level AVA.VAN5 in terms of robustness inspired from current international recognition agreements documentations on attack methods.

The document also helps the standardization of the security rating of MILS products.

To this end, the evaluation methodology provides guidelines for higher level assurance components as well as examples for the attack rating.

It should be noted that for attack rating consistency the concept of “Exploitation” and “Identification” is maintained.

1.1.2 Scope of the document

This document completes and interprets (annex B.4 of [CEM]) the current version of Common Methodology for Information Technology Security Evaluation [CEM].

Specifically, this document:

- Describes technical guidance on application of the attack potential to the evaluation of Separating Kernel products with typical separation kernel protection measures as defined in [PP]

- Provides evaluation methodology applied to Separating Kernel products for higher level assurance components that are not currently covered by [CEM].

The aim of this document is not to provide a detailed guide on concrete attack methods, but a general compilation of applicable attacks in this technological field in order to identify and describe a quantification method to rate the attack potential. This method is to be used by ITSEFs performing vulnerability analysis or by ITSEFs getting technical competence in this area.

1.1.3 Existing IT technical domains in SOG-IS context

1.1.3.1 SmartCards and similar devices (SOG-IS Technical Domain)

The following supporting document has been defined:

- Application of Attack Potential to SmartCards, v2.9, Mandatory

1.1.3.2 Hardware devices with security boxes (SOG-IS Technical Domain)

The following supporting document has been defined:

- Application of Attack Potential to Hardware Devices with Security Boxes, v1.0, Trial use

The domain “Hardware devices with security boxes” contains the following subdomains:

1.1.3.2.1 Point Of Interaction (POI)

The following supporting documents have been defined:

- Application of Attack Potential to POIs, v1.0, For trial use
- CEM Refinements for POI Evaluation, v1.0v, For trial use

1.1.3.2.2 Digital Tachygraph

The following supporting document has been defined:

- Security Evaluation and Certification of Digital Tachograph, v1.12, Mandatory

1.1.3.3 Notes regarding “Hardware Devices with Security Boxes” domain

This IT-Technical Domain is related to products produced from a series of discrete parts on one or more printed circuit boards whereby significant proportions of the required security functionality depend upon a hardware physical envelope with counter-measures (a so-called “Security Box”) against direct physical attacks (for example of payment terminals, tachograph vehicle units, smart meters, taxi meters, access control terminals, Hardware Security Modules, etc.).

More precisely, this domain covers products such as payment terminals or Hardware Security Modules (HSM) on which part of the security relies on a “secure” package that protects against attackers internally. Those secure envelops detect physical attacks and trigger security action (generally secret erasure) on detection.

In the technologies covered by the above scope, an attacker will often be able to obtain physical access to the device (or a set of devices). The device may contain critical information such as security credentials/keys, or could be used for secure entry of credentials/keys.

A significant part of the security functionality required from the device will relate to self-protection against physical attacks. These self-protection counter-measures or the “security box” of such devices is composed of physical protection counter-measures based on hardware and software mechanisms. These mechanisms involve active and passive parts of protection.

The evaluation approach needs to consider all software, firmware and hardware specific aspects of vulnerability analysis including those that may require significant additional equipment and resources. Such devices are also frequently composed from discrete parts produced by different developers. These factors must also be consistently taken into account during evaluation and certification.

Separating Kernel and MILS can be seen as a sub-system of “Hardware devices with security boxes” (a recognized JIL technical domain), were the software brought by the Separating Kernel and partitions is integrated with a hardware product.

Chapter 2 Application of Attack Potential for MILS systems

2.1 Introduction

This paragraph introduces the notion of an attack quotation. We should explain what factors are taken into account (and how) for quotation of an attack on MILS up to the high attack potential.

Note: When cryptography is involved, the responsible Certification Body (CB) should be consulted.

2.2 How to compute the potential of an attack

The risk management performed by the user of CC certificates requires clearly having a distinction between the cost of “identification” (definition of an attack) and the cost of “exploitation”, i.e. practical performing the attack (e.g. once a script is published on the World Wide Web). Although the distinction between identification and exploitation is essential for the MILS evaluation to understand and to document the attack path, the final sum of attack potential is calculated by adding the points of the two phases, as both phases build the complete attack.

Attack path identification and exploitation analysis and tests are mapped to relevant factors:

- Elapsed time
- Expertise (of the attacker)
- Knowledge of the product (TOE)
- Window of Opportunity (Access to the TOE)
- Equipment required for the attack

Even if the attack consists of several steps, identification and exploitation need only be computed for the entire attack path.

NB: In the CEM, there is no difference between Identification phase and Exploitation phase. This difference however is meaningful and is used on other technologies.

2.2.1 Identification

The identification part of an attack corresponds to the effort required to create the attack, and to demonstrate that it can be successfully applied to the TOE (including setting up or building any necessary test equipment). The demonstration that the attack can be successfully applied needs to consider any difficulties in expanding a result shown in the laboratory to create a useful attack.

It may not be necessary to carry out all of the experiments to identify the full attack. In that case the attack must actually prove that a security property of an asset having to be maintained by the TOE as defined in the Security Target is indeed not maintained (thus undermining the security policy defined in the Security Target), and that the complete attack could realistically and successfully be carried out.

One of the outputs from Identification is assumed to be a script that provides step-by-step descriptions on how to carry out the attack – this script is then to be used in the exploitation phase of the attack.

Sometimes the identification phase will involve the development of a new type of attack (possibly involving the creation of new equipment) which can subsequently be applied to other TOEs. In such a case the question arises on how to treat the elapsed time and other parameters when the attack is reapplied to other TOEs.

Note: The interpretation taken in this document is that the development time (and, if relevant, expertise) for identification includes the development time for the initial creation of the attack, until the relevant CB considers that the attack is now common. Once a CB has determined this, no points for the development of the attack (in terms of time or expertise) will be used in the future attack potential calculation. Typical example could be the case of new cryptographic attack that requires a lot of time and expertise for HW / SW development. This development effort is taken into account (through the Identification part of the quotation) during several successive evaluations until the CB considers that such HW / SW tool is common and part of the standard tools in the labs.

2.2.2 Exploitation

The exploitation phase of an attack corresponds to achieving the attack on another instance of the TOE using the analysis and techniques defined in the identification phase of an attack. It is assumed that a different attacker carries out the exploitation, but that the technique (and relevant background information) is available for the exploitation in the form of a script or set of instructions defined during the identification of the attack.

The script is assumed to identify the necessary equipment and, for example, mathematical techniques used in the analysis. This means that the elapsed time, expertise and TOE knowledge ratings for exploitation will sometimes be lower for exploitation than for identification. For example, it is assumed that the script identifies such things as the timing required for a perturbation attack, and hence, in the exploitation phase the attacker does not have to spend significant time to find the correct time point in software execution at which to apply the perturbation. Furthermore, the same information may also reduce the exploitation requirement to one of time measurement, whereas the identification phase may have required reverse engineering of hardware or software information from power data – hence the expertise requirement may be reduced.

Similarly, knowledge about the application that was used to achieve the timing of an attack may also be included either directly in the script or indirectly (through data on the timing required).

In many cases, the evaluators will estimate the parameters for the exploitation phase, rather than carry out the full exploitation. The estimates and their rationale shall be documented in the Evaluation Technical Report (ETR).

2.2.3 Final quotation

To complete an attack potential calculation the points for identification and exploitation have to be added as both phases build the complete attack. When presenting the attack potential calculation in the ETR, the evaluators will make an argument for the appropriateness of the parameter values used, and will therefore give the developer a chance to challenge the calculation before certification. The final attack potential result will therefore be based on discussions between the developer, the ITSEF and the CB, with the CB making the final decision if agreement cannot be reached.

It is an assumption that the CB will ensure that there is harmonization between national CC certification schemes. This is required, for example, where new types of attack are applied and a decision has to be taken as to when the attack is considered 'mature', at which point it will no longer gain points for the time or expertise to develop the attack (as discussed above).

2.3 Elapsed Time

The Elapsed Time indicates how long an attacker requires to identify or exploit an attack.

NB: even if the name of this section is "elapsed time", "workload" is a better translation of what we really have in mind.

Time is divided into the following intervals:

Elapsed Time (t)	Identification	Exploitation
t < one hour	0	0
One hour <t ≤ one day	1	2
One day <t ≤ one week	2	3
One week < t ≤ one month	3	4
t > one month	5	7

Table 1: Rating for Elapsed Time

For purposes of calculating time, a day = 8 hours, a week = 40 hours; and a month = 180 hours.

If the attack consists of several steps, the Elapsed Time can be determined and added to achieve a total Elapsed Time for each of these steps. Actual labour time has to be used instead of time expired as long as there is not a minimum Elapsed Time enforced by the attack method applied.

In those cases where attendance is not required during part of the Elapsed Time, the Elapsed Time is to be taken as expired time divided by 3.

The rating in Table 1 is commensurate with the related rating in [AAPHDSB].

2.4 Expertise

Expertise refers to the level of knowledge of the application area or product type (e.g., Unix operation systems, Internet protocols).

The expertise depends on knowledge of

- Common Product information
- Common knowledge on the product type (here: operating system, virtualiser) concerning its architecture (HW & SW) and widely used algorithms and protocols.
- Dedicated attacks requiring special skills according to the types of SFRs defined in the ST.
- Principles and concepts of security
- Developers knowledge on the product or on the product type
- Interrelationship between principle and concepts

For the purpose of MILS three types of experts are defined:

- **Laymen** are unknowledgeable compared to experts or proficient persons, with no particular expertise.
- **Proficient** persons are knowledgeable in that they are familiar with the security behaviour of the product.
- **Experts** are familiar with the underlying algorithms, protocols, hardware, virtualization, etc. implemented in Separation Kernels.
- The level **"Multiple Expert"** allows for situation where different fields of expertise are required at an Expert level for distinct steps of an attack.

It may occur that several types of expertise are required. By default, the higher of the different expertise factors is chosen. In very specific cases, the "Multiple Expert" level could be used when several non-alternative types of expertise are required but it should be noted that the expertise must concern fields that are strictly different like for example SW reverse engineering and cryptography.

	Definition according to the CEM	Detailed definition to be used in Separation Kernel evaluation
Experts	Familiar with underlying: • Algorithms • Protocols • Hardware • Structures • Security Behaviour • Principles and concepts of security employed • Techniques and tools for the definition of new attacks • Cryptography • Classical attacks for the product type • Attack methods • Etc…	Professional experience with • Specific algorithms • Crypto-analysis • Domain specific protocols (automotive, avionic, space…) • Other SFR-type-specific behaviour • Specific Hardware interfaces and the capability to use hardware to cause a non-ST conformant behaviour of MILS virtualiser / operating system • Deep knowledge of specific hardware platform and assembler language • Knowledge of confidential security flows in hardware platform, software IPs, development tools (development of exploit for 0days) • Interaction of security concepts • Deep knowledge of MILS architecture, virtualisation, separation kernels, operating systems
Proficient	Familiar with: • Security behaviour, • classical attacks	Familiar with: • Cryptography concepts • Standard protocols • Standard Hardware interfaces • Capability to interact with hardware • Standard hardware platform and assembler language • Knowledge of classical attacks and published security flaws on hardware platform, software IPs, development tools • Security behaviour of MILS • Operating systems architecture • Development tools
Laymen	No particular expertise	No particular expertise but as a user of IT products, general knowledge on • Cryptography means • Standard hardware interfaces • Standard hardware platforms • Ready to use attack tools • Standard Operating Systems

Table 2: Definition of Expertise

Expertise	Identification	Exploitation
Layman	0	0
Proficient	1	1
Expert	2	3
Multiple Expert	5	6

Table 3: Rating for Expertise

The rating in Table **3** is commensurate with the related rating in [AAPHDSB].

2.5 Knowledge of the TOE

It shall be clearly understood that any information required for identification shall not be considered as an additional factor for the exploitation. This means that when an attack path has been discovered during the identification phase with the help of design information, this information is considered as known to the attacker for the exploitation phase.

In the same approach, to require sensitive information for exploitation would be unusual because the attack paths are expected to be found during the exploitation phase. The only case where it could happen is for design details that would be specific to each product.

Since all sensitive and critical design information are presumed to be well controlled and also *confidentiality*-protected by the developer (this determination is the subject to ALC_DVS.x evaluation activities), it is obvious that lack of its knowledge cannot assists in determining a dedicated attack path. Therefore, it shall be clearly stated in the attack potential calculation rationale why the required critical information cannot be substituted by a related combination of time and expertise.

The following classification is to be used:

- **Public** information about the TOE (or no information): Information is considered public if it can be easily obtained by anyone (e.g., from the Internet) or if it is provided by the vendor to any customer.

- **Restricted** information concerning the TOE (e.g., as gained from vendor technical specifications): Information is considered restricted if it is distributed on request and the distribution is registered. Suitable example might be the functional specification (ADV_FSP).

- **Sensitive** information about the TOE (e.g., knowledge of internal design), which may have to be obtained by “social engineering” or exhaustive reverse engineering). Suitable example might be TOE Design Specification (ADV_TDS) information or the Source Code.

Note: Care should be taken here to distinguish between information required to identify the vulnerability and the information required to exploit it, especially in the area of sensitive information. Requiring sensitive information for exploitation would be unusual. It may occur that for sophisticated attacks, several non-alternative types of knowledge are required. In such cases, the higher of the different knowledge factors is chosen.

Knowledge	Identification	Exploitation
Public	0	0
Restricted	2	2
Sensitive	3	4

Table 4: Rating for Knowledge of TOE

The rating in Table 4 is commensurate with the related rating in [AAPHDSB].

Note: The level of general expertise and knowledge of the concrete TOE are concerned with the information required for persons to be able to attack a TOE. There is an implicit relationship between an attacker's expertise and the ability to effectively make use of equipment in an attack. The weaker the attacker's expertise, the lower the potential to effectively use equipment. Likewise, the greater the expertise, the greater the potential for equipment to be used in the attack. Although implicit, this relationship between expertise and the use of equipment does not always apply—for instance, when environmental measures prevent an expert attacker's use of equipment; or when, through the efforts of others, attack tools requiring little expertise for effective use are created and freely distributed (e.g., via the Internet).

2.6 Window of opportunity

Window of opportunity (Opportunity) is also an important consideration, and has a relationship to the Elapsed Time factor.

Identification or exploitation of vulnerability may require considerable amounts of access to a TOE that may increase the likelihood of detection.

Some attack methods may require considerable effort off-line and only brief access to the TOE to exploit. Access may also need to be continuous, or over a number of sessions.

Window of opportunity is mint to represent a variety of situation depending on the nature of the TOE.

The way an attacker can obtain the TOE may differ depending on the type of MILS.

A standard software distribution ready to be installed on standard hardware platform (like a PC) may be easy to obtain and may allow an attacker to freely experiment during the identification phase.

In the contrary, a specific MILS OS running on a dedicated system may be very difficult to obtain for free experimentation, and the attack may prefer experimenting on deployed equipment in use but taking the risk of being detected during its experimentation.

For some TOEs the Window of opportunity may represent the number of samples of the TOE that the attacker needs. This is particularly relevant where attempts to penetrate the TOE and undermine the SFRs may result in the destruction of the TOE preventing use of that TOE sample for further testing, e.g. hardware devices. Often in these cases distribution of the TOE is controlled and so the attacker must apply effort to obtain further samples of the TOE.

For some TOEs the Window of opportunity may represent how difficult it is for an attacker to obtain working samples in order to freely experiment attacks.

For some TOEs the Window of opportunity may represent how difficult it is for an attacker to work directly on a working TOE in place with environmental protections. In that case, the risk of being detected depends on the level of environmental protection and the amount of time access to the TOE is needed.

Considering this, the CEM approach based on "Windows of opportunity" is maintained allowing more flexibility for the quotation. The CEM definitions are extended.

- **Unnecessary/unlimited access** means that the attack doesn't need any kind of opportunity to be realised because there is no risk of being detected during access to the TOE. It is no problem to access the number of TOE samples needed for the attack. It is no problem for obtaining a working TOE sample for experimentation or for exploit. It is no problem for accessing a TOE in place for experimentation or exploitation. This case could be representative of a software distribution for standard hardware platform, of large diffusion public equipment for which it shall be easy to obtain a sample for tests, or of an in place working equipment with easy access and no particular environmental protection.

- **Easy** means that the number of TOE samples required to perform the attack is less than ten or, obtaining samples for free experimentation may require some organisational/financial efforts or, access to a working TOE in place is required for less than a day with low environmental protection. This case could be representative of large diffusion public equipment for which it is necessary to have several (<10) samples, or of public equipment but with controlled diffusion, or of working equipment in place within public area.

- **Moderate** means that the number of TOE samples required to perform the attack is less than one hundred or, obtaining samples for free experimentation may require high organisational/financial effort or, access to a working TOE in place is required for less than a month with low environmental protections (one week with medium environmental protections). This case could be representative of large diffusion public equipment for which it is necessary to have a high number (<100) of samples, or of non-public equipment with controlled diffusion, or of working equipment in place with restricted access (area with standard access control and low survey).

- **Difficult** means that the number of TOE samples required to perform the attack is at least one hundred or, obtaining samples for free experimentation may require very high organisational/financial effort including complicity in the manufacturing/maintenance/storage area or, access to a working TOE in place is required for at least one month with low environmental protections (between one week and one month with medium environmental protections, less than one week with high environmental protection). This case could be representative of large diffusion public equipment for which it is necessary to have a very high number (>100) of samples, or of non-public equipment with highly controlled diffusion , or of working equipment in place with controlled access (area with biometric/electronic access control and high survey).

- **None** means that the opportunity window is not sufficient to perform the attack (the length for which the asset to be exploited is available or is sensitive is less than the opportunity length needed to perform the attack - for example, if the asset key is changed each week and the attack needs two weeks); another case is, that a sufficient number of TOE samples needed to perform the attack is not accessible to the attacker - for example if the TOE is a hardware and the probability to destroy the TOE during the attack instead of being successful is very high and the attacker has only access to one sample of the TOE.

Window of Opportunity	Identification	Exploitation
Unlimited	0	0
Easy	2	1
Moderate	4	3
Difficult	6	5
None	NA	NA

Table 5: Rating for Window of Opportunity

2.7 Equipment required for the attack

Equipment refers to the IT hardware/software tools that are required to identify or exploit vulnerability.

In order to clarify equipment category, price and availability has to be taken into account. The following categories shall be used.

- **Standard equipment** is equipment that is readily available to the attacker, either for the identification of vulnerability or for an attack. This equipment may be part of the TOE itself (e.g. a debugger or a compiler/disassembler in an operating system) or can be readily obtained without additional development effort from the attacker (e.g., at a nearby store or downloaded from the Internet). The equipment might consist of simple attack scripts, software debugger/compiler/disassembler, publically available SDK, network probe or protocol manipulation tool, personal computers, pattern generators, power supplies, or simple mechanical tools like standard drills, common use mechanical tools, soldering irons, etc.

- **Specialized equipment** isn't readily available to the attacker, but could be acquired without undue effort. This could include purchase of moderate amounts of equipment or readily obtained equipment completed by in-house developed extensions or dedicated customization (e.g., software debugger/compiler/disassembler with in-house developed extension modules, dedicated SDK which distribution is controlled, network manipulation tools with in-house developed extension modules, power analysis tool, dedicated electronic cards, Laser bench, Electromagnetic bench, other specialized test bench, protocol analysers, high end digital oscilloscopes, microprobe workstation, chemical workbench, precise milling machines, etc.) or development of more extensive attack scripts or programs (in-house developed extension for public attack tools).

- **Bespoke equipment** is not readily available to the public as:
  - it might need to be specially produced (e.g., very sophisticated or dedicated software/hardware such as in-house developed debugger/compiler/disassembler, in-house developed network generator/fuzzer, in-house developed dedicated software attack tools based on confidential vulnerability),
  - or because the equipment is so specialized that its distribution is controlled, possibly even restricted.
  - Alternatively, the equipment may be very expensive. (Focused Ion Beam, Scanning Electron Microscope, and Abrasive Laser Equipment).

Bespoke equipment, which can be rented, might have to be treated as specialized equipment. Very sophisticated software that has been developed during the identification phase is considered as bespoke equipment; it must not additionally be considered for the exploitation phase.

In an ideal world definitions need to be given in order to know what are the rules and characteristics for attributing a category to equipment or a set of equipment. In particular, the price, the age of the

equipment, the availability (publicly available, sales controlled by manufacturer with potentially several levels of control, may be hired) shall be taken into account.

The range of equipment at the disposal of a potential attacker is constantly improving, typically:

- Computation power increases
- Cost of tools decreases
- Availability of tools can increase
- New tools can appear, due to new technology or to new forms of attacks

It may occur that for sophisticated attacks, several non-alternative types of equipment are required. In such cases by default the higher of the different equipment factors is chosen.

The border between standard, specialized and bespoke levels cannot be clearly defined.

The rating of the tools is just a typical example. It is a case by case decision depending on state of the art and costs involved.

The following tables are just a general guideline for tools categorization and rating.

Tool	Equipment
Voltage supply	Standard
Oscilloscope analogue	Standard
PC or work station	Standard
Signal analysis software	Standard
Signal generation software	Standard
Network probe for standard protocols	Standard
Protocol analyser for standard protocols	Standard
Simple mechanical tools	Standard
Standard script engine (Python)	Standard
COTS Disassembler	Standard
COTS debugger	Standard
COTS compiler	Standard
COTS SDK	Standard
COTS fuzzing tools for standard protocols (IP, TCP, HTTP,…)	Standard
High Voltage EMR pulse generator	Specialized
Electromagnetic measurement/injection bench	Specialized
Micro-probe Workstation	Specialized
Laser equipment (fault injection)	Specialized
Signal and function processor	Specialized
High-end digital Oscilloscope	Specialized
Signal analyser	Specialized
Tools for chemical etching (wet)	Specialized
Tools for chemical etching (plasma)	Specialized
Tools for grinding	Specialized

Tool	Equipment
PC with multi GPU for fast computation (and signal processing)	Specialized
Domain specific network probe (AFDX,…)	Specialized
Domain specific protocol analyser (CAN, ARINC…)	Specialized
Fuzzing tools for domain specific protocol	Specialized
COTS attack/exploit tools (e.g; Metasploit…)	Specialized
COTS disassembler with in-house developed extension	Specialized
COTS debugger with in-house developed extension	Specialized
COTS compiler with in-house developed extension	Specialized
Dedicated SDK (controlled distribution)	Specialized
Dedicated/Project specific protocol analyser	Bespoke
Dedicated/Project specific fuzzing tools	Bespoke
In-house developed disassembler (Byte code,…)	Bespoke
In-house developed debugger	Bespoke
In-house developed compiler	Bespoke
In-house developed software tools implementing confidential exploits or taking advantage of non-disclosed vulnerability	Bespoke
In-house dedicated hardware development (typically multi-FPGA board)	Bespoke

Table 6 : Tools categorization

The following table states the number or points earned in the JIL quotation based on the kind of tool used, and the phase of the attack:

Equipment	Identification	Exploitation
None	0	0
Standard	1	1
Specialized	3	3
Bespoke	5	6

Table 7: Rating for Equipment

2.8 Final Table

The table below sum up the various tables explained above for each criterion of the computation of the attack: time elapsed, expertise of the attacker, knowledge of the TOE needed to carry out the attack successfully, window of opportunity and the equipment required.

Factors	Identification	Exploitation
Time elapsed		
< one hour	0	0
< one day	1	2
< one week	2	3
< one month	3	4
> one month	5	7
Expertise		
Layman	0	0
Proficient	1	1
Expert	2	3
Multiple Expert	5	6
Knowledge of the TOE		
Public	0	0
Restricted	2	2
Sensitive	3	4
Window of Opportunity		
Unlimited	0	0
Easy	2	1
Moderate	4	3

Factors	Identification	Exploitation
Difficult	6	5
None	NA	NA
Equipment required		
None	0	0
Standard	1	2
Specialized	3	4
Bespoke	5	6

Table 8: Final table for the rating factors

2.9 Range for CC v3

The following table replaces table B.4 of CEM, para 1988 for MILS.

Range of Values (Final attack potential = identification + exploitation)	TOE resistant to attackers with attack potential of
0-11	No rating
12-17	Basic
18-24	Enhanced - Basic
25-30	Moderate
30 and above	High

Table 9: Rating of vulnerabilities for CC V3

The rating in Table 9 is commensurate with the related rating in [AAPHDSB].

2.10 Partial or complete attacks

An attack scenario comprises one to many attack steps.

Analysis and tests need to be carried out for each attack step on an attack scenario for a vulnerability to be realized.

Each attack step shall be identified as either “Partial”, meaning that other attack steps have to be completed in order to compromise MILS assets, or “Complete” in which case a successful attack step directly compromises one or more assets.

An example of “Partial” attack would be to attack the separation kernel of a MILS product by injecting malicious software on the system, but there may be other security barriers like security box which must be overcome before malicious software can be installed on the system. In that case, both

"Partial" attacks (malicious software installation/exploitation & security box bypassing) must be combined in order to reflect the level of the complete attack scenario.

In the case of a "Complete" attack the attack potential calculation gives the attack potential for the whole attack scenario.

There should be an approach defined for combining attack potentials, in cases where two or more "Partial" attack steps need to be combined in order to compromise an asset. A suggested method is identified below.

2.11 Combination of "Partial" Attack Potentials

Where multiple attack steps have to be combined in order to successfully accomplish an attack scenario which compromises an asset, the overall attack potential should be calculated as follows:

- Initially add the attack potential ratings of each "Partial" attack step required. Scores for Identification and Exploitation should be maintained separately as well as an overall total.
- Individual elements of each attack step should then be reviewed to identify whether there is overlap between the steps.
  For example, if the same "Expert" is involved in more than one step this score should only count once. If the "Expert" is different in each step then the "Expert" score should remain for each step.
  If the same "Expert" is required in multiple steps, then these cannot be paralleled, and the timescales should reflect consecutive working rather than parallel. If the "Expert" is different, then attack steps may be run in parallel and the overall timescale score for the complete attack should be reduced accordingly.
  Similarly, if specialised equipment is required, but the same specialised equipment is required in multiple steps, the score for this should only be counted once.

In this way an overall score for a complete attack can be constructed.

The method consists on evaluating both, first-order and second-order attacks, according to their respective attack potential computation rules and then combine both into a single table.

Note: if the MILS is protected by a security box, JIL document "Application of attack potential to Hardware Device with Security Boxes" already provides a way of combining attack quotation for the security box with attack quotation for other IT domain protected by that box.

Chapter 3 Attack method for MILS

First, this sub-section gives the template used in JIL document for attack methods and explains the expected content.

Then, practical examples of attack methods are given by EuroMILS partners. Some attack methods are directed to the Separation Kernel itself and are described in chapter 1.3.3. Some other attack methods are more directed to the hardware platform and are described in chapter 1.3.4. Finally some other attack methods are directed to the MILS system and are described in chapter 1.3.5.

NB: JIL Attack Methods documents are restricted documents. Therefore EuroMILS partners did not disclosed any extract of existing Attack Method documents to issue this one.

3.1 MILS System general description

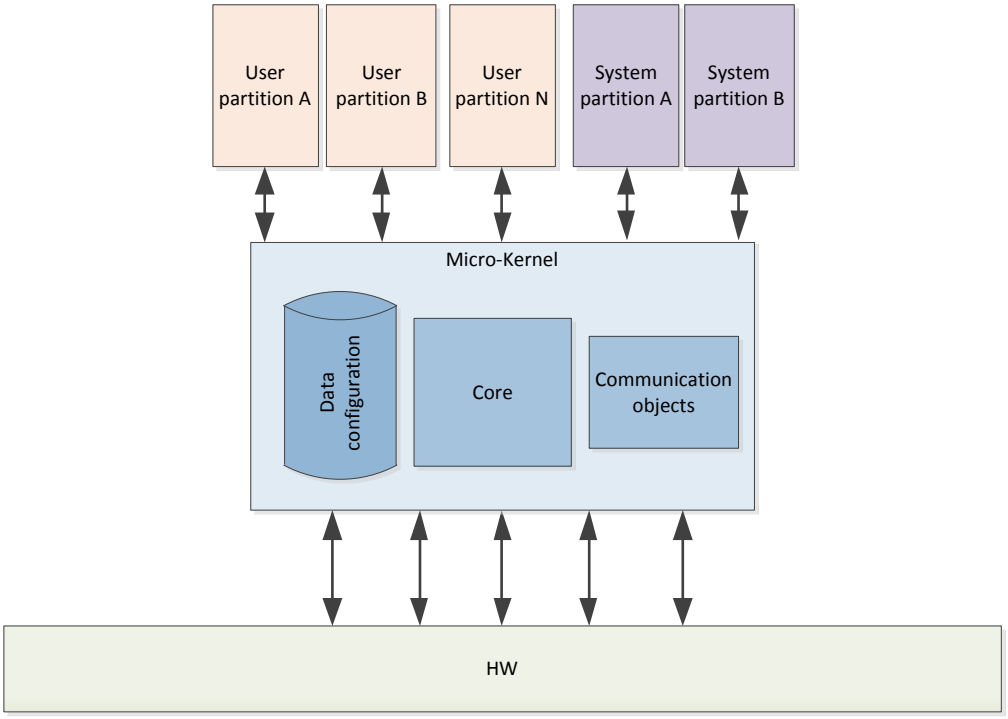


Figure 1: General MILS System

A MILS system is a system that provides a total functional solution implemented through one or several components (trustworthy or not) executed in separated execution environment and communicating through controlled information flows. A MILS system runs on a Hardware Platform and relies on a Separation Kernel that ensures that:

- The total functional solution is non-bypassable, evaluatable, always invoked, and tamperproof
- The Components can be of different level of security/safety and even untrusted. This means that untrusted components cannot compromise the security/safety of trusted components
- Security of each Components can be evaluated independently

When considering MILS systems, we can isolate several components:

- A Hardware platform providing a minimal set of security services as defined in [PP]. The Separation Kernel relies on those services (at least two privilege modes, memory management …) to enforce isolation of the kernel and the execution environments (partitions).
- A Separation Kernel with
    - APIs offered to the applications hosted in the partitions to exchange with the Kernel
    - Communication objects proposed by the kernel to the applications to support inter-partitions communications
    - Configuration data which defines rights and properties of the partitions
    - The core of the Kernel implementing mechanisms to assign resources to partitions, providing the execution environments for applications, and implementing communication between partitions, as defined by the configuration data
- User and System partitions hosting user and system applications. The execution process of the applications inside the partitions is managed by the Kernel. The nature of the applications will be discussed inside the examples of attack but to summarise, an application can be considered as System/User, Close/Open (ie open OS like Android with the possibility to install user applications), Isolated/Connected (ie with existing communication paths to the external world like USB, Bluetooth, Ethernet,…)

NB: the above definition stands for a MILS system and is voluntarily wider than the type TOE (MILS operating system) defined in [PP]. This approach allows having a global view of possible attack scenarios on MILS operating systems after MILS system integration (ie in its final configuration) and allows extending the discussions of the attack methods to MILS systems (not only the MILS OS).

3.2 Template of an attack method

This section provides a template for writing Attack Methods in the context of MILS technology. This template is compatible with JIL (Joint Interpretation Library) format.

3.2.1 Description of Attack

Here lies a short description of the purpose and method of the attack.

3.2.2 Effect of Attack

Here lies a more detailed description of the attack precising how to recognize when the attack has succeeded.

This may include variations of a basic attack (e.g. different fault induction methods).

3.2.3 Impact on TOE

Examples of how the attack may result in an exploitable vulnerability in the TOE. This may explain, for example, how a hardware attack is used in the context of software (or how an attack step such as breaching a tamper-responsive enclosure may be developed into an attack that compromises assets).

It may be useful to think of this description in terms of issues that would need to be notified to a systems integrator or risk manager who is deciding whether, or how, to use the TOE. These same

issues should therefore be checked by an evaluator in a composite evaluation based on the current TOE.

This will also explain the motivation for carrying out this attack.

3.2.4 Characteristics of the Attack

- *Factors that make the attack difficult or easy to carry out, or difficulties to apply to a real TOE.*
- *Skills and tools required to carry out the attack, in the terminology of the attack potential document.*
- *References to books, papers or standard methods, are appropriate. These references will probably not be complete – more techniques are used in evaluation labs than are published – but they may give an understanding of the basics of the attack or attack techniques.*

As a summary of the section, it is useful to include a list of "obvious attacks", representing the basic ways in which the attack should be considered for all individual TOEs.

3.2.5 Examples of Attack Potential Ratings

This will give examples to help agree consistent ways to interpret the MILS attack.

Factor	Comment	Identification	Exploitation
Elapsed time			
Expertise			
Knowledge of the TOE			
Window of opportunity			
Equipment required			
Sub-total			
Total			

3.3 Attacks towards the Separation Kernel

This section contains some example attacks relying mainly on software skills and technics. Those attacks are directed to the Separation Kernel itself and take into account the Separation Kernel attack surface (APIs, Interrupts, Exceptions, Communication objects, configuration data…).

In this section, we are focusing only on the Separation Kernel. We are then considering that we can be in the following possible contexts:

- Evaluation of a Separation Kernel alone, independently of the underlying hardware and of the future specific applications that will be integrated into the final MILS system (Same context as in [PP]).
- Evaluation of a MILS Platform (Hardware + Separation Kernel), independently of the future specific applications that will be integrated into the final MILS system.
- Evaluation of a MILS System, with the assumption that the Separation Kernel attack surface is accessible. This may be the case for MILS systems with open partitions in which the user is authorised to install applications, or in case of combined attack when an attacker is first capable of installing his own hostile application inside a partition (then using this hostile application as a relay for attacking the kernel)

NB: When evaluating a complete MILS System, the attack examples presented in this section may need preliminary attack steps like installation of a hostile application inside an open partition. Some examples of such preliminary steps are discussed in section Chapter 2. Depending on the type of MILS System, those preliminary steps may be complicated or even not practical and the total effort for the complete attack path shall be taken into account.

Attacks towards the Separation Kernel could be:

- From one partition, try to directly access RAM memory address reserved to other protected partition, system partition, or system core in order to eavesdrop or modify sensitive data.
- Accessing intensively hardware resources (CPU, RAM,..) in order to monopolize computing power and prevent access of other resources to computing power (i.e partial or total DoS)
- Attacking the switch mechanism between partitions in order to prevent access of other resources to computing power (i.e partial or total DoS)
- Fuzzing or focused attack (overflow, out of bounds) on kernel APIs, drivers APIs, or communication objects interfaces
- …

3.3.1 Buffer Overflow or Stack Overflow

NB: The current example is based on the author's anticipation of what could be a typical attack directed toward the Separation Kernel itself. This example may be tailored or completed based on evaluation results performed in the scope of EuroMILS project and not yet finalised at the date of writing the current document.

3.3.1.1 Description of Attack

This attack is applicable to Separation Kernels and more generally to MILS Platforms and MILS Systems (but in that last case, preliminary actions are needed like installing malicious application inside a partition).

Separation Kernels provide the capability to host different applications inside separated partitions. Separation Kernels also provide a set of services to those applications, like services to mediate exchanges of sensitive data between applications hosted in separated partitions.

Unauthorized access to user data or TSF data could be performed by a malicious application through buffer overflow or stack overflow during a system call to Separation Kernel services (like APIs).

3.3.1.2 Effect of attack

Overflow, when not checked by the kernel, can have various effects, such as overwriting existing content in the current stack.

The expected effect by the attacker here is that the malicious application modifies the current execution context and obtains system privileges. For example, lets imagine that the execution rights of the current context (ie attackers partition) is written in the stack; if the attacker can overwrite these rights and put the administrator rights, all operations performed after this operation will have the administrator rights. Other example, the attacker may overwrite a memory location that contains a pointer in memory. The attacker may then control where an application is getting its data from. Final example, the attacker may inject in the stack a modified return address and make the CPU execute attacker's code with system privileges.

Such attack may allow a malicious application to execute virtually every operation and then disclose/modify secret of another application.

Another effect may be that internal data or data that were not presumed to be returned by a command (like system data) are retrieved.

3.3.1.3 Impact on TOE

The attack is directed against other applications installed on the TOE and/or against the Separation Kernel itself. The possible impacts are various:

- Divulgation/Modification of secret data of the target application,
- Divulgation/Modification of secret data of the System (like policies, partitions properties…)
- Modification of application functionalities (like modifying application behaviour)
- Modification of System functionalities (like adding new services to bypass Separation Kernel initial policies)
- …

The potential use of these techniques is specialized and has to be considered in the context of each evaluation.

In the scope of evaluation of MILS System, the internal representation of application data is application dependent and may be public or not. If it is not public, the attacker should have critical knowledge of the targeted application to interpret the retrieved data. If it is public, the attacker can derive the meaning of the data without additional effort.

In the scope of evaluation of the Separation Kernel only or MIL Platform only, we cannot presume what will be the final application. In that case, internal representation of application data shall be considered as public (worst case) unless specific recommendations are provided in security guidance of the kernel.

3.3.1.4 Characteristics of the attack

This type of attack requires detailed information on the Separating Kernel APIs, and could require the development of two test applications for the identification phase (one acting as an attacker in a first partition and one acting as a target in a second partition).

At Separation Kernel level and MILS Platform level, the goal of the attack is to check the separation properties of the kernel. The attack is considered as successful when the hostile application is able to read some data from the target partition. In the case of MILS System evaluation, the knowledge of the internal representation of the target partition data may be necessary. If the internal representation of data is not public, the attacker needs knowledge of the TOE to interpret the retrieved data, or needs to derive the meaning of the data by experimental analysis on real product (reverse engineering, tests, side channel…).

The equipment needed to develop an attack application depends on the product type (development environment for the host kernel, development environment for application hosted in a partition…).

Countermeasure:

First counter measure is based on good development practices.

Many books and papers have been issued on secure coding (i.e; avoiding pitfalls in writing source code) in order to resist to attempt to overflow.

Separation Kernel entry points shall be robust against buffer overflow and stack overflow attacks.

Vulnerability coming from malicious application may also be covered by organisational means (only evaluated and signed applications can be installed) and technical means (secure boot, integrity check…).

The overall flaw management / patch management of the Separating Kernel and the applications is also another organisational measure to counter such threats.

3.3.1.5 Examples of Attack Potential Ratings

For this example we assume a standard and well known Separation Kernel, running on standard hardware architecture (eg x86), with at least two user partitions.

All three possible evaluation contexts will be considered:

- Attack directed toward the kernel in the scope of the evaluation of a Separation Kernel (the TOE is the separation kernel as in [PP])
- Attack directed toward the kernel in the scope of the evaluation of a MILS Platform
- Attack directed toward the kernel in the scope of the evaluation of a MILS system

The attacker is targeting data hosted in a target user partition isolated from other partitions.

A hostile test application is used inside an attacker user partition isolated from the other partitions. In the case of a final MILS system, the attacker effort needed to install such application inside one partition must be taken into account.

The target user partition may also be instrument (ie installed with dedicated test applications) in order to survey, for example, the manipulation of the target partition data.

The attack consist in performing a system call (with the help of the hostile application) to one function of the separation kernel API but deliberately using longer arguments than expected thus injecting attacker's data in the stack.

The exploit allows deactivation of the MPU before returning to normal execution flow.

The hostile application can now read the sensitive data in the memory area of the target partition.

The following table provides an example quotation in the scope of the evaluation of a Separation Kernel or a MILS Platform.

i) Rating justification for Separation Kernel or MILS Platform evaluation (standard hardware platform and standard separation kernel):

- Time elapsed: For identification, the development of the attack is expected to last between one week and one month. The attacker needs to identify the adequate kernel API function, develop a hostile application, tune the exploit, and dump the target partition memory content. For exploitation, once the weak API has been identified, the exploit is only based on the hostile application developed during the identification phase and may last less than one hour. We are not discussing here the effort needed by an attacker to install the hostile application on an integrated MILS system.
- Expertise: Identification: Only an expert with very good knowledge of the used Separating Kernel is expected to be able to prepare such an attack. Development skills for the development of the hostile application on the separation kernel are also expected. For exploitation, lower (or even no) expertise is needed for exploitation.
- Knowledge of the TOE: For identification, restricted knowledge of the TOE (functional specifications, guidance) is needed (kernel APIs, kernel MPU management functions). For exploitation: here also, the exploit is only based on the hostile application developed during the identification phase. No specific additional knowledge is necessary for the exploitation phase.
- Equipment required: Identification: As the separation kernel and the hardware platform are considered as standard, associated development tools are also considered as standard. The hostile application is also developed using standard tools. Exploitation: no equipment required.
- Windows of opportunity: Identification: This factor largely depends on the type of product under evaluation. For this example we have assumed that the hardware platform and the separation kernel are standard products that can be easily obtained by an attacker. The experimentation on the APIs of the separation kernel may be performed on any test system integrated by the attacker with little effort. Exploitation: this will depend on the final application which is unknown here. For this example we will consider the case of a Separation kernel or MILS Platform evaluation for the automotive domain in which genuine equipment is considered as easy to obtain.

Factors	Identification	Exploitation
Time elapsed	3	0
Expertise	2	1
Knowledge of the TOE	2	0
Equipment required	3	0
Windows of opportunity	2	1
Total	**14**	

ii) Rating justification for MILS System evaluation (standard hardware platform and standard separation kernel but with specific applications; typically automotive):

- Time elapsed: For identification, same as above but the attacker also needs to identify the researched sensitive data in the dumped data. For exploitation: same as above. NB: If the sensitive data is a cryptographic key not diversified, there is no exploitation phase (the key found during identification is directly the one that is researched by the attacker).

- Expertise: For identification, same as above but expertise regarding the targeted application domain may also be needed. For exploitation, same as above.
- Knowledge of the TOE: For identification, same as above but knowledge about the data organisation in the target partition is necessary to retrieve the targeted sensitive data. For exploitation, same as above.
- Equipment required: Same as above.
- Windows of opportunity: For identification, the attacker may need to experiment on genuine equipment rather than on a test platform, which could be more difficult to obtain (this will depend on the type of MILS system). For Exploitation, if some specific application data are targeted, the attacker may need to perform the exploitation on genuine equipment. For the current example, it is not considered as very difficult to obtain such genuine equipment (purchase or complicity with vendor)

Factors	Identification	Exploitation
Time elapsed	3	0
Expertise	2	1
Knowledge of the TOE	3	0
Equipment required	3	0
Windows of opportunity	2	1
Total	**15**	

As shown in the previous quotation tables, the TOE is considered as resistant to an attacker with a Basic attack potential. We must remind here that if we consider a final MILS System, the attack steps needed to install the hostile application has not been taken into account. The quotation for those attack steps shall be added to the quotation provided here in order to obtain the quotation of a complete attack path on a MILS system.

The first example (i) can be further tailored by considering that the hardware platform is very specific with a custom processor and a dedicated Separation Kernel. Due to this the Identification phase is expected to be harder as it needs very specific knowledge on the hardware and critical design information on the separation kernel itself.

iii) Rating justification for Separation Kernel or MILS Platform evaluation (very specific hardware platform and specific separation kernel):

- Time elapsed: Identification: same as above plus the attacker needs much time for hardware and software design analysis and/or reverse engineering. Exploitation: same as above.
- Expertise: Identification and Exploitation: Same as above. NB: Multiple-Expert is not considered here because only one technical domain is implied. Multiple experts could be considered if important Cryptographic or Microelectronic skills were implied.
- Knowledge of the TOE: Identification and Exploitation: Same as above.
- Equipment required: Identification: It is expected here that very specific tools (like home-made development tools) are necessary for the Identification phase. Exploitation: no equipment required, the exploit has been developed during Identification.
- Window of Opportunity: Identification: Considering that the hardware platform and the separation kernel are very specific products with controlled distribution, a test platform is

considered as very difficult to obtain. The attacker may need to perform his tests on operational equipment with restricted access. Exploitation: the same constraints apply for identification and exploitation.

Factors	Identification	Exploitation
Time elapsed	5	2
Expertise	2	1
Knowledge of the TOE	3	0
Equipment required	5	0
Windows of opportunity	6	5
Total	**29**	

In this last example, the TOE is now considered as resistant to an attacker with Moderate - attack potential. Again, wet remind here that an attack path on a complete MILS system will require additional attack steps like installing the hostile application into an open partition. The quotation for those attack steps shall be added to the quotation provided here in order to obtain the quotation of a complete attack path on a MILS system. We can see in that case that any additional effort needed to install the hostile application inside a partition will most likely rise the quotation above 30 points leading to a MILS system resistant to an attacker with a High Attack potential.

3.4 Attacks towards MILS Platform

This section contains some example attacks relying both on software and/or hardware skills and technics. Those attacks are directed to the MILS platform made of a Hardware layer with the MILS OS integrated on it and take into account the Hardware Platform attack surface (DMA controllers, peripheral coprocessors, network interfaces, local interfaces…). The attacks directed to the separation kernel itself are applicable in the scope of the evaluation of a MILS Platform but they have been treated in section 1.3.3.

In this section, we are focusing on attack methods allowing bypassing the separation properties of the Kernel by using the hardware platform properties. We are then considering that we can be in the following possible contexts:

- Evaluation of a MILS Platform (Hardware + Separation Kernel), independently of the future specific applications that will be integrated into the final MILS system.
- Evaluation of a MILS system (or product), with a specific architecture making it possible for an attacker to get access to the Hardware platform attack surface. This may be the case for MILS systems with open partitions that have access to some hardware devices and in which the user is authorised to install some applications, or in case of combined attack when an attacker is first capable of installing his own hostile application inside a partition (then using this hostile application as a relay for attacking the kernel).

When evaluating a complete MILS system, the attack examples presented here may need additional effort like installation of a hostile application inside the partitions of the MILS product. Those methods are discussed in other sections. Depending on the type of MILS system, they may be complicated or even not practical and the total effort for the complete attack path shall be taken into account.

Attacks directed to the MILS Platform could be:

- Use of known hardware bugs (e.g. in the MPU management) in order to bypass the kernel separation properties
- Use of architectural properties (e.g. DMA, shared memory, , programmable coprocessor…) in order to bypass kernel separation properties.
- Sniff hardware buffers (e.g; keyboard, USB, network interface,…) to eavesdrop user data from other partitions
- Use of external network interfaces or removable device interfaces of the platform in order to try to bypass the kernel separation properties
- Use of dedicated software in order to virtualize the separation kernel in order to inspect (or modify) the accesses made by the software (kernel or partition application) to the Hardware
- Modify the configuration data or the binary image of the kernel before it is launched
- Physical attacks which will require a physical access to the MILS Platform (and so which applicability will depend on the assumption regarding the operational environment of the final product)
    - Exploit hardware intrinsic physical weakness (sensitivity to power-glitch or electromagnetic fault injection to disturb MPU, power or electromagnetic leakage analysis…)
    - Bus probing of hardware platform in order to spy data exchanged between memory and CPU
    - Exploitation of Tests features (like jtag connectors)
    - Reverse engineering, extraction of memory content, replacement of components…

NB: Physical attacks are not analysed in the current document. Those attacks rely on specific skills and some example of attack methods can be found in supporting documents for other IT security domains like Smartcards and similar devices, Point of Interaction, or Hardware devices with security boxes.

3.4.1 Network attacks (Hardware Platform)

This section contains some example of remote attacks based on the exploitation of weaknesses in the management of the interfaces. Two different cases can be considered. The network layer targeted here is not the application layer (that may be managed within user partitions) but lower or intermediate layers that are managed by the kernel and/or dedicated drivers integrated to the MILS OS.

Attacking a network layer means try to exploit protocol weaknesses or gaps in order to compromise components (drivers, kernel) managing one or several layers of the ISO/OSI model. In that case, the component in charge of the network management may run in a system partition thus providing the attacker with privileged right.

The following assumptions are done:

- Physical layer is managed by hardware (possibly within the SoC) and by hypervisor but physical layer attacks should be considered in a physical security scenario, so are not considered here

3.4.1.1 Description of Attack

Exploit protocol weaknesses or gaps while connecting to a network interface.

This attack consists in attacking an open partition of the MILS system (open meaning connected to external untrusted network).

This allows gaining high privileges on this partition of the MILS system and installing specific tools to ease attacks against the Separating Kernel.

3.4.1.2 Effect of Attack

Exploits of protocol or protocol implementation weakness can potentially occur on any protocol layer. The protocol stack may lose control of execution (crash or stuck), relinquish secret information or execute arbitrary code in the context and with the permissions of the protocol driver component. Exploits can be triggered sending manipulated network data packets or a sequence of packets which e.g. do not conform to the protocol standards.

Any software installed in the open partition which opens is vulnerable to attack from external untrusted network.

3.4.1.3 Impact on TOE

Protocol implementation weaknesses are usually based on missing checks or faulty error handling of input data. Effects to the protocol stack can be buffer overflows (and possibly resulting execution of injected code), wrong decisions, deadlock or endless looping, unintended responses on the network or simple crashes.

A successful attack can open another attack door, e.g. to follow more sophisticated attack targets. Attacks on the network interface can reach a partition, but not directly the TOE.

3.4.1.4 Characteristics of the Attack

High level network protocol based attacks are easy to deploy, and many devices can be attacked simultaneously. The attacking network packets are conforming to the low level protocol standards and are hence not detected by network routers. Low level protocol exploits, in contrast, need special network devices that push non-standard network MAC packets, and hence are difficult to deploy. They can only attack devices inside the range of the primary network interface, so the number of devices that can be attacked simultaneously is limited. Low level network protocol exploits are hard to identify, since most of the functionality of this layer is realized in hardware, or, if implemented in software, the code base is small.

3.4.1.5 Examples of Attack Potential Ratings

High level network protocol attacks

The network connection is based on TCP/IP, which is a well-known protocol standard. Services are implemented on top of the session layer using proprietary protocol definitions combined with data encryption (e.g; TLS/SSL).

Once an exploit has been identified, it is easy to deploy, since the attack itself can be performed automatically or semi-automatically. However, since regular software updates fix security issues once they have been found, most identified attack methods will probably be closed within some time.

The ratings we give for high level network protocol attacks are justified as follows:

- Time elapsed: Low level network attacks are considered as much more complicated to put in place. Even with detailed knowledge of the protocols used, it would take more than one month

to find a weakness that can be exploited. Once such a weak spot has been identified and turned into an attack, the attack execution itself probably may be executed in seconds range.

- Expertise: Multiexpert level is chosen here because low level network attack will imply a network expert but also probably a hardware expert with hardware knowledge and development skills (see “Equipment”). The exploitation can be supported by tools that can abstract some complexity. Therefore the required expertise is reduced for the exploitation phase.
- Knowledge of the TOE: For the identification of network exploits detailed information about the used TCP/IP stack version or the higher proprietary protocols are needed. For the exploitation the knowledge of the TOE can be covered by tools. Therefore the required knowledge of the TOE is reduced for the exploitation phase.
- Equipment required: To identify network exploits, specialized equipment like computer with debugger/network analysis software with dedicated plugins capable of managing proprietary protocols are needed. If debug interfaces are permanently disabled in delivered targets, a development board variant of the TOE is needed to do in-situ debugging. Exploits can be performed by specifically prepared computers connected to the Internet.
- Windows of opportunity: The probability to discover a new network exploit seems to be larger for new products rather than for products that have been established and maintained for some time. In the case that a network exploit has been identified, it is deployable until it has been detected and fixed by security maintainers. This window is basically large in months or years scale for the identification, but may reduce to the days or weeks scale once it has been deployed.

Factors	Identification	Exploitation
Time elapsed	5	0
Expertise	5	1
Knowledge of the TOE	3	0
Equipment required	5	2
Windows of opportunity	2	1
Total	**24**	

3.4.2 Example of attacks towards programmable coprocessors

3.4.2.1 Description of Attack

This section contains some example of attacks based on the exploitation of programmable coprocessors or DMA controller in order to bypass the separation properties of the kernel. The prerequisite for such an attack is that the attacker has an access to such coprocessor or DMA controller. In the perspective of a final MILS product, this can be the case when open partition with access to display controller (typically Android) is present.

The MMU will protect forbidden access to RAM areas at CPU level (ie when the code of an application running in an attacker partition tries to access protected memories areas using standards CPU op-codes) but will not prevent a coprocessor from accessing such area as the coprocessor is outside the CPU. This is the role of IOMMUs that can be found on some SoCs (System on Chip) and that plays the role of the MMU but at lower level to prevent peripherals accessing protected memory areas. When the Hardware platform is based on SoCs that contains such IOMMU, those attacks can be

considered as not applicable in the scope of the MILS Platform evaluation and the assumption will be that the IOMMU is correctly used at final MILS system level. In the scope of the evaluation of a MILS system, the evaluator will then have to verify that the IOMMU is correctly configured and that this configuration cannot be altered buy an attacker.

We remind here that when evaluating a complete MILS system, the attack examples presented here may need additional effort like installation of a hostile application inside the open partition and/or getting root access to the coprocessor or the DMA controller. Those methods are discussed in other sections. Depending on the type of MILS system, they may be complicated or even not practical and the total effort for the complete attack path shall be taken into account.

3.4.2.2 Effect of Attack

The protected memory area of the target partition is transferred to a memory area that can be read by the attacker (typically the attacker partition memory area), or the attacker is able to transfer his own data into the protected memory area. This transfer is invisible from the CPU point of view and so cannot be countered by the separation kernel. Only an IOMMU can protect from this.

3.4.2.3 Impact on TOE

With this attack, any data in memory can be read or replaced (application critical data, system configuration data stored in memory, buffers used by the communication channels…). When considering the final MILS system, the knowledge of the targeted data structure may be necessary for correct exploitation.

3.4.2.4 Characteristics of the Attack

Those attacks are applicable only if the attacker is able to program the peripherals. In this chapter we will make the assumption that the attacker has such an access. In a final MILS system, it may be the case that preliminary attack steps are necessary to get access to such peripherals (e.g. get root access to in Android partition).

Different types of peripherals may be used. DMA controller seams the easiest as it may require only programming of a few registers. Video controllers could be used but probably with more programming effort. DSP or cryptographic coprocessors could also be used but probably with even more specific programming effort.

3.4.2.5 Examples of Attack Potential Ratings

Use DMA controller to access protected memory area

The DMA transfer mechanism is a well-known mechanism. It is used to delegate the copy of memory areas to a specific controller in order not to overload the CPU with basic tasks. This mechanism is commonly used to transfer automatically sound/video information to sound/video processor. The DMA controller has to be reprogrammed with targeted memory areas and activated. In our case we will consider a well-known hardware platform in the industry but not commonly used in public domain equipment like computers or smartphones.

- Time elapsed: even with detailed knowledge of the hardware platform architecture, it takes probably more than one week to be able to find a way to reconfigure the DMA controller in an exploitable way (identification). The attack exploitation itself may be executed in seconds range.
- Expertise: The identification of the attack can only be performed by experts knowing how DMA transfers work and capable of programming a DMA controller on the targeted hardware platform. The platform is not common to the public domain but is well known in the industry

domain (expert). The exploitation can be supported by tools that can abstract some complexity. Therefore the required expertise is reduced for the exploitation phase (proficient).

- Knowledge of the TOE: For the identification the attacker must know the hardware architecture of the platform, and must know (or find) how to access the DMA controller (addresses of the registers). Such level of information can be considered as restricted if the platform manufacturer has a serious customer control policy. When targeting a precise data in a precise partition or in the system area, the attacker must also know the architecture of the final MILS product (sensitive). For the exploitation the knowledge of the TOE can be covered by tools. Therefore the required knowledge of the TOE is reduced for the identification
- Equipment required: For identification, equipment required is the SdK for the application (OS) running into the attacker partition. In case of MILS platform evaluation, the application will be a test application (Linux or Android) with freely available SdK (Standard). In case MILS system evaluation and if the open partition runs an exotic OS, such SdK may be harder to find. Also, if some specific system data are targeted, a preliminary attack may be necessary in order to find where in memory are stored those data. For exploitation, no specific equipment is used as the hostile application has already been developed.
- Windows of opportunity: In the case of a MILS platform evaluation, we will consider an unlimited access (worst case) as we cannot presume of the context of use of the future MILS system (unless the MILS platform is dedicated to a specific domain). In case of a MILS system, this will completely depend on the use case and application domain. If we consider the automotive domain, windows of opportunity may be considered as easy for both identification and exploitation. If we consider the avionic domain, windows of opportunity may be considered as difficult for both identification and exploitation.

Example rating for MILS platform

Factors	Identification	Exploitation
Time elapsed	3	0
Expertise	2	1
Knowledge of the TOE	2	0
Equipment required	1	0
Windows of opportunity	0	0
Total	**9**	

Example rating for MILS system with Android running in the attacker partition (automotive)

Factors	Identification	Exploitation
Time elapsed	3	0
Expertise	2	1
Knowledge of the TOE	2	0
Equipment required	1	0
Windows of opportunity	2	1
Total	**12**	

Example rating for MILS system with Android running in the attacker partition (avionics)

Factors	Identification	Exploitation
Time elapsed	3	0
Expertise	2	1
Knowledge of the TOE	2	0
Equipment required	1	0
Windows of opportunity	6	5
Total	**20**	

NB: when considering a MILS system, the quotation provided above must be added to the quotation for taking the control over the open partition (if additional effort is needed for that). Example of such attacks is provided in chapter 1.3.6.1.

3.5 Attacks towards MILS Systems

This section contains some example attacks relying on software skills and technics. Those attacks are directed to the final MILS product (Hardware layer with the MILS OS and the final functionality dispatched through partitions) and take into account the Product attack surface (open partition, post issuance update capabilities, network interfaces, local interfaces…). The attacks directed to the Separation Kernel itself and to the hardware platform are applicable in the scope of the evaluation of a MILS Product but they have been treated in section 1.3.3 and 1.3.4.

In this section, we are focusing on attack methods directed to the MILS product by using the characteristics of the product. We are then considering that we can be in the following possible contexts:

- Evaluation of a MILS system (or product), with a specific architecture making it possible for an attacker to get access to the Hardware platform attack surface or the Separation Kernel attack surface. This may be the case for MILS systems with open partitions that have access to some hardware devices.

It is clear that we cannot presume here about the final functionality of the product. The robustness of the functionality against attackers will depend on the architecture of the product, how the functionality is dispatched into partitions, and how the separation properties and communication channels are used. This analysis must be performed in a case by case approach and it could be that due to poor architecture, the product functionality is defeated or bypassed even without bypassing the separation properties of the kernel.

Nevertheless, the goal of this chapter is to provide some typical attack paths that may be common to a large amount of MILS product independently of the product functionality.

When evaluating a complete MILS system, the evaluator shall consider the attack examples presented here and analyse whether they are applicable or not and whether the can combined with attacks at Hardware Platform or Separation kernel level. Depending on the type of MILS system, they may be complicated or even not practical and the total effort for the complete attack path shall be taken into account.

Attacks directed to the MILS System could be:

- Install a malware in an open partition (i.e. running an OS that allows installation of user application) as a proxy to attack the hardware or the kernel
- Software Side channel analysis
- Attack applications specific mechanisms

3.5.1 Network attacks (MILS Systems)

This section contains some example of remote attacks of a MILS System based on the exploitation of weaknesses in the management of the interfaces. More precisely, the idea is to use network (Ethernet, WiFi, 4G, Bluetooth…) interface to install a malware in an open partition (i.e. running an OS like Linux or Android). Then this malware can be used as a relay to attack the separation kernel or the hardware platform (see chapters 1.3.4 and 1.3.5).

A network attack can potentially involve any layer of the ISO/OSI model, thus every component which manages one or more layers can be targeted. The attack we are considering here targets the layer that is managed inside the open partition. The following assumptions are done:

- Physical layer is managed by hardware (possibly within the SoC) and by hypervisor but physical layer attacks should be considered in a physical security scenario, so are not considered here

- Upper layers are managed by open partition. If some layers are managed by the system or a system partition, an attack against that layer could become a single-step-attack against the system assets (see chapter 1.3.4.1).

3.5.1.1 Description of Attack

It targets the open partition from external untrusted networks.

The attacker can be remote through the WAN network devices (G2/G3/G4) or close to the product through the LAN network devices (Bluetooth, Wi-Fi) and can conduct a big variety of attacks according to the network used.

3.5.1.2 Effect of Attack

Exploits of protocol or protocol implementation weakness can potentially occur on any protocol layer. The protocol stack may lose control of execution (crash or stuck), relinquish secret information or execute arbitrary code in the context and with the permissions of the protocol driver component. Exploits can be triggered sending manipulated network data packets or a sequence of packets which e.g. do not conform to the protocol standards.

Any software installed in the open partition with networking capabilities is vulnerable to network application-level attack.

3.5.1.3 Impact on TOE

Protocol implementation weaknesses are usually based on missing checks or faulty error handling of input data. Effects to the protocol stack can be buffer overflows (and possibly resulting execution of injected code), wrong decisions, deadlock or endless looping, unintended responses on the network or simple crashes.

A successful attack can open another attack door, e.g. to follow more sophisticated attack targets..

3.5.1.4 Characteristics of the Attack

High level network protocol based attacks are easy to deploy, and many devices can be attacked simultaneously. The attacking network packets are conforming to the low level protocol standards and are hence not detected by network routers. Low level protocol exploits, in contrast, need special network devices that push non-standard network MAC packets, and hence are difficult to deploy. They can only attack devices inside the range of the primary network interface, so the number of devices that can be attacked simultaneously is limited. Low level network protocol exploits are hard to identify, since most of the functionality of this layer is realized in hardware, or, if implemented in software, the code base is small.

3.5.1.5 Examples of Attack Potential Ratings

High level network protocol attacks

The network connection is based on TCP/IP, which is a well-known protocol standard. Services are implemented on top of the session layer using proprietary protocol definitions combined with data encryption (e.g; TLS/SSL).

Once an exploit has been identified, it is easy to deploy, since the attack itself can be performed automatically or semi-automatically. However, since regular software updates fix security issues once they have been found, most identified attack methods will probably be closed within some time.

The ratings we give for high level network protocol attacks are justified as follows:

- Time elapsed: even with detailed knowledge of the protocols used, it takes for sure more than

one week to find a weakness in the protocol stack that can be exploited. Once such a weak spot has been identified and turned into an attack, the attack execution itself probably may be executed in seconds range.

- Expertise: The identification of network exploits can only be performed by experts (network protocols and algorithms). The exploitation can be supported by tools that can abstract some complexity. Therefore the required expertise is reduced for the exploitation phase.
- Knowledge of the TOE: For the identification of network exploits detailed information about the used TCP/IP stack version or the higher proprietary protocols are needed. For the exploitation the knowledge of the TOE can be covered by tools. Therefore the required knowledge of the TOE is reduced for the exploitation phase.
- Equipment required: To identify network exploits, specialized equipment like computer with debugger/network analysis software with dedicated plugins capable of managing proprietary protocols are needed. If debug interfaces are permanently disabled in delivered targets, a development board variant of the TOE is needed to do in-situ debugging. Exploits can be performed by specifically prepared computers connected to the internet.
- Windows of opportunity: The probability to discover a new network exploit seems to be larger for new products rather than for products that have been established and maintained for some time. In the case that a network exploit has been identified, it is deployable until it has been detected and fixed by security maintainers. This window is basically large in months or years scale for the identification, but may reduce to the days or weeks scale once it has been deployed.

Factors	Identification	Exploitation
Time elapsed	3	0
Expertise	2	1
Knowledge of the TOE	3	0
Equipment required	3	2
Windows of opportunity	2	1
Total	**17**	

3.5.2 Malwares & Root Kits

This section contains an example of first step attack that consists in installing malicious software inside an open user partition. This malicious software can then operate autonomously or operates as a relay for a second step attack against the separation kernel or the hardware platform (see sections 1.3.3 and 1.3.4).

3.5.2.1 Description of Attack

The attacker injects the malware into the open partition by an infected USB drive or from the external networks through malicious downloading, social engineering, etc. In this example, the installation of the malware is considered as easy (i.e. no preliminary attack step to hack the network stack)

As soon as the malware is executed the attack begins and it finishes when privilege escalation has reached.

3.5.2.2 Effect of Attack

The malware type and the way it performs privilege escalation is strongly related to the open partition system.

Triggering the execution of the malware is the first step that can be automatic through some system or application event or induced by the user.

If the execution is already in privilege mode so the privilege escalation is reached, otherwise the malicious actions are limited by the last privilege policies (memory and file-system permissions, inter-process communications rules, etc.) and access control mechanism in place: overrunning them is the malware core operation.

Generically the core operation can be abstracted as a 0-day exploit, which makes the malware theoretically invisible to pattern-based malware recognition systems but not to behaviour-based ones.

3.5.2.3 Impact on TOE

No TOE partitions should expect retrieving files (especially executable) from external untrusted parties; an exception would be a Software Update mechanism receiving downloaded data. Such data shall be protected authentication, and integrity mechanisms (even with encryption).

Thus the impact should be limited to the Open Partition, which can be fully taken over after a privilege escalation and its malicious capabilities make persistent if the malware is a rootkit.

3.5.2.4 Characteristics of the Attack

The likelihood of malware activities, especially in the user open partitions, has to be considered with high likelihood if no mitigation mechanism is in place.

Mitigation mechanisms can be of two kinds: i) secure boot and chain of trust for integrity verification at start-up, and/or ii) dynamic pattern-based or behaviour-based malware recognition system.

3.5.2.5 Examples of Attack Potential Ratings

In the table the rating is referred to a rootkit which apply a 0-day exploit.

- Time elapsed: Identification of a 0-day exploit is a long task research and is expected to last more than one month. 0-day can be bought but still some development is needed for adaptation lowering the time to lower than one month but more than one week. Once the exploit has been created, the attack execution itself probably may be executed in seconds range.
- Expertise: The identification or adaptation of a 0-Day can only be done by an expert. For exploitation, the exploit is supposed to be automated but may rely on minimal technical infrastructure for deployment and exploitation that cannot be handled by a layman, so proficient level is chosen.
- Knowledge of the TOE: Restricted information is chosen for identification because the attacker

needs to know some high level information about the open partition. No more information needed for exploitation.

- Equipment required: Discovering a 0-Day is supposed to be a complicated task based on some home-made specific tools. Also buying a 0-day is expensive on the black market and may need adaptation, further development, and testing with specialized tools. Bespoke is retained for identification. Exploit is unintentionally triggered by the user plugging USB key or downloading infected application.
- Windows of opportunity: The probability to discover a new OS exploit seems to be larger for new products rather than for products that have been established and maintained for some time. In the case that an exploit has been identified, it is deployable until it has been detected and fixed by security maintainers. This window is basically large in weeks or months for the identification, but may reduce to the days or weeks scale once it has been deployed. Regarding the application domain, we will consider that the TOE is a MILS system for the automotive domain with rather easy access.

Factors	Identification	Exploitation
Time elapsed	5	0
Expertise	2	1
Knowledge of the TOE	2	0
Equipment required	5	0
Windows of opportunity	2	1
Total	**18**	

Chapter 4 Refinement of the CEM

During a CC evaluation, ITSEFs and certifiers refer to the Common Evaluation Methodology [CEM] document.

This document recalls the evaluation activities that need to be performed by ITSEFs depending on the assurance level claimed in the Security Target.

The Common Evaluation Methodology [CEM] recognises that not all questions concerning IT security evaluation are answered in the document and that further interpretations will be needed.

Indeed, concerning evaluations according to CC version 3.1, the CEM version 3.1 includes already the methodology for the assurance components related to EAL5, but the methodology for EAL6 and above is not completely defined.

For evaluations according to EAL6 or above or if those components are used at a lower assurance level for augmentation, a methodology still needs to be defined.

This document is a suggestion of the EuroMILS partners to address the missing work units not defined in the CEM in the context of the technology of Separation Kernel (SK).

4.1 ADV class

4.1.1 Evaluation of sub-activity (ADV_SPM.1)

4.1.1.1 Objectives

The objectives of this sub-activity are to determine whether the formal security policy model of the TSF clearly and consistently describes the rules and characteristics of the security policies and whether this description corresponds with the description of security functions in the functional specification.

The related part of the AIS34 from the German Scheme and the Note12 from the French Scheme were used as the basis here.

4.1.1.2 Input

The evaluation evidence for this sub-activity is:

- the ST,
- the functional specification,
- The source code of the formal model (named SRC),
- The proofs associated to the formal model (PROOF),
- A global document (ARG) which details the choices taken on the formal model and more specifically:
    - An argumentation of the confidence given on the methodology and tools associated with the formal approach (ARG_TOOL),
    - An explanation of the formal model (design, conception, …) and the diverse properties and notions used in the formal model (ARG_SPM),
    - An argumentation on the link between the formal model and the security target and more precisely the link between the characteristics (and/or rules) and the features

(and/or properties) (ARG_CDS). This argumentation should be based on the traceability of requirements and shall define precisely the perimeter of the model.
    - A complete list of the hypotheses of the model (with justifications) (ARG_PROOF). This document shall demonstrate that the hypotheses considered are not inconsistent. This document complements ARG_SPM: it underlines the hypotheses resulting from developer choices of the model design,
    - A correspondence between the formal model and the specification of the TOE ARG_FSP. The formalism used for this trace is defined by the evaluation criteria. At least, it seems necessary to realize traceability between FSP requirements and formal model constructions.
- The tools used to support the formal method shall be delivered to the evaluator.

4.1.1.3 Application notes

This activity applies to cases where the developer has provided a formal security policy model of the TOE.

The creation of a formal security policy model helps to identify and eliminate ambiguous, inconsistent, contradictory, or unenforceable security policy elements. Once the TOE has been built, the formal model serves the evaluation effort by contributing to the evaluator's judgment of how well the developer has understood the security functionality being implemented and whether there are inconsistencies between the security requirements and the TOE design. The confidence in the model is accompanied by a proof that it contains no inconsistencies.

4.1.1.4 Action ADV_SPM.1.1E

ADV_SPM.1.1C The model shall be in a formal style, supported by explanatory text as required, and identify the security policies of the TSF that are modelled.

ADV_SPM.1-1 The evaluator shall examine the TOE security policy model to determine that it is written in a formal style.

This work unit requires the source code of the model SRC and the justification of the formal method used shall be supplied in ARG_TOOL. The evaluator identifies the formal framework upon which the TOE security policy model is based and ensures that it is founded on well-established mathematical concepts. The justification and the documentation of the formal method shall include explanations about common and classical difficulties or traps of the formal tools to facilitate the review of the evaluator. He also identifies the security properties and features addressed in the ST and he ensures that the formalization of at least one security policy is present in the model.

ADV_SPM.1-2 The evaluator shall examine the TOE security policies model description (ARG_SPM) to determine that it contains all necessary explanatory text.

Supporting narrative descriptions are necessary for all parts of the model (for example, to make clear the meaning of any formal notation and how they are used) including the security properties and features. The main design principles of the model should be described and should be argumented to give a complete understanding of the abstraction mechanisms of the security policy model. The level of details of these descriptions shall be sufficient to evaluate the consistency of the modelling.

ADV_SPM.1-3 The evaluator shall examine the TOE security policies model to determine that all security policies listed in the ADV_SPM in the ST are modelled.

This activity is mainly based on ARG_CDS with additional information from ST and ARG_SPM. The evaluator shall adopt a very restrictive position: by default all security policies should be described (it is the ideal list) and the developer shall justify the absence of each security policy not considered in the model.

The evaluator determines whether the SPM identifies the security policies for which a model is provided, identifying the relevant portions of the statement of SFRs that comprise each of the modelled policies.

ADV_SPM.1-4 The evaluator shall examine the principles and characteristics of the security policies to determine that the modelled security behaviour of the TOE is clearly articulated.

The security policies are expressed in terms of security principles (or rules) which are modelled by security properties and define the secure state of the TOE. The document ARG_CDS shall supply a correspondence (mainly based on requirement traceability) between informal notions of the ST (principles, characteristics) and their formal counterparts. The perimeter and the level of abstraction of the model shall be justified with regards to the security policies modelled and shall allow a correct identification of all security objectives..

For example, a model based on state transitions could describe the security policies in terms of principles of its states, identify its initial state, and define what it means to be a secure state.

ADV_SPM.1.2C For all policies that are modelled, the model shall define security for the TOE and provide a formal proof that the TOE cannot reach a state that is not secure.

ADV_SPM.1-5 The evaluator shall examine the TOE security policies model (SRC), to determine that it formally proves that the behaviour modelled cannot reach a state that is not secure.

The definition on an unsecure state shall be formally defined [ARG_SPM]. This work unit leads to check the consistency of the identified elements (hypotheses and others …) in [ARG_PROOF] and check the completeness of this document. The proofs realized [PROOF] should be checked by the evaluator with regard to the model source [SRC] and the theory and tools ([ARG_TOOL]).

The evaluator shall replay the formal proofs with the help of the tools supplied by the developer. Moreover, the evaluator shall check that the proofs are completely achieved and that the hypotheses used to conduct the proofs are not inconsistent with the properties of the model.

ADV_SPM.1-6 The evaluator shall examine the TOE security policy model rationale to determine that it proves the internal consistency of the TOE security policy model.

The proof shall show the absence of contradictions within the TOE security policy model. In determining the absence of contradictions, the evaluator verifies that the arguments used in the proof are valid. At least, the assumptions of the model shall be identified and the demonstration of the consistency of these hypotheses shall be demonstrated in ARG_PROOF.

Since the TOE security policy model is formal, the proof of its internal consistency shall be formal. It is recognized that a complete formal proof of the internal consistency of the TOE security policy model usually is not possible due to the fundamental nature of formal frameworks. Generally, it is sufficient to generate evidence using formal proofs based on the specific TOE security policy model that prove the internal consistency by means of a combination with generic arguments of the formal framework.

ADV_SPM.1.3C The correspondence between the model and the functional specification shall be at the correct level of formality.

ADV_SPM.1-7 In case of low level assurance, (i.e. for ADV_FSP.1 to ADV_FSP.4, the evaluator shall examine the correspondence between the model and the functional specification based on traceability requirements in ARG_FSP. All the requirements of the functional specification shall be considered in this traceability.

ADV_SPM.1-8 In case of ADV_FSP.5, the evaluator shall examine the correspondence between the model and the functional specification to determine that a semiformal demonstration of correspondence between the model and any semiformal functional specification is provided in ARG_FSP. The traceability shall be clearly formalized (*) and all requirements of the functional specification shall be associated to semi-formal objects (**) to support the consistency of the semi-formal model. It implies than all part of the semi-formal model is linked to requirements and no requirements are missed in the semi-formal model.

(*): i.e. specifications shall be written in requirements with dedicated identifiers, the traceability scheme shall be identified, the coverage ratio of the complete set of requirements is computable and each non covered requirement is justified.

(**): the term of semi-formal object is a generic term to designate constructions offered by a semi-formal method (for example UML, MatLab Simulink, ...).

ADV_SPM.1-9 In case of ADV_FSP.6, the evaluator shall examine the correspondence between the model and the functional specification to determine that a formal proof of correspondence between the model and any formal functional specification is provided in ARG_FSP.

This proof (part of PROOF should be a refinement proof to demonstrate that the behaviour defined in the security policy model are preserved in the functional security policy model. Additionally the refinement scheme should be described by the developer in ARG_FSP (or in ARG_TOOL) and the evaluator shall validate the consistency of the refinement scheme proposed and proof obligations associated. For the non-formal parts of the model FSP, traceability shall be realized between the requirements of the FSP and the formal SPM model.

ADV_SPM.1.4C The correspondence shall show that the functional specification is consistent and complete with respect to the model.

ADV_SPM.1-10 The evaluator shall examine the functional specification correspondence demonstration of the TOE security policies model to determine that it is complete

The evaluator shall examine the correspondence to determine that the behaviour at the TSF interfaces (as articulated in the functional specification) is complete with

respect to the behaviour modelled by the security features.

ADV_SPM.1-11 The evaluator shall examine the correspondence to determine that the behaviour at the TSF interfaces (as articulated in the functional specification) is consistent with respect to the behaviour modelled by the security features.

ADV_SPM.1-12 The evaluator shall examine the functional specification correspondence demonstration of the TOE security policies model to determine that the descriptions of the functions identified as implementing the TSP model are consistent with the descriptions in the functional specification.

Additionally, the evaluator shall determine that the interfaces described in the functional specification are consistent and complete with the behaviour modeled (e.g. the features).

4.1.2 Evaluation of sub-activity (ADV_FSP.6)

4.1.2.1 Objectives

The objective of this sub-activity is to determine whether the developer has completely described all of the TSFI in a manner such that the evaluator is able to determine whether the TSFI are completely and accurately described, and appears to implement the security functional requirements of the ST. The completeness of the interfaces is judged based upon the implementation representation.

4.1.2.2 Input

The evaluation evidence for this sub-activity that is required by the work-units is:

a) the ST;

b) the functional specification:

➔ Including formal model source code SRC

➔ Proof of this model PROOF

➔ Rationale on level of confidence for the chosen method ARG_TOOL;

➔ Explanatory text of the formal presentation of the TSF and the relationship between the various notions handled in this functional specification ARG_FSP;

➔ Presentation and justification of the “assumptions” (used in the evidence but not themselves proved) that may be introduced in the formal model ARG_PROOF;

c) the TOE design;

d) the implementation representation.

The evaluation evidence for this sub-activity that is used if included in the ST for the TOE is:

a) the security architecture description;

b) the TSF internals description;

c) the formal security policy model;

d) the operational user guidance;

4.1.2.3 Application notes

Regarding MILS systems, TSFI should be the externals APIs provided by the MILS to the application.

4.1.2.4 Action ADV_FSP.6.1E

ADV_FSP.6.1C The functional specification shall completely represent the TSF.

ADV_FSP.6-1 The evaluator shall examine the functional specification to determine that the TSF is fully represented.

The identification of the TSFI is a necessary prerequisite to all other activities in this sub-activity. The TSF must be identified (done as part of the TOE design (ADV_TDS) work units) in order to identify the TSFI. This activity can be done at a high level to ensure that no large groups of interfaces have been missed (network protocols, hardware interfaces, configuration files), or at a low level as the evaluation of the functional specification proceeds.

In making an assessment for this work unit, the evaluator determines that all portions of the TSF are addressed in terms of the interfaces listed in the functional specification. All portions of the TSF should have a corresponding interface description, or if there are no corresponding interfaces for a portion of the TSF, the evaluator determines that that is acceptable.

ADV_FSP.6.2C The functional specification shall describe the TSFI using a **formal** style.

ADV_FSP.6-2 The evaluator shall **examine** the functional specification to determine that it is presented using a **formal** style.

The evaluator shall identify the formal methods and tools used by the developer in order to ensure that the used methods rely on correct theoretical principles. For this the evaluator may use the document [ARG_TOOL] provided by the developer. Evaluator shall identify the common mistakes of this method.

ADV_FSP.6.3C The functional specification shall describe the purpose and method of use for all TSFI.

ADV_FSP.6-3 The evaluator shall examine the functional specification to determine that it states the purpose of each TSFI.

The purpose of a TSFI is a general statement summarising the functionality provided by the interface, **this information can be found for example in the informal explanatory text part of functional specification**. It is not intended to be a complete statement of the actions and results related to the interface, but rather a statement to help the reader understand in general what the interface is intended to be used for. The evaluator should not only determine that the purpose exists, but also that it accurately reflects the TSFI by taking into account other information about the interface, such as the description of actions and error messages.

ADV_FSP.6-4 The evaluator shall examine the functional specification to determine that the method of use for each TSFI is given.

The method of use for a TSFI summarises how the interface is manipulated in order to invoke the actions and obtain the results associated with the TSFI. The evaluator should be able to determine, from reading this material in the functional specification, how to use each interface. This does not necessarily mean that there needs to be a separate method of use for each TSFI, as it may be possible to describe in general how kernel calls are invoked, for instance, and then identify each interface using that

general style. Different types of interfaces will require different method of use specifications. APIs, network protocol interfaces, system configuration parameters, and hardware bus interfaces all have very different methods of use, and this should be taken into account by the developer when developing the functional specification, as well as by the evaluator evaluating the functional specification. **This information can be found for example in the informal explanatory text part of functional specification**.

For administrative interfaces whose functionality is documented as being inaccessible to untrusted users, the evaluator ensures that the method of making the functions inaccessible is described in the functional specification. It should be noted that this inaccessibility needs to be tested by the developer in their test suite.

The evaluator should not only determine that the set of method of use descriptions exist, but also that they accurately cover each TSFI.

ADV_FSP.6-5 The evaluator shall examine the functional specification to determine the completeness of the TSFI.

The evaluator shall use the design documentation to identify the possible types of interfaces. The evaluator shall search the design documentation and the guidance documentation for potential TSFI not contained in the developer's documentation, thus indicating that the set of TSFI defined by the developer is incomplete. The evaluator shall examine the arguments presented by the developer that the TSFI is complete and check down to the lowest level of design or with the implementation representation that no additional TSFI exist.

As the functional specification shall describe the TSFI using a formal style, the verification of the completeness shall be done on formal model and could be proved if refinement of the model is conducted to the source code.

ADV_FSP.6.4C The functional specification shall identify and describe all parameters associated with each TSFI.

ADV_FSP.6-6 The evaluator shall examine the presentation of the TSFI to determine that it completely identifies all parameters associated with every TSFI.

The evaluator examines the functional specification to ensure that all of the parameters are described for each TSFI. Parameters are explicit inputs or outputs to an interface that control the behaviour of that interface. For examples, parameters are the arguments supplied to an API; the various fields in packet for a given network protocol; the individual key values in the Windows Registry; the signals across a set of pins on a chip; etc.

In order to determine that all of the parameters are present in the TSFI, the evaluator should examine the rest of the interface description (actions, error messages, etc.) to determine if the effects of the parameter are accounted for in the description. The evaluator should also check other evidence provided for the evaluation (e.g., TOE design, security architecture description, operational user guidance, implementation representation) to see if behaviour or additional parameters are described there but not in the functional specification.

As the functional specification shall describe the TSFI using a formal style, the verification shall be done on formal model and could be proved if refinement of the model is conducted to the source code.

ADV_FSP.6-7 The evaluator shall examine the presentation of the TSFI to determine that it completely and accurately describes all parameters associated with every TSFI. **This**

description must be performed by using a formal style supported by informal explanatory text where appropriate.

Once all of the parameters have been identified, the evaluator needs to ensure that they are accurately described, and that the description of the parameters is complete. A parameter description tells what the parameter is in some meaningful way. For instance, the interface foo(i) could be described as having “parameter i which is an integer”; this is not an acceptable parameter description. A description such as “parameter i is an integer that indicates the number of users currently logged in to the system” is much more acceptable.

In order to determine that the description of the parameters is complete, the evaluator should examine the rest of the interface description (purpose, method of use, actions, error messages, etc.) to determine if the descriptions of the parameter(s) are accounted for in the description. The evaluator should also check other evidence provided (e.g., TOE design, architectural design, operational user guidance, implementation representation) to see if behaviour or additional parameters are described there but not in the functional specification.

As the functional specification shall describe the TSFI using a formal style, the verification shall be done on formal model and could be proved if refinement of the model is conducted to the source code.

ADV_FSP.6.5C The functional specification shall describe all actions associated with each TSFI.

ADV_FSP.6-8 The evaluator shall examine the presentation of the TSFI to determine that it completely and accurately describes all actions associated with every TSFI.

The evaluator checks to ensure that all of the actions are described. Actions available through an interface describe what the interface does (as opposed to the TOE design, which describes how the actions are provided by the TSF). **This description must be performed by using a formal style supported by informal explanatory text where appropriate.**

Actions of an interface describe functionality that can be invoked through the interface, and can be categorised as regular actions, and SFR-related actions. Regular actions are descriptions of what the interface does. The amount of information provided for this description is dependent on the complexity of the interface. The SFR-related actions are those that are visible at any external interface (for instance, audit activity caused by the invocation of an interface (assuming audit requirements are included in the ST) should be described, even though the result of that action is generally not visible through the invoked interface). Depending on the parameters of an interface, there may be many different actions able to be invoked through the interface (for instance, an API might have the first parameter be a “subcommand”, and the following parameters be specific to that subcommand. The IOCTL API in some Unix systems is an example of such an interface).

In order to determine that the description of the actions of a TSFI is complete, the evaluator should review the rest of the interface description (parameter descriptions, error messages, etc.) to determine if the actions described are accounted for. The evaluator should also analyse other evidence provided for the evaluation (e.g., TOE design, security architecture description, operational user guidance, implementation representation) to see if there is evidence of actions that are described there but not in the functional specification.

As the functional specification shall describe the TSFI using a formal style, the verification shall be done on formal model and could be proved if refinement of the model is conducted to the source code.

ADV_FSP.6.6C The functional specification shall describe all direct error messages that may result from an invocation of each TSFI.

ADV_FSP.6-9 The evaluator shall examine the presentation of the TSFI to determine that it completely and accurately describes all errors messages resulting from an invocation of each TSFI.

Errors can take many forms, depending on the interface being described. **This description must be performed by using a formal style supported by informal explanatory text where appropriate.** For an API, the interface itself may return an error code; set a global error condition, or set a certain parameter with an error code. For a configuration file, an incorrectly configured parameter may cause an error message to be written to a log file. For a hardware PCI card, an error condition may raise a signal on the bus, or trigger an exception condition to the CPU.

Errors (and the associated error messages) come about through the invocation of an interface. The processing that occurs in response to the interface invocation may encounter error conditions, which trigger (through an implementation-specific mechanism) an error message to be generated. In some instances this may be a return value from the interface itself; in other instances a global value may be set and checked after the invocation of an interface. It is likely that a TOE will have a number of low-level error messages that may result from fundamental resource conditions, such as “disk full” or “resource locked”. While these error messages may map to a large number of TSFI, they could be used to detect instances where detail from an interface description has been omitted. For instance, a TSFI that produces a “disk full” message, but has no obvious description of why that TSFI should cause an access to the disk in its description of actions, might cause the evaluator to examine other evidence (ADV_ARC, ADV_TDS) related that TSFI to determine if the description is complete and accurate.

The evaluator determines that, for each TSFI, the exact set of error messages that can be returned on invoking that interface can be determined. The evaluator reviews the evidence provided for the interface to determine if the set of errors seems complete. They cross-check this information with other evidence provided for the evaluation (e.g., TOE design, security architecture description, operational user guidance, implementation representation) to ensure that there are no errors steaming from processing mentioned that are not included in the functional specification.

Evaluator shall examine that all conditions to raise errors messages are part of the functional specification formal model. The proof shall verify that unsecured state cannot be reached with these conditions.

ADV_FSP.6-10 The evaluator shall examine the presentation of the TSFI to determine that it completely and accurately describes the meaning of all error messages resulting from an invocation of each TSFI.

In order to determine accuracy, the evaluator must be able to understand meaning of the error. For example, if an interface returns a numeric code of 0, 1, or 2, the evaluator would not be able to understand the error if the functional specification only listed: “possible errors resulting from invocation of the foo() interface are 0, 1, or 2”. Instead the evaluator checks to ensure that the errors are described such as: “possible errors resulting from invocation of the foo() interface are 0 (processing successful), 1 (file not found), or 2 (incorrect filename specification)”. **This information can be found for example in the informal explanatory text part of functional specification**.

In order to determine that the description of the errors due to invoking a TSFI is complete, the evaluator examines the rest of the interface description (parameter descriptions, actions, etc.) to determine if potential error conditions that might be

caused by using such an interface are accounted for. The evaluator also checks other evidence provided for the evaluation (e.g., TOE design, security architecture description, operational user guidance, implementation representation) to see if error processing related to the TSFI is described there but is not described in the functional specification.

ADV_FSP.6.7C The functional specification shall describe all error messages **contained in the TSF implementation representation**.

ADV_FSP.6-11 The evaluator shall examine the functional specification to determine that it completely and accurately describes all errors messages **contained in the TSF implementation representation**.

This work unit complements work unit ADV_FSP.6-9, which describes those error messages that result from an invocation of the TSFI. Taken together, these work units cover all error messages **contained in the TSF implementation representation**.

The evaluator assesses the completeness and accuracy of the functional specification by comparing its contents to instances of error message generation within the implementation representation. Most of these error messages will have already been covered by work unit ADV_FSP.6-9.

The error messages related to this work unit are typically those that are not expected to be generated, but are constructed as a matter of good programming practices. For example, a case statement that defines actions resulting from each of a list of cases may end with a final else statement to apply to anything that might not be expected; this practise ensures the TSF does not get into an undefined state. However, it is not expected that the path of execution would ever get to this else statement; therefore, any error message generation within this else statement would never be generated. Although it would not get generated, it must still be included in the functional specification.

ADV_FSP.6.8C The functional specification shall provide a rationale for each error message contained in the TSF implementation **that is not otherwise described in the functional specification justifying why it is not associated with** a TSFI.

ADV_FSP.6-12 The evaluator shall examine the functional specification to determine that it provides a rationale for each error message contained in the TSF implementation **that is not otherwise described in the functional specification justifying why it is not associated with** a TSFI.

The evaluator ensures that every error message found under work unit ADV_FSP.6-11 contains a rationale **justifying why it is not associated with** the TSFI.

As was described in the previous work unit, this rationale might be as straightforward as the fact that the error message in question is provided for completeness of execution logic and that it is never expected to be generated. The evaluator ensures that the rationale for each such error message is logical.

ADV_FSP.6.9C **The formal presentation of the functional specification of the TSF shall describe the TSFI using a formal style, supported by informal, explanatory text where appropriate**.

ADV_FSP.6-13 **The evaluator shall examine the TSFI description to determine that it contains all necessary explanatory text**.

Evaluator shall examine that explanatory text is sufficient to understand the formal model.

If assumptions are done or some TSFI are not represented in formal style, justifications have to be present in explanatory text. Evaluator shall examine the relevance of these justifications.

If a refinement process is conducted from ADV_SPM to ADV_FSP, the refinement scheme shall be presented and explained in explanatory text. Evaluator shall examine the relevance and the consistence of the refinement scheme.

ADV_FSP.6-14 **The evaluator shall examine the TSFI description to determine that the theorems proved in formal model are consistent with the security objectives of Security Policy Model.**

The consistency can be verified with a refinement proof or with manual verification on formal model. Evaluator shall examine the proof to ensure that there is no inconsistency.

Evaluator shall replay the proof.

ADV_FSP.6.10C The tracing shall demonstrate that the SFRs trace to TSFIs in the functional specification.

ADV_FSP.6-15 The evaluator shall check that the tracing links the SFRs to the corresponding TSFIs.

The tracing is provided by the developer to serve as a guide to understand which SFRs are related to which TSFIs. This tracing can be as simple as a table; it is used as input to the evaluator for use in the following work units, in which the evaluator verifies its completeness and accuracy.

4.1.2.5 Action ADV_FSP.6.2E

ADV_FSP.6-16 The evaluator shall examine the functional specification to determine that it is a complete instantiation of the SFRs.

To ensure that all SFRs are covered by the functional specification, as well as the test coverage analysis, the evaluator may build upon the developer's tracing (see ADV_FSP.6-15) a map between the TOE security functional requirements and the TSFI. Note that this map may have to be at a level of detail below the component or even element level of the requirements, because of operations (assignments, refinements, selections) performed on the functional requirement by the ST author.

For example, the FDP_ACC.1 component contains an element with assignments. If the ST contained, for instance, ten rules in the FDP_ACC.1 assignment, and these ten rules were covered by three different TSFI, it would be inadequate for the evaluator to map FDP_ACC.1 to TSFI A, B, and C and claim they had completed the work unit. Instead, the evaluator would map FDP_ACC.1 (rule 1) to TSFI A; FDP_ACC.1 (rule 2) to TSFI B; etc. It might also be the case that the interface is a wrapper interface (e.g., IOCTL), in which case the mapping would need to be specific to certain set of parameters for a given interface.

The evaluator must recognise that for requirements that have little or no manifestation at the TSF boundary (e.g., FDP_RIP) it is not expected that they completely map those requirements to the TSFI. The analysis for those requirements will be performed in the analysis for the TOE design (ADV_TDS) when included in the ST. It is also important to note that since the parameters, actions, and error messages associated with TSFIs must be fully specified, the evaluator should be able to determine if all aspects of an SFR appear to be implemented at the interface level.

As the functional specification shall be presented in a formal style, the verification shall be done on formal model. Evaluator shall examine that the model is sufficient to cover the SFRs with the help of explanatory text. A justification of SFR that are no not covered by the formal model shall be provided by the developer. Evaluator shall examine the relevance of these justifications.

ADV_FSP.6-17 The evaluator shall examine the functional specification to determine that it is an accurate instantiation of the SFRs.

For each functional requirement in the ST that results in effects visible at the TSF boundary, the information in the associated TSFI for that requirement specifies the required functionality described by the requirement. For example, if the ST contains a requirement for access control lists, and the only TSFI that map to that requirement specify functionality for Unix-style protection bits, then the functional specification is not accurate with respect to the requirements.

The evaluator must recognise that for requirements that have little or no manifestation at the TSF boundary (e.g., FDP_RIP) it is not expected that the evaluator completely map those requirements to the TSFI. The analysis for those requirements will be performed in the analysis for the TOE design (ADV_TDS) when included in the ST.

As the functional specification shall be presented in a formal style, the verification shall be done on formal model. Evaluator shall examine that the model is consistent with the SFRs with the help of explanatory text. A justification of SFR that are no not covered by the formal model shall be provided by the developer. Evaluator shall examine the relevance of these justifications.

4.1.3 Evaluation of sub-activity (ADV_TDS.5)

4.1.3.1 Objectives

The objective of this sub-activity is to determine whether the TOE design provides a description of the TOE in terms of subsystems sufficient to determine the TSF boundary, and provides a description of the TSF internals in terms of modules (and optionally higher-level abstractions). It provides **semiformal description of the modules** for the evaluator to determine that the SFRs are completely and accurately implemented; as such, the TOE design provides an explanation of the implementation representation.

The related part of the AIS34 from the German Scheme and the Note12 from the French Scheme were used as the basis here.

4.1.3.2 Input

The evaluation evidence for this sub-activity is:

a) the ST;

b) the functional specification;

c) security architecture description;

d) the TOE design.

4.1.3.3 Application notes

There are three types of activity that the evaluator must undertake with respect to the TOE design. First, the evaluator determines that the TSF boundary has been adequately described. Second, the evaluator determines that the developer has provided documentation that conforms to the content and presentation requirements this subsystem, and that is consistent with other documentation provided for the TOE. Finally, the evaluator must analyse the design information provided for the **modules** to understand how the system is implemented, and with that knowledge ensure that the TSFI in the functional specification are adequately described, and that the test information adequately tests the TSF (done in the Class ATE: Tests work units).

4.1.3.4 Action ADV_TDS.5.1E

ADV_TDS.5.1C The design shall describe the structure of the TOE in terms of subsystems.

ADV_TDS.5-1 The evaluator shall examine the TOE design to determine that the structure of the entire TOE is described in terms of subsystems.

The evaluator ensures that all of the subsystems of the TOE are identified. This description of the TOE will be used as input to work unit **ADV_TDS.5-4**, where the parts of the TOE that make up the TSF are identified. That is, this requirement is on the entire TOE rather than on only the TSF.

The TOE (and TSF) may be described in multiple layers of abstraction (i.e. subsystems and modules) Depending upon the complexity of the TOE, its design may be described in terms of subsystems and modules, as described in CC Part 3 Annex A.4, ADV_TDS: Subsystems and Modules. For a very simple TOE that can be described solely at the “module” level (see **ADV_TDS.5-2**), this work unit is not applicable and therefore considered to be satisfied.

In performing this activity, the evaluator examines other evidence presented for the TOE (e.g., ST, operator user guidance) to determine that the description of the TOE in such evidence is consistent with the description contained in the TOE design.

ADV_TDS.5.2C The design shall describe the TSF in terms of modules, designating each module as SFR-enforcing, SFR-supporting, or SFR-non-interfering.

ADV_TDS.5-2 The evaluator shall examine the TOE design to determine that the entire TSF is described in terms of modules.

The evaluator will examine the modules for specific properties in other work units; in this work unit the evaluator determines that the modular description covers the entire TSF, and not just a portion of the TSF. The evaluator uses other evidence provided for the evaluation (e.g., functional specification, architectural description) in making this determination. For example, if the functional specification contains interfaces to functionality that does not appear to be described in the TOE design description, it may be the case that a portion of the TSF has not been included appropriately. Making this determination will likely be an iterative process, whereas more analysis is done on the other evidence, more confidence can be gained with respect to the completeness of the documentation.

Unlike subsystems, modules describe the implementation in a level of detail that can serve as a guide to reviewing the implementation representation. A description of a module should be such that one could create an implementation of the module from the description, and the resulting implementation would be 1) identical to the actual TSF implementation in terms of the interfaces presented, 2) identical in the use of interfaces that are mentioned in the design, and 3) functionally equivalent to the

description of the purpose of the TSF module. For instance, RFC 793 provides a high-level description of the TCP protocol. It is necessarily implementation independent. While it provides a wealth of detail, it is not a suitable design description because it is not specific to an implementation. An actual implementation can add to the protocol specified in the RFC, and implementation choices (for instance, the use of global data vs. local data in various parts of the implementation) may have an impact on the analysis that is performed. The design description of the TCP module would list the interfaces presented by the implementation (rather than just those defined in RFC 793), as well as an algorithm description of the processing associated with the modules implementing TCP (assuming it was part of the TSF).

ADV_TDS.5-3 The evaluator shall check the TOE design to determine that the TSF modules are identified as either SFR-enforcing, SFR-supporting, or SFR-non-interfering.

The purpose of designating each module (according to the role a particular module plays in the enforcement of the SFRs) is to allow developers to provide less information about the parts of the TSF that have little role in security. It is always permissible for the developer to provide more information or detail than the requirements demand, as might occur when the information has been gathered outside the evaluation context. In such cases the developer must still designate the modules as either SFR-enforcing, SFR-supporting, or SFR-non-interfering.

The accuracy of these designations is continuously reviewed as the evaluation progresses. The concern is the mis-designation of modules as being less important (and hence, having less information) than is really the case. While blatant mis-designations may be immediately apparent (e.g., designating an authentication module as anything but SFR-enforcing when User identification (FIA_UID) is one of the SFRs being claimed), other mis-designations might not be discovered until the TSF is better understood. The evaluator must therefore keep in mind that these designations are the developer's initial best effort, but are subject to change. Further guidance is provided under work unit **ADV_TDS.5-15**, which examines the accuracy of these designations.

ADV_TDS.5.3C The design shall identify all subsystems of the TSF.

ADV_TDS.5-4 The evaluator shall examine the TOE design to determine that all subsystems of the TSF are identified.

If the design is presented solely in terms of modules, then subsystems in these requirements are equivalent to modules and the activity should be performed at the module level.

In work unit ADV_TDS.5-1 all of the subsystems of the TOE were identified, and a determination made that the non-TSF subsystems were correctly characterised. Building on that work, the subsystems that were not characterised as non-TSF subsystems should be precisely identified. The evaluator determines that, of the hardware and software installed and configured according to the Preparative procedures (AGD_PRE) guidance, each subsystem has been accounted for as either one that is part of the TSF, or one that is not.

ADV_TDS.5.4C The design shall provide a semiformal description of each subsystem of the TSF, supported by informal, explanatory text where appropriate.

ADV_TDS.5-5 The evaluator shall examine the TDS documentation to determine that the semiformal notation used for describing the subsystems, modules and their interfaces is defined or referenced.

A semiformal notation can be either defined by the sponsor or a corresponding standard be referenced. The evaluator should provide a mapping of security functions and their interfaces outlining in what part of the documentation a function or interface is semiformal described and what notation is used. The evaluator examines all semiformal notations used to make sure that they are of a semiformal style and to justify the appropriateness of the manner how the semiformal notations are used for the TOE.

The evaluator is reminded that a semi-formal presentation is characterised by a standardised format with a well-defined syntax that reduces ambiguity that may occur in informal presentations. The syntax of all semiformal notations used in the functional specification shall be defined or a corresponding standard be referenced. The evaluator verifies that the semiformal notations used for expressing the functional specification are capable of expressing features relevant to security. In order to determine this, the evaluator can refer to the SFR and compare the TSF security features stated in the ST and those described in the FSP using the semiformal notations.

ADV_TDS.5-6 The evaluator shall examine the TOE design to determine that each subsystem of the TSF describes its role in the enforcement of SFRs described in the ST.

If the design is presented solely in terms of modules, then this work unit will be considered satisfied by the assessment done in subsequent work units; no explicit action on the part of the evaluator is necessary in this case.

On systems that are complex enough to warrant a subsystem-level description of the TSF in addition to the modular description, the goal of the subsystem-level description is to give the evaluator context for the modular description that follows. Therefore, the evaluator ensures that the subsystem-level description contains a description of how the security functional requirements are achieved in the design, but at a level of abstraction above the modular description. This description should discuss the mechanisms used at a level that is aligned with the module description; this will provide the evaluators the road map needed to intelligently assess the information contained in the module description. A well-written set of subsystem descriptions will help guide the evaluator in determining the modules that are most important to examine, thus focusing the evaluation activity on the portions of the TSF that have the most relevance with respect to the enforcement of the SFRs.

The evaluator ensures that all subsystems of the TSF have a description. While the description should focus on the role that the subsystem plays in enforcing or supporting the implementation of the SFRs, enough information must be present so that a context for understanding the SFR-related functionality is provided.

ADV_TDS.5-7 The evaluator shall examine the TOE design to determine that each SFR-non-interfering subsystem of the TSF is described such that the evaluator can determine that the subsystem is SFR-non-interfering.

If the design is presented solely in terms of modules, then this work unit will be considered satisfied by the assessment done in subsequent work units; no explicit action on the part of the evaluator is necessary in this case.

An SFR-non-interfering subsystem is one on which the SFR-enforcing and SFR-supporting subsystems have no dependence; that is, they play no role in implementing SFR functionality.

The evaluator ensures that all subsystems of the TSF have a description. While the description should focus on the role that the subsystem do not plays in enforcing or supporting the implementation of the SFRs, enough information must be present so that a context for understanding the SFR-non-interfering functionality is provided.

ADV_TDS.5.5C The design shall provide a description of the interactions among all subsystems of the TSF.

ADV_TDS.5-8 The evaluator shall examine the TOE design to determine that interactions between the subsystems of the TSF are described.

If the design is presented solely in terms of modules, then this work unit will be considered satisfied by the assessment done in subsequent work units; no explicit action on the part of the evaluator is necessary in this case.

On systems that are complex enough to warrant a subsystem-level description of the TSF in addition to the modular description, the goal of describing the interactions between the subsystems is to help provide the reader a better understanding of how the TSF performs it functions. These interactions do not need to be characterised at the implementation level (e.g., parameters passed from one routine in a subsystem to a routine in a different subsystem; global variables; hardware signals (e.g., interrupts) from a hardware subsystem to an interrupt-handling subsystem), but the data elements identified for a particular subsystem that are going to be used by another subsystem need to be covered in this discussion. Any control relationships between subsystems (e.g., a subsystem responsible for configuring a rule base for a firewall system and the subsystem that actually implements these rules) should also be described.

It should be noted while the developer should characterise all interactions between subsystems, the evaluators need to use their own judgement in assessing the completeness of the description. If the reason for an interaction is unclear, or if there are SFR-related interactions (discovered, for instance, in examining the module-level documentation) that do not appear to be described, the evaluator ensures that this information is provided by the developer. However, if the evaluator can determine that interactions among a particular set of subsystems, while incompletely described by the developer, and a complete description will not aid in understanding the overall functionality nor security functionality provided by the TSF, then the evaluator may choose to consider the description sufficient, and not pursue completeness for its own sake.

ADV_TDS.5.6C The design shall provide a mapping from the subsystems of the TSF to the modules of the TSF.

ADV_TDS.5-9 The evaluator shall examine the TOE design to determine that the mapping between the subsystems of the TSF and the modules of the TSF is complete.

If the design is presented solely in terms of modules, then this work unit is considered satisfied.

For TOEs that are complex enough to warrant a subsystem-level description of the TSF in addition to the modular description, the developer provides a simple mapping showing how the modules of the TSF are allocated to the subsystems. This will provide the evaluator a guide in performing their module-level assessment. To determine completeness, the evaluator examines each mapping and determines that all subsystems map to at least one module, and that all modules map to exactly one subsystem.

ADV_TDS.5-10 The evaluator shall examine the TOE design to determine that the mapping between the subsystems of the TSF to the modules of the TSF is accurate.

If the design is presented solely in terms of modules, then this work unit is considered satisfied.

For TOEs that are complex enough to warrant a subsystem-level description of the TSF in addition to the modular description, the developer provides a simple mapping showing how the modules of the TSF are allocated to the subsystems. This will provide the evaluator a guide in performing their module-level assessment. The evaluator may choose to check the accuracy of the mapping in conjunction with performing other work units. An “inaccurate” mapping is one where the module is mistakenly associated with a subsystem where its functions are not used within the subsystem. Because the mapping is intended to be a guide supporting more detailed analysis, the evaluator is cautioned to apply appropriate effort to this work unit. Expending extensive evaluator resources verifying the accuracy of the mapping is not necessary. Inaccuracies that lead to mis-understandings related to the design that are uncovered as part of this or other work units are the ones that should be associated with this work unit and corrected.

ADV_TDS.5.7C The design shall **provide a semiformal description of** each module in terms of its **purpose, interaction,** interfaces, return values from those interfaces, and called interfaces to other modules, **supported by informal, explanatory text where appropriate**.

ADV_TDS.5-11 **The evaluator shall examine the TDS documentation to determine that the semiformal notation used for describing the modules and their interfaces is defined or referenced.**

A semiformal notation can be either defined by the sponsor or a corresponding standard be referenced. The evaluator should provide a mapping of security functions and their interfaces outlining in what part of the documentation a function or interface is semiformally described and what notation is used. The evaluator examines all semiformal notations used to make sure that they are in a semiformal style and to justify the appropriateness of the manner how the semiformal notations are used for the TOE.

The evaluator is reminded that a semi-formal presentation is characterised by a standardised format with a well-defined syntax that reduces ambiguity that may occur in informal presentations. The syntax of all semiformal notations used in the functional specification shall be defined or a corresponding standard be referenced. The evaluator verifies that the semiformal notations used for expressing the functional specification allow specifying security-relevant features. In order to determine this, the evaluator can refer to the SFRs and compare the TSF security features stated in the ST and those described in the FSP using the semiformal notations.

ADV_TDS.5-12 The evaluator shall examine the TOE design to determine that the description of the purpose of each module, and relationship with other modules is complete and accurate.

The purpose of a module provides a description indicating what function the module is fulfilling. A word of caution to evaluator is in order. The focus of this work unit should be to provide the evaluator an understanding of how the module works so that determinations can be made about the soundness of the implementation of the SFRs, as well as to support architectural analysis performed for ADV_ARC subsystems. As

long as the evaluator has a sound understanding of the module's operation, and its relationship to other modules and the TOE as a whole, the evaluator should consider the objective of the work achieved and not engage in a documentation exercise for the developer (by requiring, for example, a complete algorithmic description for a self-evident implementation representation).

Because the modules are at such a low level, it may be difficult determine completeness and accuracy impacts from other documentation, such as operational user guidance, the functional specification, the TSF internals, or the security architecture description. However, the evaluator uses the information present in those documents to the extent possible to help ensure that the purpose is accurately and completely described. This analysis can be aided by the analysis performed for the work units for the **ADV_TDS.5.8C** element, which maps the TSFI in the functional specification to the modules of the TSF.

ADV_TDS.5-13 The evaluator shall examine the TOE design to determine that the description of the interfaces presented by each module contain an accurate and complete description of the parameters, the invocation conventions for each interface, and any values returned directly by the interface.

The SFR-related interfaces of a module are those interfaces used by other modules as a means to invoke the SFR-related operations provided, and to provide inputs to or receive outputs from the module. The purpose in the specification of these interfaces is to permit the exercise of them during testing. Inter-module interfaces that are not SFR-related need not be specified or described, since they are not a factor in testing. Likewise, other internal interfaces that are not a factor in traversing SFR-related paths of execution (such as those internal paths that are fixed).

SFR-related interfaces are described in terms of how they are invoked, and any values that are returned. This description would include a list of parameters, and descriptions of these parameters. Note that global data would also be considered parameters if used by the module (either as inputs or outputs) when invoked. If a parameter were expected to take on a set of values (e.g., a “flag” parameter), the complete set of values the parameter could take on that would have an effect on module processing would be specified. Likewise, parameters representing data structures are described such that each field of the data structure is identified and described. Note that different programming languages may have additional “interfaces” that would be non-obvious; an example would be operator/function overloading in C++. This “implicit interface” in the class description would also be described as part of the low-level TOE design. Note that although a module could present only one interface, it is more common that a module presents a small set of related interfaces.

In terms of the assessment of parameters (inputs and outputs) to a module, any use of global data must also be considered. A module “uses” global data if it either reads or writes the data. In order to assure the description of such parameters (if used) is complete, the evaluator uses other information provided about the module in the TOE design (interfaces, algorithmic description, etc.), as well as the description of the particular set of global data assessed in work unit ADV_TDS.5-13. For instance, the evaluator could first determine the processing the module performs by examining its function and interfaces presented (particularly the parameters of the interfaces). They could then check to see if the processing appears to “touch” any of the global data areas identified in the TDS design. The evaluator then determines that, for each global data area that appears to be “touched”, that global data area is listed as a means of input or output by the module the evaluator is examining.

Invocation conventions are a programming-reference-type description that one could use to correctly invoke a module's interface if one were writing a program to make use of the module's functionality through that interface. This includes necessary inputs and

outputs, including any set-up that may need to be performed with respect to global variables.

Values returned through the interface refer to values that are either passed through parameters or messages; values that the function call itself returns in the style of a “C” program function call; or values passed through global means (such as certain error routines in *ix-style operating systems).

In order to assure the description is complete, the evaluator uses other information provided about the module in the TOE design (e.g., algorithmic description, global data used) to ensure that it appears all data necessary for performing the functions of the module is presented to the module, and that any values that other modules expect the module under examination to provide are identified as being returned by the module. The evaluator determines accuracy by ensuring that the description of the processing matches the information listed as being passed to or from an interface.

ADV_TDS.5.8C The mapping shall demonstrate that all TSFIs trace to the behaviour described in the TOE design that they invoke.

ADV_TDS.5-14 The evaluator shall examine the TOE design to determine that it contains a complete and accurate mapping from the TSFI described in the functional specification to the modules of the TSF described in the TOE design.

The modules described in the TOE design provide a description of the implementation of the TSF. The TSFI provide a description of how the implementation is exercised. The evidence from the developer identifies the module that is initially invoked when an operation is requested at the TSFI, and identifies the chain of modules invoked up to the module that is primarily responsible for implementing the functionality. However, a complete call tree for each TSFI is not required for this work unit. The cases in which more than one module would have to be identified are where there are “entry point” modules or wrapper modules that have no functionality other than conditioning inputs or de-multiplexing an input. Mapping to one of these modules would not provide any useful information to the evaluator.

The evaluator assesses the completeness of the mapping by ensuring that all of the TSFI map to at least one module. The verification of accuracy is more complex.

The first aspect of accuracy is that each TSFI is mapped to a module at the TSF boundary. This determination can be made by reviewing the module description and its interfaces/interactions. The next aspect of accuracy is that each TSFI identifies a chain of modules between the initial module identified and a module that is primarily responsible for implementing the function presented at the TSF. Note that this may be the initial module, or there may be several modules, depending on how much pre-conditioning of the inputs is done. It should be noted that one indicator of a pre-conditioning module is that it is invoked for a large number of the TSFI, where the TSFI are all of similar type (e.g., system call). The final aspect of accuracy is that the mapping makes sense. For instance, mapping a TSFI dealing with access control to a module that checks passwords is not accurate. The evaluator should again use judgement in making this determination. The goal is that this information aids the evaluator in understanding the system and implementation of the SFRs, and ways in which entities at the TSF boundary can interact with the TSF. The bulk of the assessment of whether the SFRs are described accurately by the modules is performed in other work units.

4.1.3.5 Action ADV_TDS.5.2E

ADV_TDS.5-15 The evaluator shall examine the TOE security functional requirements and the TOE design, to determine that all ST security functional requirements are covered by the TOE design.

The evaluator may construct a map between the TOE security functional requirements and the TOE design. This map will likely be from a functional requirement to a set of subsystems, and later to modules. Note that this map may have to be at a level of detail below the component or even element level of the requirements, because of operations (assignments, refinements, selections) performed on the functional requirement by the ST author.

For example, the FDP_ACC.1 Subset access control component contains an element with assignments. If the ST contained, for instance, ten rules in the FDP_ACC.1 Subset access control assignment, and these ten rules were implemented in specific places within fifteen modules, it would be inadequate for the evaluator to map FDP_ACC.1 Subset access control to one subsystem and claim the work unit had been completed. Instead, the evaluator would map FDP_ACC.1 Subset access control (rule 1) to modules x, y and z of subsystem A; FDP_ACC.1 Subset access control (rule 2) to x, p, and q of subsystem A; etc.

ADV_TDS.5-16 The evaluator shall examine the TOE design to determine that it is an accurate instantiation of all security functional requirements.

The evaluator may construct a map between the TOE security functional requirements and the TOE design. This map will likely be from a functional requirement to a set of subsystems. Note that this map may have to be at a level of detail below the component or even element level of the requirements, because of operations (assignments, refinements, selections) performed on the functional requirement by the ST author.

As an example, if the ST requirements specified a role-based access control mechanism, the evaluator would first identify the subsystems, and modules that contribute to this mechanism's implementation. This could be done by in-depth knowledge or understanding of the TOE design or by work done in the previous work unit. Note that this trace is only to identify the subsystems, and modules, and is not the complete analysis.

The next step would be to understand what mechanism the subsystems and modules implemented. For instance, if the design described an implementation of access control based on UNIX-style protection bits, the design would not be an accurate instantiation of those access control requirements present in the ST example used above. If the evaluator could not determine that the mechanism was accurately implemented because of a lack of detail, the evaluator would have to assess whether all of the SFR-enforcing subsystems and modules have been identified, or if adequate detail had been provided for those subsystems and modules.

4.1.4 Evaluation of sub-activity (ADV_TDS.6)

4.1.4.1 Objectives

The objective of this sub-activity is to determine whether the TOE design provides a description of the TOE in terms of subsystems sufficient to determine the TSF boundary, and provides a description of the TSF internals in terms of modules (and optionally higher-level abstractions). It provides formal description of the subsystems and semiformal description of the modules for the evaluator to determine that the SFRs are completely and accurately implemented; as such, the TOE design provides an explanation of the implementation representation.

4.1.4.2 Input

The evaluation evidence for this sub-activity is:

a) the ST;

b) the functional specification;

c) security architecture description;

d) the TOE design:

➔ Including formal model source code of the TSF subsystems SRC

➔ Proof of this model PROOF

➔ Rationale on level of confidence for the chosen method ARG_TOOL;

➔ Explanatory text of the formal presentation of the TSF and the relationship between the various notions handled in this design specification of the TSF subsystems ARG_TDS.

➔ Presentation and justification of the “assumptions” (used in the evidence but not themselves proved) that may be introduced in the formal model ARG_PROOF;

4.1.4.3 Application notes

There are three types of activity that the evaluator must undertake with respect to the TOE design. First, the evaluator determines that the TSF boundary has been adequately described. Second, the evaluator determines that the developer has provided documentation that conforms to the content and presentation requirements this subsystem, and that is consistent with other documentation provided for the TOE. Finally, the evaluator must analyse the design information provided for the **modules** to understand how the system is implemented, and with that knowledge ensure that the TSFI in the functional specification are adequately described, and that the test information adequately tests the TSF (done in the Class ATE: Tests work units).

4.1.4.4 Action ADV_TDS.6.1E

ADV_TDS.6.1C The design shall describe the structure of the TOE in terms of subsystems.

ADV_TDS.6-1 The evaluator shall examine the TOE design to determine that the structure of the entire TOE is described in terms of subsystems.

The evaluator ensures that all of the subsystems of the TOE are identified. This description of the TOE will be used as input to work unit **ADV_TDS.6-4**, where the parts of the TOE that make up the TSF are identified. That is, this requirement is on the entire TOE rather than on only the TSF.

The TOE (and TSF) may be described in multiple layers of abstraction (i.e. subsystems and modules). Depending upon the complexity of the TOE, its design may be described in terms of subsystems and modules, as described in CC Part 3 Annex A.4, ADV_TDS: Subsystems and Modules. For a very simple TOE that can be described solely at the “module” level (see **ADV_TDS.6-2**), this work unit is not applicable and therefore considered to be satisfied.

In performing this activity, the evaluator examines other evidence presented for the TOE (e.g., ST, operator user guidance) to determine that the description of the TOE in such evidence is consistent with the description contained in the TOE design.

As subsystems shall be described using a formal style, the verification shall be done on formal model.

ADV_TDS.6.2C The design shall describe the TSF in terms of modules, designating each module as SFR-enforcing, SFR-supporting, or SFR-non-interfering.

ADV_TDS.6-2 The evaluator shall examine the TOE design to determine that the entire TSF is described in terms of modules.

The evaluator will examine the modules for specific properties in other work units; in this work unit the evaluator determines that the modular description covers the entire TSF, and not just a portion of the TSF. The evaluator uses other evidence provided for the evaluation (e.g., functional specification, architectural description) in making this determination. For example, if the functional specification contains interfaces to functionality that does not appear to be described in the TOE design description, it may be the case that a portion of the TSF has not been included appropriately. Making this determination will likely be an iterative process, where as more analysis is done on the other evidence, more confidence can be gained with respect to the completeness of the documentation.

Unlike subsystems, modules describe the implementation in a level of detail that can serve as a guide to reviewing the implementation representation. A description of a module should be such that one could create an implementation of the module from the description, and the resulting implementation would be 1) identical to the actual TSF implementation in terms of the interfaces presented, 2) identical in the use of interfaces that are mentioned in the design, and 3) functionally equivalent to the description of the purpose of the TSF module. For instance, RFC 793 provides a high-level description of the TCP protocol. It is necessarily implementation independent. While it provides a wealth of detail, it is not a suitable design description because it is not specific to an implementation. An actual implementation can add to the protocol specified in the RFC, and implementation choices (for instance, the use of global data vs. local data in various parts of the implementation) may have an impact on the analysis that is performed. The design description of the TCP module would list the interfaces presented by the implementation (rather than just those defined in RFC 793), as well as an algorithm description of the processing associated with the modules implementing TCP (assuming it was part of the TSF).

ADV_TDS.6-3 The evaluator shall check the TOE design to determine that the TSF modules are identified as either SFR-enforcing, SFR-supporting, or SFR-non-interfering.

The purpose of designating each module (according to the role a particular module plays in the enforcement of the SFRs) is to allow developers to provide less information about the parts of the TSF that have little role in security. It is always permissible for the developer to provide more information or detail than the requirements demand, as might occur when the information has been gathered outside the evaluation context. In such cases the developer must still designate the modules as either SFR-enforcing, SFR-supporting, or SFR-non-interfering.

The accuracy of these designations is continuously reviewed as the evaluation progresses. The concern is the mis-designation of modules as being less important (and hence, having less information) than is really the case. While blatant mis-designations may be immediately apparent (e.g., designating an authentication module as anything but SFR-enforcing when User identification (FIA_UID) is one of the SFRs being claimed), other mis-designations might not be discovered until the TSF is better understood. The evaluator must therefore keep in mind that these designations are the developer's initial best effort, but are subject to change. Further guidance is provided under work unit **ADV_TDS.6-19**, which examines the accuracy of these designations.

ADV_TDS.6.3C The design shall identify all subsystems of the TSF.

ADV_TDS.6-4 The evaluator shall examine the TOE design to determine that all subsystems of the TSF are identified.

If the design is presented solely in terms of modules, then subsystems in these requirements are equivalent to modules and the activity should be performed at the module level.

In work unit ADV_TDS.6-1 all of the subsystems of the TOE were identified, and a determination made that the non-TSF subsystems were correctly characterised. Building on that work, the subsystems that were not characterised as non-TSF subsystems should be precisely identified. The evaluator determines that, of the hardware and software installed and configured according to the Preparative procedures (AGD_PRE) guidance, each subsystem has been accounted for as either one that is part of the TSF, or one that is not.

As subsystems shall be described using a formal style, the verification shall be done on formal model.

ADV_TDS.6.4C The design shall provide a semiformal description of each subsystem of the TSF, supported by informal, explanatory text where appropriate.

ADV_TDS.6-5 The evaluator shall examine the TDS documentation to determine that the semiformal notation used for describing the subsystems, modules and their interfaces is defined or referenced.

A semiformal notation can be either defined by the sponsor or a corresponding standard be referenced. The evaluator should provide a mapping of security functions and their interfaces outlining in what part of the documentation a function or interface is semiformal described and what notation is used. The evaluator examines all semiformal notations used to make sure that they are of a semiformal style and to justify the appropriateness of the manner how the semiformal notations are used for the TOE.

The evaluator is reminded that a semi-formal presentation is characterised by a standardised format with a well-defined syntax that reduces ambiguity that may occur in informal presentations. The syntax of all semiformal notations used in the functional specification shall be defined or a corresponding standard be referenced. The evaluator verifies that the semiformal notations used for expressing the functional specification are capable of expressing features relevant to security. In order to determine this, the evaluator can refer to the SFR and compare the TSF security features stated in the ST and those described in the FSP using the semiformal notations.

ADV_TDS.6-6 The evaluator shall examine the TOE design to determine that each subsystem of the TSF describes its role in the enforcement of SFRs described in the ST.

If the design is presented solely in terms of modules, then this work unit will be considered satisfied by the assessment done in subsequent work units; no explicit action on the part of the evaluator is necessary in this case.

On systems that are complex enough to warrant a subsystem-level description of the TSF in addition to the modular description, the goal of the subsystem-level description is to give the evaluator context for the modular description that follows. Therefore, the evaluator ensures that the subsystem-level description contains a description of how the security functional requirements are achieved in the design, but at a level of abstraction above the modular description. This description should discuss the mechanisms used at a level that is aligned with the module description; this will provide the evaluators the road map needed to intelligently assess the information contained in the module description. A well-written set of subsystem descriptions will help guide the evaluator in determining the modules that are most important to

examine, thus focusing the evaluation activity on the portions of the TSF that have the most relevance with respect to the enforcement of the SFRs.

The evaluator ensures that all subsystems of the TSF have a description. While the description should focus on the role that the subsystem plays in enforcing or supporting the implementation of the SFRs, enough information must be present so that a context for understanding the SFR-related functionality is provided.

ADV_TDS.5-7 The evaluator shall examine the TOE design to determine that each SFR-non-interfering subsystem of the TSF is described such that the evaluator can determine that the subsystem is SFR-non-interfering.

If the design is presented solely in terms of modules, then this work unit will be considered satisfied by the assessment done in subsequent work units; no explicit action on the part of the evaluator is necessary in this case.

An SFR-non-interfering subsystem is one on which the SFR-enforcing and SFR-supporting subsystems have no dependence; that is, they play no role in implementing SFR functionality.

The evaluator ensures that all subsystems of the TSF have a description. While the description should focus on the role that the subsystem do not plays in enforcing or supporting the implementation of the SFRs, enough information must be present so that a context for understanding the SFR-non-interfering functionality is provided.

ADV_TDS.6.5C The design shall provide a description of the interactions among all subsystems of the TSF.

ADV_TDS.6-8 The evaluator shall examine the TOE design to determine that interactions between the subsystems of the TSF are described.

If the design is presented solely in terms of modules, then this work unit will be considered satisfied by the assessment done in subsequent work units; no explicit action on the part of the evaluator is necessary in this case.

On systems that are complex enough to warrant a subsystem-level description of the TSF in addition to the modular description, the goal of describing the interactions between the subsystems is to help provide the reader a better understanding of how the TSF performs it functions. These interactions do not need to be characterised at the implementation level (e.g., parameters passed from one routine in a subsystem to a routine in a different subsystem; global variables; hardware signals (e.g., interrupts) from a hardware subsystem to an interrupt-handling subsystem), but the data elements identified for a particular subsystem that are going to be used by another subsystem need to be covered in this discussion. Any control relationships between subsystems (e.g., a subsystem responsible for configuring a rule base for a firewall system and the subsystem that actually implements these rules) should also be described.

It should be noted while the developer should characterise all interactions between subsystems, the evaluators need to use their own judgement in assessing the completeness of the description. If the reason for an interaction is unclear, or if there are SFR-related interactions (discovered, for instance, in examining the module-level documentation) that do not appear to be described, the evaluator ensures that this information is provided by the developer. However, if the evaluator can determine that interactions among a particular set of subsystems, while incompletely described by the developer, and a complete description will not aid in understanding the overall functionality nor security functionality provided by the TSF, then the evaluator may choose to consider the description sufficient, and not pursue completeness for its own sake.

ADV_TDS.6.6C The design shall provide a mapping from the subsystems of the TSF to the modules of the TSF.

ADV_TDS.6-9 The evaluator shall examine the TOE design to determine that the mapping between the subsystems of the TSF and the modules of the TSF is complete.

If the design is presented solely in terms of modules, then this work unit is considered satisfied.

For TOEs that are complex enough to warrant a subsystem-level description of the TSF in addition to the modular description, the developer provides a simple mapping showing how the modules of the TSF are allocated to the subsystems. This will provide the evaluator a guide in performing their module-level assessment. To determine completeness, the evaluator examines each mapping and determines that all subsystems map to at least one module, and that all modules map to exactly one subsystem.

ADV_TDS.6-10 The evaluator shall examine the TOE design to determine that the mapping between the subsystems of the TSF to the modules of the TSF is accurate.

If the design is presented solely in terms of modules, then this work unit is considered satisfied.

For TOEs that are complex enough to warrant a subsystem-level description of the TSF in addition to the modular description, the developer provides a simple mapping showing how the modules of the TSF are allocated to the subsystems. This will provide the evaluator a guide in performing their module-level assessment. The evaluator may choose to check the accuracy of the mapping in conjunction with performing other work units. An “inaccurate” mapping is one where the module is mistakenly associated with a subsystem where its functions are not used within the subsystem. Because the mapping is intended to be a guide supporting more detailed analysis, the evaluator is cautioned to apply appropriate effort to this work unit. Expending extensive evaluator resources verifying the accuracy of the mapping is not necessary. Inaccuracies that lead to mis-understandings related to the design that are uncovered as part of this or other work units are the ones that should be associated with this work unit and corrected.

ADV_TDS.6.7C The design shall describe each module in semiformal style in terms of its purpose, interaction, interfaces, return values from those interfaces, and called interfaces to other modules, supported by informal, explanatory text where appropriate.

ADV_TDS.6-11 The evaluator shall examine the TDS documentation to determine that the semiformal notation used for describing the modules and their interfaces is defined or referenced.

A semiformal notation can be either defined by the sponsor or a corresponding standard be referenced. The evaluator should provide a mapping of security functions and their interfaces outlining in what part of the documentation a function or interface is semiformal described and what notation is used. The evaluator examines all semiformal notations used to make sure that they are of a semiformal style and to justify the appropriateness of the manner how the semiformal notations are used for the TOE.

The evaluator is reminded that a semi-formal presentation is characterised by a standardised format with a well-defined syntax that reduces ambiguity that may occur in informal presentations. The syntax of all semiformal notations used in the functional specification shall be defined or a corresponding standard be referenced. The

evaluator verifies that the semiformal notations used for expressing the functional specification are capable of expressing features relevant to security. In order to determine this, the evaluator can refer to the SFR and compare the TSF security features stated in the ST and those described in the FSP using the semiformal notations.

ADV_TDS.6-12 The evaluator shall examine the TOE design to determine that the description of the purpose of each module, and relationship with other modules is complete and accurate.

The purpose of a module provides a description indicating what function the module is fulfilling. A word of caution to evaluator is in order. The focus of this work unit should be to provide the evaluator an understanding of how the module works so that determinations can be made about the soundness of the implementation of the SFRs, as well as to support architectural analysis performed for ADV_ARC subsystems. As long as the evaluator has a sound understanding of the module's operation, and its relationship to other modules and the TOE as a whole, the evaluator should consider the objective of the work achieved and not engage in a documentation exercise for the developer (by requiring, for example, a complete algorithmic description for a self-evident implementation representation).

Because the modules are at such a low level, it may be difficult determine completeness and accuracy impacts from other documentation, such as operational user guidance, the functional specification, the TSF internals, or the security architecture description. However, the evaluator uses the information present in those documents to the extent possible to help ensure that the purpose is accurately and completely described. This analysis can be aided by the analysis performed for the work units for the **ADV_TDS.6.9C** element, which maps the TSFI in the functional specification to the modules of the TSF.

ADV_TDS.6-13 The evaluator shall examine the TOE design to determine that the description of the interfaces presented by each module contain an accurate and complete description of the parameters, the invocation conventions for each interface, and any values returned directly by the interface.

The SFR-related interfaces of a module are those interfaces used by other modules as a means to invoke the SFR-related operations provided, and to provide inputs to or receive outputs from the module. The purpose in the specification of these interfaces is to permit the exercise of them during testing. Inter-module interfaces that are not SFR-related need not be specified or described, since they are not a factor in testing. Likewise, other internal interfaces that are not a factor in traversing SFR-related paths of execution (such as those internal paths that are fixed).

SFR-related interfaces are described in terms of how they are invoked, and any values that are returned. This description would include a list of parameters, and descriptions of these parameters. Note that global data would also be considered parameters if used by the module (either as inputs or outputs) when invoked. If a parameter were expected to take on a set of values (e.g., a “flag” parameter), the complete set of values the parameter could take on that would have an effect on module processing would be specified. Likewise, parameters representing data structures are described such that each field of the data structure is identified and described. Note that different programming languages may have additional “interfaces” that would be non-obvious; an example would be operator/function overloading in C++. This “implicit interface” in the class description would also be described as part of the low-level TOE design. Note that although a module could present only one interface, it is more common that a module presents a small set of related interfaces.

In terms of the assessment of parameters (inputs and outputs) to a module, any use of global data must also be considered. A module “uses” global data if it either reads or writes the data. In order to assure the description of such parameters (if used) is complete, the evaluator uses other information provided about the module in the TOE design (interfaces, algorithmic description, etc.), as well as the description of the particular set of global data assessed in work unit ADV_TDS.5-13. For instance, the evaluator could first determine the processing the module performs by examining its function and interfaces presented (particularly the parameters of the interfaces). They could then check to see if the processing appears to “touch” any of the global data areas identified in the TDS design. The evaluator then determines that, for each global data area that appears to be “touched”, that global data area is listed as a means of input or output by the module the evaluator is examining.

Invocation conventions are a programming-reference-type description that one could use to correctly invoke a module's interface if one were writing a program to make use of the module's functionality through that interface. This includes necessary inputs and outputs, including any set-up that may need to be performed with respect to global variables.

Values returned through the interface refer to values that are either passed through parameters or messages; values that the function call itself returns in the style of a “C” program function call; or values passed through global means (such as certain error routines in *ix-style operating systems).

In order to assure the description is complete, the evaluator uses other information provided about the module in the TOE design (e.g., algorithmic description, global data used) to ensure that it appears all data necessary for performing the functions of the module is presented to the module, and that any values that other modules expect the module under examination to provide are identified as being returned by the module. The evaluator determines accuracy by ensuring that the description of the processing matches the information listed as being passed to or from an interface.

ADV_TDS.6.8C **The formal specification of the TSF subsystems shall describe the TSF using a formal style, supported by informal, explanatory text where appropriate**.

ADV_TDS.6-14 **The evaluator shall examine the TDS documentation to determine that the TSF subsystems are presented using a formal style**.

The evaluator shall identify the formal methods and tools used by the developer in order to ensure that the used methods rely on correct theoretical principles. For this, the evaluator may use the document [ARG_TOOL] provided by the developer. Evaluator shall identify the common mistakes of this method.

ADV_TDS.6-15 **The evaluator shall examine the TDS documentation to determine that it contains all necessary explanatory text.**

Evaluator shall examine that explanatory text in ARG_TDS is sufficient to understand the formal model.

If assumptions are done or some TSF Subsystems are not represented in formal style, justifications have to be present in explanatory text. Evaluator shall examine the relevance of these justifications.

ADV_TDS.6-16 **The evaluator shall examine the TDS documentation to determine that it formally proves the correct description of the TSF subsystems.**

Evaluator shall examine ARG_PROOF to determine that the information provided is relevant, complete and consistent.

Evaluator shall replay the proof with the tools provided by the developer to determine there is no inconsistency.

ADV_TDS.6.9C The mapping shall demonstrate that all TSFIs trace to the behaviour described in the TOE design that they invoke.

ADV_TDS.6-17 The evaluator shall examine the TOE design to determine that it contains a complete and accurate mapping from the TSFI described in the functional specification to the modules of the TSF described in the TOE design.

The modules described in the TOE design provide a description of the implementation of the TSF. The TSFI provide a description of how the implementation is exercised. The evidence from the developer identifies the module that is initially invoked when an operation is requested at the TSFI, and identifies the chain of modules invoked up to the module that is primarily responsible for implementing the functionality. However, a complete call tree for each TSFI is not required for this work unit. The cases in which more than one module would have to be identified are where there are “entry point” modules or wrapper modules that have no functionality other than conditioning inputs or de-multiplexing an input. Mapping to one of these modules would not provide any useful information to the evaluator.

The evaluator assesses the completeness of the mapping by ensuring that all of the TSFI map to at least one module. The verification of accuracy is more complex.

The first aspect of accuracy is that each TSFI is mapped to a module at the TSF boundary. This determination can be made by reviewing the module description and its interfaces/interactions. The next aspect of accuracy is that each TSFI identifies a chain of modules between the initial module identified and a module that is primarily responsible for implementing the function presented at the TSF. Note that this may be the initial module, or there may be several modules, depending on how much pre-conditioning of the inputs is done. It should be noted that one indicator of a pre-conditioning module is that it is invoked for a large number of the TSFI, where the TSFI are all of similar type (e.g., system call). The final aspect of accuracy is that the mapping makes sense. For instance, mapping a TSFI dealing with access control to a module that checks passwords is not accurate. The evaluator should again use judgement in making this determination. The goal is that this information aids the evaluator in understanding the system and implementation of the SFRs, and ways in which entities at the TSF boundary can interact with the TSF. The bulk of the assessment of whether the SFRs are described accurately by the modules is performed in other work units.

ADV_TDS.6.10C **The proof of correspondence between the formal specifications of the TSF subsystems and of the functional specification shall demonstrate that all behaviour described in the TOE design is a correct and complete refinement of the TSFI that invoked it.**

ADV_TDS.6-18 **The evaluator shall examine the proof of correspondence between the formal specifications of the TSF subsystems and of the functional specification to determine that the description of the TSF subsystems is consistent with the description in the functional specification.**

If the examination of the proof of correspondence reveals an inconsistency between TOE design and functional specification representation, then this evaluator action fails.

ADV_TDS.6-19 **The evaluator shall examine the proof of correspondence between the formal specifications of the TSF subsystems and of the functional specification to determine that it is complete.**

4.1.4.5 Action ADV_TDS.6.2E

ADV_TDS.6-19 The evaluator shall examine the TOE security functional requirements and the TOE design, to determine that all ST security functional requirements are covered by the TOE design.

The evaluator may construct a map between the TOE security functional requirements and the TOE design. This map will likely be from a functional requirement to a set of subsystems, and later to modules. Note that this map may have to be at a level of detail below the component or even element level of the requirements, because of operations (assignments, refinements, selections) performed on the functional requirement by the ST author.

For example, the FDP_ACC.1 Subset access control component contains an element with assignments. If the ST contained, for instance, ten rules in the FDP_ACC.1 Subset access control assignment, and these ten rules were implemented in specific places within fifteen modules, it would be inadequate for the evaluator to map FDP_ACC.1 Subset access control to one subsystem and claim the work unit had been completed. Instead, the evaluator would map FDP_ACC.1 Subset access control (rule 1) to modules x, y and z of subsystem A; FDP_ACC.1 Subset access control (rule 2) to x, p, and q of subsystem A; etc.

ADV_TDS.6-20 The evaluator shall examine the TOE design to determine that it is an accurate instantiation of all security functional requirements.

The evaluator may construct a map between the TOE security functional requirements and the TOE design. This map will likely be from a functional requirement to a set of subsystems. Note that this map may have to be at a level of detail below the component or even element level of the requirements, because of operations (assignments, refinements, selections) performed on the functional requirement by the ST author.

As an example, if the ST requirements specified a role-based access control mechanism, the evaluator would first identify the subsystems, and modules that contribute to this mechanism's implementation. This could be done by in-depth knowledge or understanding of the TOE design or by work done in the previous work unit. Note that this trace is only to identify the subsystems, and modules, and is not the complete analysis.

The next step would be to understand what mechanism the subsystems and modules implemented. For instance, if the design described an implementation of access control based on UNIX-style protection bits, the design would not be an accurate instantiation of those access control requirements present in the ST example used above. If the evaluator could not determine that the mechanism was accurately implemented because of a lack of detail, the evaluator would have to assess whether all of the SFR-enforcing subsystems and modules have been identified, or if adequate detail had been provided for those subsystems and modules.

4.1.5 Evaluation of sub-activity (ADV_IMP.2)

4.1.5.1 Objectives

The objective of this sub-activity is to determine that the implementation representation made available by the developer is suitable for use in other analysis activities; suitability is judged by its conformance to the requirements for this component.

The related part of the AIS34 from the German Scheme and the Note12 from the French Scheme were used as the basis here.

4.1.5.2 Input

The evaluation evidence for this sub-activity is:

a) the implementation representation;

b) the documentation of the development tools, as resulting from ALC_TAT;

c) TOE design description.

4.1.5.3 Application notes

The entire implementation representation is made available to ensure that analysis activities are not curtailed due to lack of information. This does not imply that all of the representation is examined when the analysis activities are being performed. This is likely impractical in almost all cases. **However, to get high-level assurance in the correctness of the representation, the portion of the implementation representation to be examined for this sub-activity shall include at the minimum all the related parts that map the TSFI of the functional specification**.

4.1.5.4 Action ADV_IMP.2.1E

ADV_IMP.2.1C The implementation representation shall define the TSF to a level of detail such that the TSF can be generated without further design decisions.

ADV_IMP.2-1 The evaluator shall check that the implementation representation defines the TSF to a level of detail such that the TSF can be generated without further design decisions.

Source code or hardware diagrams and/or IC hardware design language code or layout data that are used to build the actual hardware are examples of parts of an implementation representation. The evaluator **examines** the implementation representation to **make sure** that it is at the appropriate level and not, for instance, a pseudo-code level which requires additional design decisions to be made. **Although the entire implementation representation should be examined, it may be suitable to narrow the scope of examination to the portion of the implementation representation that map the TSFI of the functional specification. For such intent, this work may be performed in parallel with the other work units that call for examining the implementation.**

ADV_IMP.2.2C The implementation representation shall be in the form used by the development personnel.

ADV_IMP.2-2 The evaluator shall check that the implementation representation is in the form used by development personnel.

The implementation representation is manipulated by the developer in form that it suitable for transformation to the actual implementation. For instance, the developer may work with files containing source code, which is eventually compiled to become part of the TSF. The developer makes available the implementation representation in the form they use, so that the evaluator may use automated techniques in the analysis. This also increases the confidence that the implementation representation examined is actually the one used in the production of the TSF (as opposed to the case where it is supplied in an alternate presentation format, such as a word processor document). It should be noted that other forms of the implementation representation may also be used by the developer; these forms are supplied as well. The overall goal is to supply the evaluator with the information that will maximise the effectiveness of the evaluator's analysis efforts.

The evaluator **examines** the **entire** implementation representation to **make sure** that it is the version that is usable by the developer. **A possible way to get** assurance that all areas of the implementation representation are in conformance with the requirement **is to make this examination using similar development tools to those of the developer.**

Conventions in some forms of the implementation representation may make it difficult or impossible to determine from just the implementation representation itself what the actual result of the compilation or run-time interpretation will be. For example, compiler directives for C language compilers will cause the compiler to exclude or include entire portions of the code.

Some forms of the implementation representation may require additional information because they introduce significant barriers to understanding and analysis. Examples include shrouded source code or source code that has been obfuscated in other ways such that it prevents understanding and/or analysis. These forms of implementation representation typically result from by taking a version of the implementation representation that is used by the TOE developer and running a shrouding or obfuscation program on it. While the shrouded representation is what is compiled and may be closer to the implementation (in terms of structure) than the original, un-shrouded representation, supplying such obfuscated code may cause significantly more time to be spent in analysis tasks involving the representation. When such forms of representation are created, the components require details on the shrouding tools/algorithms used so that the un-shrouded representation can be supplied, and the additional information can be used to gain confidence that the shrouding process does not compromise any security mechanisms.

The evaluator **checks** that all of the information needed to interpret the implementation representation has been supplied. Note that the tools are among those referenced by Tools and techniques (ALC_TAT) components. **This work may be performed in parallel with the** other work units that call for examining the implementation.

ADV_IMP.2.3C The mapping between the TOE design description and the **entire** implementation representation shall demonstrate their correspondence.

ADV_IMP.2-3 The evaluator shall examine the mapping between the TOE design description and the **entire** implementation representation to determine that it is accurate.

The evaluator augments the determination of existence (specified in work unit ADV_IMP.2-1) by verifying the accuracy of the implementation representation and the TOE design description. For **all** parts of the TOE design description that map the TSFI of the functional specification, the evaluator would verify the implementation representation accurately reflects the description provided in the TOE design description. **The TOE design description considered for the mapping with the**

implementation representation is lowest level of decomposition available in the TOE design, typically modules.

For example, the TOE design description might identify a login module that is used to identify and authenticate users. The evaluator would verify that the corresponding code in fact implements that service as described in the TOE design description, and also, accepts the parameters as described in the functional specification.

4.1.6 Evaluation of sub-activity (ADV_INT.3)

4.1.6.1 Objectives

The objective of this sub-activity is to determine whether the TSF is designed and structured with minimal complexity such that the likelihood of flaws is reduced and that maintenance can be more readily performed without the introduction of flaws.

The related part of the AIS34 from the German Scheme and the Note12 from the French Scheme were used as the basis here.

4.1.6.2 Input

The evaluation evidence for this sub-activity is:

a) the modular design description;

b) the implementation representation (if ADV_IMP is part of the claimed assurance);

c) the TSF internals description;

d) the documentation of the coding standards, as resulting from ALC_TAT.

4.1.6.3 Application notes

The role of the internals description is to provide evidence of the structure **and complexity** of the design and implementation of the TSF.

The structure of the design has two aspects: the constituent parts of the TSF and the procedures used to design the TSF. In cases where the TSF is designed in a manner consistent with the design represented by the TOE design (see ADV_TDS), the assessment of the TSF design is obvious. In cases where the design procedures (see ALC_TAT) are being followed, the assessment of the TSF design procedures is similarly obvious.

In cases where the TSF is implemented using procedure based software, this structure is assessed on the basis of its modularity; the modules identified in the internals description are the same as the modules identified in the TOE design (TOE design (ADV_TDS)). A module consists of one or more source code files that cannot be decomposed into smaller compilable units.

The complexity of the design is related to the measure of degree of difficulty to understand the design and implementation of the TSF.

The primary goal of this component is to ensure the TSF's implementation representation is understandable to facilitate maintenance and analysis (of both the developer and evaluator).

4.1.6.4 Action ADV_INT.3.1E

ADV_INT.3.1C The justification shall describe the characteristics used to judge the meaning of “well-structured” **and “complex”**.

ADV_INT.3-1 The evaluator shall examine the justification to determine that it identifies the basis for determining whether the TSF is well-structured.

The evaluator verifies that the criteria for determining the characteristic of being well-structured are clearly defined in the justification. Acceptable criteria typically originate from industry standards for the technology discipline. For example, procedural software that executes linearly is traditionally viewed as well-structured if it adheres to software engineering programming practises, such as those defined in the IEEE Standard (IEEE Std 610.12-1990). For example, it would identify the criteria for the procedural software portions of the TSF:

a) the process used for modular decomposition

b) coding standards used in the development of the implementation

c) a description of the maximum acceptable level of inter-module coupling exhibited by the TSF

d) a description of the minimum acceptable level of cohesion exhibited the modules of the TSF.

For other types of technologies used in the TOE - such as non-procedural software (e.g. object-oriented programming), widespread commodity hardware (e.g. PC microprocessors), and special-purpose hardware (e.g. smart-card processors) - the evaluation authority should be consulted for determining the adequacy of criteria for being “well-structured”.

ADV_INT.3-2 **The evaluator shall examine the justification to determine that it identifies the basis for determining whether the TSF s not overly complex**.

The evaluator verifies that the criteria for determining the characteristic of minimizing complexity are clearly defined in the justification. Acceptable criteria typically originate from industry standards for the technology discipline.

For example, it would identify software metrics to determine software complexity like McCabe’s cyclomatic complexity.

For criteria that do not originate from industry standards for the technology discipline, the evaluation authority should be consulted for determining the adequacy of criteria for being “not overly complex”.

ADV_INT.3.2C The TSF internals description shall demonstrate that the entire TSF is well-structured **and is not overly complex**.

ADV_INT.3-3 The evaluator shall examine the TSF internals description to determine that it demonstrates that the TSF is well-structured.

The evaluator examines the internals description to ensure that it provides a sound explanation of how the TSF meets the criteria from ADV_INT.3-1

For example, it would explain how the procedural software portions of the TSF meet the following:

a) that there is a one-to-one correspondence between the modules identified in the TSF and the modules described in the TOE design (ADV_TDS)

b) how the TSF design is a reflection of the modular decomposition process

c**) a justification for all instances where the coding standards were not used or met**

d) a justification for any coupling or cohesion outside the acceptable bounds.

ADV_INT.3-4 **The evaluator shall examine the TSF internals description to determine that it demonstrates that the TSF is not overly complex**.

The evaluator examines the internals description to ensure that it provides a sound explanation of how the TSF meets the criteria from ADV_INT.3-2.

4.1.6.5 Action ADV_INT.3.2E

ADV_INT.3-5 The evaluator shall determine that the TOE design is well-structured **and is not overly complex**.

The evaluator examines the TOE design of **the entire** TSF to verify the accuracy of the justification. For example, the TOE design is analysed to determine its adherence to the design standards, etc.

The description of the TOE's decomposition into subsystems and modules will make the argument that the **entire** TSF is well-structured self-evident. Verification that the procedures for structuring the TSF **and minimizing its complexity** (as examined in ALC_TAT) are being followed will make it self-evident that the **entire** TSF is well-structured **and is not overly complex**.

ADV_INT.3-6 The evaluator shall determine that the TSF is well-structured **and is not overly complex**.

If ADV_IMP is not part of the claimed assurance, then this work unit is not applicable and is therefore considered to be satisfied.

The evaluator examines the **entire** TSF to verify the accuracy of the internals description. For example, the procedural software portions of the **entire** TSF are analysed to determine their cohesion and coupling, their adherence to the coding standards, etc.

Regarding minimal complexity, for example, the evaluator may use an automated quality tool to verify that the criteria from ADV_INT.3-2 are correctly applied. Alternatively, if an automated quality tool is used by the developer, the evaluator shall qualify the tool used and analyze the results.

4.2 ATE class

4.2.1 Evaluation of sub-activity (ATE_COV.3)

4.2.1.1 Objectives

The objective of this sub-activity is to determine whether the developer has performed exhaustive tests of all interfaces, and that the developer's test coverage evidence shows correspondence between the tests identified in the test documentation and the TSFIs described in the functional specification.

4.2.1.2 Input

The evaluation evidence for this sub-activity is:

a) the ST;

b) the functional specification;

c) the test documentation;

d) the test coverage analysis.

4.2.1.3 Application notes

Regarding MILS System, if an application is developed to test the TSFI then this application has to be specified, tested and provided to the evaluator.

4.2.1.4 Action ATE_COV.3.1E

ATE_COV.3.1C The analysis of the test coverage shall demonstrate the correspondence between the tests in the test documentation and the TSFIs in the functional specification.

ATE_COV.3-1 The evaluator shall examine the test coverage analysis to determine that the correspondence between the tests in the test documentation and the interfaces in the functional specification is accurate.

A simple cross-table may be sufficient to show test correspondence. The identification of the tests and the interfaces presented in the test coverage analysis has to be unambiguous.

The evaluator is reminded that this does not imply that all tests in the test documentation must map to interfaces in the functional specification.

ATE_COV.3-2 The evaluator shall examine the test plan to determine that the testing approach for each interface demonstrates the expected behaviour of that interface.

Guidance on this work unit can be found in the [CEM]:

a) 14.2.1, Understanding the expected behaviour of the TOE

b) 14.2.2, Testing vs. alternate approaches to verify the expected behaviour of functionality

ATE_COV.3-3 The evaluator shall examine the test procedures to determine that the test prerequisites, test steps and expected result(s) adequately test each interface.

Guidance on this work units, as it pertains to the functional specification, can be found in the [CEM]:

a) 14.2.3, Verifying the adequacy of tests

ATE_COV.3.2C The analysis of the test coverage shall demonstrate that all TSFIs in the functional specification have been **completely** tested.

ATE_COV.3-4 The evaluator shall examine the test coverage analysis to determine that the correspondence between the interfaces in the functional specification and the tests in the test documentation is complete.

All TSFIs that are described in the functional specification have to be present in the test coverage analysis and mapped to tests in order for completeness to be claimed. Exhaustive specification testing of interfaces is required for this mapping. Incomplete coverage would be evident if an interface was identified in the functional specification and no test was mapped to it.

The evaluator is reminded that this does not imply that all tests in the test documentation must map to interfaces in the functional specification.

ATE_COV.3-5 The evaluator shall examine the test coverage analysis to determine that the correspondence between the interfaces in the functional specification and the tests in the test documentation shows that all TSFIs were tested completely

This means that the evaluator examines whether all aspects of purpose, method of use, parameters, parameter descriptions, actions and error messages for all TSFIs present in the functional specification are covered by the tests. Note that the level of detail present in the functional specification depends on the component of ADV_FSP chosen in the ST of the TOE.

The evaluator may conclude that the higher level descriptions in the functional specification, like purpose or method of use, are implicitly covered, if coverage of lower level descriptions like parameters, parameter descriptions, actions and error messages are covered. Therefore in general it will only be necessary to confirm coverage on these lower levels.

The developer is required to demonstrate that the tests exercise all of the parameters of all TSFIs. This additional requirement includes bounds testing (i.e. verifying that errors are generated when stated limits are exceeded) and negative testing (e.g. when access is given to User A, verifying not only that User A now has access, but also that User B did not suddenly gain access).

It should be noted that this kind of testing is not, strictly speaking, exhaustive because not every possible value of the parameters is expected to be checked.

Incomplete coverage would be evident if an interface was identified in the functional specification and no test was mapped to **one of its parameters**.

The evaluator is reminded that (for example) coverage of all parameters does not necessarily mean coverage of every possible value a parameter may allow.

Evaluator could use equivalences classes from a boundary analysis (developer have to provide boundary for each interface) to ensure that test provided by the developer covered all equivalence classes with bounds testing (in and out of range values).

A code coverage tools could be used to verify that parameter equivalences classes are correctly covered during test execution. This information doesn't mean that tests are relevant for the tested interfaces.

From these analysis, evaluator will conclude that coverage is sufficient or if additional test cases have to be added.

Similar considerations as for parameters hold for error messages specified in the functional specification: Each error message, which belongs to a qualitatively distinct error case, needs to be covered by testing. Note, that there may be exceptions, for example error messages for errors, which cannot be provoked during testing,. For such error messages other ways of coverage need to be found as discussed in 14.2.2 [CEM3], "Testing vs. alternate approaches to verify the expected behaviour of functionality".

Note that also the developer is allowed to use such alternative approaches to testing (e. g. checking something in the source code) in his coverage table. Of course the evaluator has to examine in this case, if this use of an alternative approach is acceptable (usually only in cases where testing is practically impossible).

4.2.2 Evaluation of sub-activity (ATE_FUN.2)

4.2.2.1 Objectives

The objective of this sub-activity is to determine whether the developer correctly performed and documented the tests in the test documentation, and structured the testing in order to ensure the correctness of the interfaces being tested.

4.2.2.2 Input

The evaluation evidence for this sub-activity is:

a) the ST;

b) the functional specification;

c) the test documentation.

4.2.2.3 Application notes

The extent to which the test documentation is required to cover the TSF is dependent upon the coverage assurance component.

For the developer tests provided, the evaluator determines whether the tests are repeatable, and the extent to which the developer's tests can be used for the evaluator's independent testing effort. Any TSFI for which the developer's test results indicate that it might not perform as specified should be tested independently by the evaluator to determine whether or not it does.

The developer analysis of test ordering is important for the evaluator to determine the adequacy of testing, as it provides a rationale how the ordering of tests being settled prevents the possibility of faults related to test dependencies.

Regarding MILS System, if an application is developed to test the TSFI then this application has to be specified, tested and provided to the evaluator.

4.2.2.4 Action

ATE_FUN.2.1C The test documentation shall consist of test plans, expected test results and actual test results.

ATE_FUN.2-1 The evaluator shall check that the test documentation includes test plans, expected test results and actual test results.

The evaluator checks that test plans, expected tests results and actual test results are included in the test documentation.

ATE_FUN.2.2C The test plans shall identify the tests to be performed and describe the scenarios for performing each test. These scenarios shall include any ordering dependencies on the results of other tests.

ATE_FUN.2-2 The evaluator shall examine the test plan to determine that it describes the scenarios for performing each test.

The evaluator determines that the test plan provides information about the test configuration being used: both on the configuration of the TOE and on any test equipment being used. This information should be detailed enough to ensure that the test configuration is reproducible.

The evaluator also determines that the test plan provides information about how to execute the test: any necessary automated set-up procedures (and whether they require privilege to run), inputs to be applied, how these inputs are applied, how output is obtained, any automated clean-up procedures (and whether they require privilege to run), etc. This information should be detailed enough to ensure that the test is reproducible.

The evaluator may wish to employ a sampling strategy when performing this work unit.

ATE_FUN.2-3 The evaluator shall examine the test plan to determine that the TOE test configuration is consistent with the ST.

The TOE referred to in the developer's test plan should have the same unique reference as established by the CM capabilities (ALC_CMC) sub-activities and identified in the ST introduction.

It is possible for the ST to specify more than one configuration for evaluation. The evaluator verifies that all test configurations identified in the developer test documentation are consistent with the ST. For example, the ST might define configuration options that must be set, which could have an impact upon what constitutes the TOE by including or excluding additional portions. The evaluator verifies that all such variations of the TOE are considered.

The evaluator should consider the security objectives for the operational environment described in the ST that may apply to the test environment. There may be some objectives for the operational environment that do not apply to the test environment. For example, an objective about user clearances may not apply; however, an objective about a single point of connection to a network would apply.

The evaluator may wish to employ a sampling strategy when performing this work unit.

If this work unit is applied to a component TOE that might be used/integrated in a composed TOE (see Class ACO: Composition), the following will apply. In the instances that the component TOE under evaluation depends on other components in the operational environment to support their operation, the developer may wish to consider using the other component(s) that will be used in the composed TOE to fulfil the requirements of the operational environment as one of the test configurations. This will reduce the amount an additional testing that will be required for the composed TOE evaluation.

ATE_FUN.2-4 The evaluator shall examine the test plans to determine that sufficient instructions are provided for any ordering dependencies.

Some steps may have to be performed to establish initial conditions. For example, user accounts need to be added before they can be deleted. An example of ordering dependencies on the results of other tests is the need to perform actions in a test that will result in the generation of audit records, before performing a test to consider the searching and sorting of those audit records. Another example of an ordering dependency would be where one test case generates a file of data to be used as input for another test case.

The evaluator may wish to employ a sampling strategy when performing this work unit.

ATE_FUN.2.3C The expected test results shall show the anticipated outputs from a successful execution of the tests.

ATE_FUN.2-5 The evaluator shall examine the test documentation to determine that all expected tests results are included.

The expected test results are needed to determine whether or not a test has been successfully performed. Expected test results are sufficient if they are unambiguous and consistent with expected behaviour given the testing approach.

The evaluator may wish to employ a sampling strategy when performing this work unit.

ATE_FUN.2.4C The actual test results shall be consistent with the expected test results.

ATE_FUN.2-6 The evaluator shall check that the actual test results in the test documentation are consistent with the expected test results in the test documentation.

A comparison of the actual and expected test results provided by the developer will reveal any inconsistencies between the results. It may be that a direct comparison of actual results cannot be made until some data reduction or synthesis has been first performed. In such cases, the developer's test documentation should describe the process to reduce or synthesise the actual data.

For example, the developer may need to test the contents of a message buffer after a network connection has occurred to determine the contents of the buffer. The message buffer will contain a binary number. This binary number would have to be converted to another form of data representation in order to make the test more meaningful. The conversion of this binary representation of data into a higher-level representation will have to be described by the developer in enough detail to allow an evaluator to perform the conversion process (i.e. synchronous or asynchronous transmission, number of stop bits, parity, etc.).

It should be noted that the description of the process used to reduce or synthesise the actual data is used by the evaluator not to actually perform the necessary modification but to assess whether this process is correct. It is up to the developer to transform the expected test results into a format that allows an easy comparison with the actual test results.

The evaluator may wish to employ a sampling strategy when performing this work unit.

ATE_FUN.2-7 The evaluator shall report the developer testing effort, outlining the testing approach, configuration, depth and results.

The developer testing information recorded in the ETR allows the evaluator to convey the overall testing approach and effort expended on the testing of the TOE by the developer. The intent of providing this information is to give a meaningful overview of the developer testing effort. It is not intended that the information regarding developer testing in the ETR be an exact reproduction of specific test steps or results of individual tests. The intention is to provide enough detail to allow other evaluators and evaluation authorities to gain some insight about the developer's testing approach, amount of testing performed, TOE test configurations, and the overall results of the developer testing.

Information that would typically be found in the ETR section regarding the developer testing effort is:

a) TOE test configurations. The particular configurations of the TOE that were tested, including whether any privileged code was required to set up the test or clean up afterwards
b) testing approach. An account of the overall developer testing strategy employed;
c) testing results. A description of the overall developer testing results.

This list is by no means exhaustive and is only intended to provide some context as to the type of information that should be present in the ETR concerning the developer testing effort.

ATE_FUN.2.5C The test documentation shall include an analysis of the test procedure ordering dependencies.

ATE_FUN.2-9 The evaluator shall examine the analysis of the test procedure ordering dependencies to determine that it provides a justification of the adequacy of testing.

Usually the evaluator will generate a table of all cases, where the test documentation requires a certain ordering of the tests and will then examine if sufficient justification is given in any case, why testing in this ordering is adequate and sufficient.

The test procedure ordering dependencies is twofold:

a) One type of ordering dependencies is regarding test execution. Considering that test T1 is executed immediately before test T2, the state resulting from the successful execution of test T1 shall be consistent with the pre-requisite initial test conditions of the test T2.

b) The second type of ordering dependencies is more an integrative aspect. Let's consider two sub-systems or modules A and B with their respective interfaces I_A and I_B, and T_A and T_B the respective tests for I_A and I_B. If A depends on B because interface I_A of A calls interface I_A of B, then the result of the successful execution of test T_A will only be valid after successful execution of test T_B.

As an example we assume that the TSF provide a random number generator, which needs to be initialised (for example with an adequate seed) before random numbers of a specified quality can be generated. In this case the evaluator will consider the following question:

Does the test documentation only describe an ordering of tests, where the initialisation is done before calling the function to generate a random number?

In this case the justification needs to show, why the developer expects, that in the intended environment of the TOE the random number function will not be called without initialisation of the random number generator.

If for example the user guidance documentation includes a clear instruction that the random number generator needs to be initialised adequately before being called, this may be considered adequate as a justification. (note that the question if it can be plausibly assumed that users will follow such instruction is covered by the evaluation activities for the classes ASE and AGD and needs not to be re-examined here.)

If, on the other hand, the TOE provides an authentication protocol, which implicitly uses random numbers provided by the random number generator, and an attacker can therefore "call" the random number generator implicitly by simply trying to authenticate himself, and if neither the TOE nor the environment prevent an attacker from doing this even before the random number generator is initialised, a test case needs to show, what happens in such situation.

If, for example, instead of returning a "bad" random number, the random number function would return an error, when called without proper initialisation, it would be much better to include a test showing this secure behaviour instead of trying to justify why the functions are only tested in the usual order.

Note: Of course even without ATE_FUN.2 an evaluator would be expected to look for potential vulnerabilities like the one described above. However, ATE_FUN.2.5C adds assurance by requiring the developer to give a systematic justification, why his chosen order of test cases doesn't hide such potential failures of security functions.

4.2.3 Evaluation of sub-activity (ATE_DPT.4)

4.2.3.1 Objectives

The objective of this sub-activity is to determine whether the developer has tested the all the TSF subsystems and modules against the TOE design and the security architecture description, and in accordance with the implementation representation.

4.2.3.2 Input

The evaluation evidence for this sub-activity is:

a) the ST;

b) the functional specification;

c) the TOE design;

d) the security architecture description;

e) the implementation representation;

f) the test documentation;

g) the depth of testing analysis.

4.2.3.3 Application notes

Regarding MILS System, if an application is developed to test the TSFI then this application has to be specified, tested and provided to the evaluator.

4.2.3.4 Action ATE_DPT.4.1E

ATE_DPT.4.1C The analysis of the depth of testing shall demonstrate the correspondence between the tests in the test documentation and the TSF subsystems and modules in the TOE design.

ATE_DPT.4-1 The evaluator shall examine the depth of testing analysis to determine that descriptions of the behaviour of TSF subsystems and of their interactions are included within the test documentation.

This work unit verifies the content of the correspondence between the tests and the descriptions in the TOE design. A simple cross-table may be sufficient to show test correspondence. The identification of the tests and the behaviour/interaction presented in the depth-of coverage analysis has to be unambiguous.

The evaluator is reminded that not all tests in the test documentation must map to a subsystem behaviour or interaction description.

ATE_DPT.4-2 The evaluator shall examine the test plan, test prerequisites, test steps and expected result(s) to determine that the testing approach for the behaviour description demonstrates the behaviour of that subsystem as described in the TOE design.

Guidance on this work unit can be found in the [CEM]:

a) 14.2.1, Understanding the expected behaviour of the TOE
b) 14.2.2, Testing vs. alternate approaches to verify the expected behaviour of functionality

If TSF subsystem interfaces are provided, the behaviour of those subsystems may be performed directly from those interfaces. Otherwise, the behaviour of those subsystems is tested from the TSFI interfaces. Or a combination of the two may be employed. Whatever strategy is used the evaluator will consider its appropriateness for adequately testing the behaviour that is described in the TOE design.

ATE_DPT.4-3 The evaluator shall examine the test plan, test prerequisites, test steps and expected result(s) to determine that the testing approach for the behaviour description demonstrates the interactions among subsystems as described in the TOE design.

Guidance on this work unit can be found in the [CEM]:

a) 14.2.1, Understanding the expected behaviour of the TOE
b) 14.2.2, Testing vs. alternate approaches to verify the expected behaviour of functionality

While the previous work unit addresses behaviour of subsystems, this work unit addresses the interactions among subsystems.

If TSF subsystem interfaces are provided, the interactions with other subsystems may be performed directly from those interfaces. Otherwise, the interactions among subsystems must be inferred from the TSFI interfaces.

Whatever strategy is used the evaluator will consider its appropriateness for adequately testing the interactions among subsystems that are described in the TOE design.

ATE_DPT.4-4 The evaluator shall examine the depth of testing analysis to determine that the interfaces of TSF modules are included within the test documentation.

This work unit verifies the content of the correspondence between the tests and the descriptions in the TOE design. A simple cross-table may be sufficient to show test correspondence. The identification of the tests and the behaviour/interaction presented in the depth-of coverage analysis has to be unambiguous.

The evaluator is reminded that not all tests in the test documentation must map to a subsystem behaviour or interaction description.

ATE_DPT.4-5 The evaluator shall examine the test plan, test prerequisites, test steps and expected result(s) to determine that the testing approach for each TSF module interface demonstrates the expected behaviour of that interface.

Guidance on this work unit can be found in the [CEM]:

a) 14.2.1, Understanding the expected behaviour of the TOE
b) 14.2.2, Testing vs. alternate approaches to verify the expected behaviour of functionality

Testing of an interface may be performed directly at that interface, or at the external interfaces, or a combination of both. Whatever strategy is used the evaluator will consider its appropriateness for adequately testing the interfaces. Specifically the evaluator determines whether testing at the internal interfaces is necessary or whether these internal interfaces can be adequately tested (albeit implicitly) by exercising the external interfaces.

This determination is left to the evaluator, as is its justification.

ATE_DPT.4.2C The analysis of the depth of testing shall demonstrate that all TSF subsystems in the TOE design have been tested.

ATE_DPT.4-6 The evaluator shall examine the test procedures to determine that all descriptions of TSF subsystem behaviour and interaction are tested.

This work unit verifies the completeness of work unit ATE_DPT.4-1. All descriptions of TSF subsystem behaviour and of interactions among TSF subsystems that are provided in the TOE design have to be tested. Incomplete depth of testing would be evident if a description of TSF subsystem behaviour or of interactions among TSF subsystems was identified in the TOE design and no tests could be attributed to it.

The evaluator is reminded that this does not imply that all tests in the test documentation must map to the subsystem behaviour or interaction description in the TOE design.

ATE_DPT.4.3C The analysis of the depth of testing shall demonstrate that all modules in the TOE design have been tested.

ATE_DPT.4-7 The evaluator shall examine the test procedures to determine that all interfaces of all TSF modules are tested.

This work unit verifies the completeness of work unit ATE_DPT.4-4. All interfaces of TSF modules that are provided in the TOE design have to be tested. Incomplete depth of testing would be evident if any interface of any TSF module was identified in the TOE design and no tests could be attributed to it.

The evaluator is reminded that this does not imply that all tests in the test documentation must map to an interface of a TSF module in the TOE design.

ATE_DPT.4.4C The analysis of the depth of testing shall demonstrate that all Decision Conditions in the TOE implementation have been tested correctly

ATE_DPT.4-8 The evaluator shall examine the test procedures to determine that all decision conditions in the TOE implementation are tested.

The evaluator shall verify that each decision condition is covered by at least one test.

The evaluator could use code coverage tool to ensure that each decision condition in TOE implementation is executed during test execution.

ATE_DPT.4.5C The analysis of the depth of testing shall demonstrate that the TSF operates in accordance with its implementation representation.

ATE_DPT.4-9 The evaluator shall examine the test procedures to determine that the TSF implementation representation is tested.

The evaluator shall examine the test plan, test prerequisites, test steps and expected result(s) to determine that the testing approach for the TSF demonstrates the expected behaviour of the TSF implementation representation.

Guidance on this work unit can be found in the [CEM] :

a) 14.2.1, Understanding the expected behaviour of the TOE

b) 14.2.2, Testing vs. alternate approaches to verify the expected behaviour of functionality

The evaluator will consider the appropriateness of the developer's test strategy for adequately testing the TSF against its implementation representation.

If only a sample of the implementation representation is to be examined by the evaluator (ADV_IMP.1), then the evaluator can restrict his analysis of depth of testing to this sample.

The aim of this activity is to verify that the programming language and compiler used don't introduce unexpected behavior (for example remove or modify security mechanism).

4.2.4 Evaluation of sub-activity (ATE_IND.3)

4.2.4.1 Objectives

The goal of this activity is to determine, by independently testing the TSF, whether the TOE behaves as specified in the design documentation, and to gain confidence in the developer's test results by performing all of the developer's tests.

4.2.4.2 Input

The evaluation evidence for this sub-activity is:

a) the ST;

b) the functional specification;

c) the TOE design description;

d) the operational user guidance;

e) the preparative user guidance;

f) the configuration management documentation;

g) the test documentation;

h) the TOE suitable for testing.

4.2.4.3 Application notes

None.

4.2.4.4 Action ATE_IND.3.1E

ATE_IND.3.1C The TOE shall be suitable for testing.

ATE_IND.3-1 The evaluator shall examine the TOE to determine that the test configuration is consistent with the configuration under evaluation as specified in the ST.

The TOE provided by the developer and identified in the test plan should have the same unique reference as established by the CM capabilities (ALC_CMC) sub-activities and identified in the ST introduction.

It is possible for the ST to specify more than one configuration for evaluation. The TOE may comprise a number of distinct hardware and software entities that need to be tested in accordance with the ST. The evaluator verifies that all test configurations are consistent with the ST.

The evaluator should consider the security objectives for the operational environment described in the ST that may apply to the test environment and ensure they are met in the testing environment. There may be some objectives for the operational environment that do not apply to the test environment. For example, an objective about user clearances may not apply; however, an objective about a single point of connection to a network would apply.

If any test resources are used (e.g. meters, analysers) it will be the evaluator's responsibility to ensure that these resources are calibrated correctly.

ATE_IND.3-2 The evaluator shall examine the TOE to determine that it has been installed properly and is in a known state.

It is possible for the evaluator to determine the state of the TOE in a number of ways. For example, previous successful completion of the Evaluation of sub-activity (AGD_PRE.1) sub-activity will satisfy this work unit if the evaluator still has confidence that the TOE being used for testing was installed properly and is in a known state. If this is not the case, then the evaluator should follow the developer's procedures to install and start up the TOE, using the supplied guidance only.

If the evaluator has to perform the installation procedures because the TOE is in an unknown state, this work unit when successfully completed could satisfy work unit AGD_PRE.1-3.

ATE_IND.3.2C The developer shall provide an equivalent set of resources to those that were used in the developer's functional testing of the TSF.

ATE_IND.3-3 The evaluator shall examine the set of resources provided by the developer to determine that they are equivalent to the set of resources used by the developer to functionally test the TSF.

The set of resource used by the developer is documented in the developer test plan, as considered in the Functional tests (ATE_FUN) family. The resource set may include laboratory access and special test equipment, among others. Resources that are not identical to those used by the developer need to be equivalent in terms of any impact they may have on test results.

4.2.4.5 Action ATE_IND.3.2E

ATE_IND.3-4 The evaluator shall conduct testing **repeating all of the** tests found in the developer test plan and procedures.

The aim of this work unit is to perform **all** the developer tests to confirm the validity of the developer's test results.

ATE_IND.3-5 The evaluator shall check that all the actual test results are consistent with the expected test results.

Inconsistencies between the developer's expected test results and actual test results will compel the evaluator to resolve the discrepancies. Inconsistencies encountered by the evaluator could be resolved by a valid explanation and resolution of the inconsistencies by the developer.

If a satisfactory explanation or resolution can not be reached, the evaluator's confidence in the developer's test results may be lessened: deficiencies with the developer's tests need to result in either corrective action to the TOE by the developer (e.g., if the inconsistency is caused by incorrect behaviour) or to the developer's tests

(e.g., if the inconsistency is caused by an incorrect test), or in the production of new tests by the evaluator.

4.2.4.6 Action ATE_IND.3.3E

ATE_IND.3-6 The evaluator shall devise a test **set**.

The testing strategy for this work unit is to rigorously test as many interfaces as possible.

The evaluator, when **devising the test set**, should consider the following factors:

a) The developer test evidence. The developer test evidence consists of: the test documentation, the available test coverage analysis, and the available depth of testing analysis. The developer test evidence will provide insight as to how the TSF has been exercised by the developer during testing. The evaluator applies this information when developing new tests to independently test the TOE. Specifically the evaluator should consider:

   1) augmentation of developer testing for interfaces. The evaluator may wish to perform more of the same type of tests by varying parameters to more rigorously test the interface.

   2) supplementation of developer testing strategy for interfaces. The evaluator may wish to vary the testing approach of a specific interface by testing it using another test strategy.

b) **The test coverage. All of the interfaces shall be rigorously tested.**

   Depending on the exhaustiveness of the developer testing (results from ATE_COV analysis), the evaluator shall determine the test set that is necessary to complete the testing of the entire TSF.

   For exemple, if ATE_COV.3 concludes that more tests are required to cover an interface then the evaluator may add these test to his test plan.

   Although the testing shall cover the entire TSF, the evaluator is reminded that the effort expended on the test activity should be commensurate with that expended on any other evaluation activity.

   To determine the relevant tests that need to make up the test set, the evaluator should take the following factors into consideration:

a) Rigour of developer testing of the interfaces. Those interfaces that the evaluator determines require additional testing should be included in the test **set**.

b) Developer test results. If the results of developer tests cause the evaluator to doubt that an interface is not properly implemented, then the evaluator should include such interfaces in the test **set**.

c) Significance of interfaces. Those interfaces more significant than others should be included in the test subset. One major factor of “significance” is the security-relevance (SFR-enforcing interfaces would be more significant than SFR-supporting interfaces, which are more significant than SFR-non-interfering interfaces; see CC Part 3 Section ADV_FSP). The other major factor of “significance” is the number of SFRs mapping to this interface (as determined when identifying the correspondence between levels of abstraction in ADV).

d) Implicit testing. Testing some interfaces may often implicitly test other interfaces, and their inclusion in the subset may maximise the number of interfaces tested (albeit implicitly). Certain interfaces will typically be used to provide a variety of security functionality, and will tend to be the target of an effective testing approach.

e) Types of interfaces (e.g. programmatic, command-line, protocol). The evaluator **shall include** tests for all different types of interfaces that the TOE supports.

This guidance articulates factors to consider during the selection process of an appropriate test **set**, but these are by no means exhaustive.

In case of ATE_DPT.4 is used, the evaluator may write some tests on known compiler bugs or particular language weaknesses, if ATE_DPT.4 concludes that it's required.

ATE_IND.3-7 The evaluator shall produce test documentation for the test **set** that is sufficiently detailed to enable the tests to be reproducible.

With an understanding of the expected behaviour of the TSF, from the ST, the functional specification, and the TOE design description, the evaluator has to determine the most feasible way to test the interface. Specifically the evaluator considers:

a) the approach that will be used, for instance, whether an external interface will be tested, or an internal interface using a test harness, or will an alternate test approach be employed (e.g. in exceptional circumstances, a code inspection);
b) the interface(s) that will be used to test and observe responses;
c) the initial conditions that will need to exist for the test (i.e. any particular objects or subjects that will need to exist and security attributes they will need to have);
d) special test equipment that will be required to either stimulate an interface (e.g. packet generators) or make observations of an interface (e.g. network analysers).

The evaluator may find it practical to test each interface using a series of test cases, where each test case will test a very specific aspect of expected behaviour of that interface.

If an application is needed to perform tests, the evaluator should specify this application and test it to ensure that this application doesn't introduce errors in test results.

The evaluator's test documentation should specify the derivation of each test, tracing it back to the relevant interface(s).

ATE_IND.3-8 The evaluator shall conduct testing.

The evaluator uses the test documentation developed as a basis for executing tests on the TOE. The test documentation is used as a basis for testing but this does not preclude the evaluator from performing additional ad hoc tests. The evaluator may devise new tests based on behaviour of the TOE discovered during testing. These new tests are recorded in the test documentation.

ATE_IND.3-9 The evaluator shall record the following information about the tests that compose the test **set**:

a) identification of the interface behaviour to be tested;
b) instructions to connect and setup all required test equipment as required to conduct the test;
c) instructions to establish all prerequisite test conditions;
d) instructions to stimulate the interface;
e) instructions for observing the interface;
f) descriptions of all expected results and the necessary analysis to be performed on the observed behaviour for comparison against expected results;
g) instructions to conclude the test and establish the necessary post-test state for the TOE;
h) actual test results.

The level of detail should be such that another evaluator could repeat the tests and obtain an equivalent result. While some specific details of the test results may be

different (e.g. time and date fields in an audit record) the overall result should be identical.

There may be instances when it is unnecessary to provide all the information presented in this work unit (e.g. the actual test results of a test may not require any analysis before a comparison between the expected results can be made). The determination to omit this information is left to the evaluator, as is the justification.

ATE_IND.3-10 The evaluator shall check that all actual test results are consistent with the expected test results.

Any differences in the actual and expected test results may indicate that the TOE does not perform as specified or that the evaluator test documentation may be incorrect. Unexpected actual results may require corrective maintenance to the TOE or test documentation and perhaps require rerunning of impacted tests and modifying the test sample size and composition. This determination is left to the evaluator, as is its justification.

ATE_IND.3-11 The evaluator shall report in the ETR the evaluator testing effort, outlining the testing approach, configuration, depth and results.

The evaluator testing information reported in the ETR allows the evaluator to convey the overall testing approach and effort expended on the testing activity during the evaluation. The intent of providing this information is to give a meaningful overview of the testing effort. It is not intended that the information regarding testing in the ETR be an exact reproduction of specific test instructions or results of individual tests. The intention is to provide enough detail to allow other evaluators and evaluation authorities to gain some insight about the testing approach chosen, amount of evaluator testing performed, amount of developer tests performed, TOE test configurations, and the overall results of the testing activity.

Information that would typically be found in the ETR section regarding the evaluator testing effort is:

a) TOE test configurations. The particular configurations of the TOE that were tested.
b) Interfaces tested. A brief listing of the interfaces that merited inclusion in the **test set, with a justification why the other interfaces did not need further testing**.
c) developer tests performed. **Reminder that all the developer tests performed**.
d) verdict for the activity. The overall judgement on the results of testing during the evaluation.

This list is by no means exhaustive and is only intended to provide some context as to the type of information that should be present in the ETR concerning the testing the evaluator performed during the evaluation.

4.3 AVA class

4.3.1 Specific reference

The table below gives the reference document used in this section:

Reference	Details
[AIS 34]	Application Notes and Interpretation of the Scheme (AIS), AIS34, v3, September 2009
[2411/SGDN/DCSSI/SDR]	INTERPRETATION DU VLA.4/VAN.5 DANS LE DOMAINE DU LOGICIEL, VUL/I/01.1, Version 1 (interpretation of VLA.4/VAN.5 in the domain of software) NB: DCSSI is the former name of ANSSI
[PP2009-02]	PP Embedded Software for Smart Secure Devices Basic and Extended Configurations

4.3.2 Evaluation of sub-activity (AVA_VAN.5)

4.3.2.1 Objectives

The work units for the evaluation of the sub-activity AVA_VAN.5 are copied from the work units of AVA_VAN.4 as far as possible except that the TOE is attacked by attackers possessing High attack potential. The objective of this sub-activity is to determine whether the TOE, in its operational environment, has vulnerabilities exploitable by attackers possessing High attack potential.

This section is written in the specific case of the MILS Separating Kernel technology.

4.3.2.2 Input

The evaluation evidence for this sub-activity is:

a) the ST;

b) the functional specification;

c) the TOE design;

d) the security architecture description;

e) the implementation representation (i.e white box approach);

f) source code of the whole product with a Control Flow Graph (or any equivalent tool). In case some binary module or librairies are used, it must be stated with the specifications of the librairies

g) toolchain with the detailed configuration of each tool of the chain and justification of the configuration of the tool chain,

h) the guidance documentation;users manual;

i) the TOE suitable for testing;

j) information publicly available to support the identification of possible potential vulnerabilities;

k) the results of the testing of the basic design.

The remaining implicit evaluation evidence for this sub-activity depends on the components that have been included in the assurance package. The evidence provided for each component is to be used as input in this sub-activity.

4.3.2.3 Application notes

This activity applies to cases where the developer has provided a formal security policy model of the TOE.

The methodical analysis approach takes the form of a structured examination of the evidence. This method requires the evaluator to specify the structure and form the analysis will take (i.e. the manner in which the analysis is performed is predetermined, unlike the focused analysis). The method is specified in terms of the information that will be considered and how/why it will be considered. Further guidance on methodical vulnerability analysis can be found in [Doc4] Annex B.2.2.2.3.

The activities described below are supposed to be conducted in the chronological order. The evaluator should start with activity 5-1 and finish with activity 5-13.

4.3.2.4 Action AVA_VAN.5.1E

AVA_VAN.5.1C ***The TOE shall be suitable for testing.***

AVA_VAN.5-1 The evaluator shall examine the TOE to determine that the test configuration is consistent with the configuration under evaluation as specified in the ST.

The TOE provided by the developer and identified in the test plan should have the same unique reference as established by the CM capabilities (ALC_CMC) sub-activities and identified in the ST introduction. It is possible for the ST to specify more than one configuration for evaluation. The TOE may comprise a number of distinct hardware and software entities that need to be tested in accordance with the ST. The evaluator verifies that all test configurations are consistent with the ST. The evaluator should consider the security objectives for the operational environment described in the ST that may apply to the test environment and ensure they are met in the testing environment. There may be some objectives for the operational environment that do not apply to the test environment. For example, an objective about user clearances may not apply; however, an objective about a single point of connection to a network would apply. If any test resources are used (e.g. meters, analysers) it will be the evaluator's responsibility to ensure that these resources are calibrated correctly.

AVA_VAN.5-2 The evaluator shall examine the TOE to determine that it has been installed properly and is in a known state

It is possible for the evaluator to determine the state of the TOE in a number of ways. For example, previous successful completion of the Evaluation of sub-activity (AGD_PRE.1) sub-activity will satisfy this work unit if the evaluator still has confidence that the TOE being used for testing was installed properly and is in a known state. If this is not the case, then the evaluator should follow the developer's procedures to install and start up the TOE, using the supplied guidance only.

If the evaluator has to perform the installation procedures because the TOE is in an unknown state, this work unit when successfully completed could satisfy work unit AGD PRE.1-3.

4.3.2.5 Action AVA_VAN.5.2.E

AVA_VAN.5-3 The evaluator shall examine sources of information publicly available to identify potential vulnerabilities in the TOE.

The evaluator examines the sources of information publicly available to support the identification of possible potential vulnerabilities in the TOE. There are many sources of publicly available information which the evaluator should consider using items such as those available on the world wide web, including:

- specialist publications (magazines, books);
- research papers;
- conference proceedings;
- public web database of know exploit of COTS;
- CERT bulletin;
- Developer bulletin (in case the product is not covered by a CERT) and changelog.

The evaluator should not constrain their consideration of publicly available information to the above, but should consider any other relevant information available.

The evaluator should be able to interpret publically available information. It may happen that security bugs are not clearly publically disclosed.

While examining the evidence provided the evaluator will use the information in the public domain to further search for potential vulnerabilities. Where the evaluators have identified areas of concern, the evaluator should consider information publicly available that relate to those areas of concern.

The availability of information that may be readily available to an attacker that helps to identify and facilitate attacks may substantially enhance the attack potential of a given attacker. The accessibility of vulnerability information and sophisticated attack tools on the Internet makes it more likely that this information will be used in attempts to identify potential vulnerabilities in the TOE and exploit them. Modern search tools make such information easily available to the evaluator, and the determination of resistance to published potential vulnerabilities and well known generic attacks can be achieved in a cost-effective manner. The search of the information publicly available should be focused on those sources that refer to the technologies used in the development of the product from which the TOE is derived. The extensiveness of this search should consider the following factors: TOE type, evaluator experience in this TOE type, expected attack potential and the level of ADV evidence available.

The identification process is iterative, where the identification of one potential vulnerability may lead to identifying another area of concern that requires further investigation.

The evaluator will describe the approach to be taken to identify potential vulnerabilities in the publicly available material, detailing the search to be performed. This may be driven by factors such as areas of concern identified by the evaluator, linked to the evidence the attacker is assumed to be able to obtain. However, it is recognised that in this type of search the approach may further evolve as a result of findings during the search. Therefore, the evaluator will also report any actions taken in addition to those described in the approach to further investigate issues thought to lead to potential vulnerabilities, and will report the evidence examined in completing the search for potential vulnerabilities.

4.3.2.6 Action AVA_VAN.5.3.E

AVA_VAN.5-4 The evaluator shall conduct a methodical analysis of ST, guidance documentation, functional specification, TOE design, security architecture description and implementation representation to identify possible potential vulnerabilities in the TOE.

Detailed recommendations methodical vulnerability analysis is provided in Chapter 3.

This approach to identification of potential vulnerabilities is to take an ordered and planned approach. A system is to be applied in the examination. The evaluator is to describe the method to be used in terms of the manner in which this information is to be considered and the hypothesis that is to be created.

A flaw hypothesis methodology should be used whereby the ST, development (functional specification, TOE design and implementation representation) and guidance evidence are analysed and then vulnerabilities in the TOE are hypothesised, or speculated.

The evaluator should use the knowledge of the TOE design and operation gained from the TOE deliverables to conduct a flaw hypothesis to identify potential flaws in the development of the TOE and potential errors in the specified method of operation of the TOE.

The security architecture description provides the developer vulnerability analysis, as it documents how the TSF protects itself from interference from untrusted subjects and prevents the bypass of security enforcement functionality. Therefore, the evaluator should build upon the understanding of the TSF protection gained from the analysis of this evidence and then develop this in the knowledge gained from other development (e.g. ADV) evidence.

The approach taken to the methodical search for vulnerabilities is to consider any areas of concern identified in the results of the evaluator's assessment of the development and guidance evidence. However, the evaluator should also consider each aspect of the security architecture analysis to search for any ways in which the protection of the TSF can be undermined. It may be helpful to structure the methodical analysis on the basis of the material presented in the security architecture description, introducing concerns from other ADV evidence as appropriate. The analysis can then be further developed to ensure all other material from the ADV evidence is considered.

The following provide some examples of hypotheses that may be created when examining the evidence:

a) consideration of malformed input for interfaces available to an attacker at the external interfaces;

b) examination of a key security mechanism cited in the security architecture description, such as process separation, hypothesising internal buffer overflows that may lead to degradation of separation;

c) search to identify any objects created in the TOE implementation representation that are then not fully controlled by the TSF, and could be used by an attacker to undermine SFRs.

For example, the evaluator may identify that interfaces are a potential area of weakness in the TOE and specify an approach to the search that 'all interface specifications in the evidence provided will be searched to hypothesise potential vulnerabilities' and go on to explain the methods used in the hypothesis.

In addition, areas of concern the evaluator has identified during examination of the evidence during the conduct of evaluation activities. Areas of concern may also be identified during the conduct of other work units associated with this component, in particular AVA_VAN.5-7, AVA_VAN.5-5 and AVA_VAN.5-6) where the development and conduct of penetration tests may identify further areas of concerns for investigation, or potential vulnerabilities.

However, examination of only a subset of the development and guidance evidence or their contents is not permitted in this level of rigour. The approach description should provide a demonstration that the methodical approach used is complete, providing confidence that the approach used to search the deliverables has considered all of the information provided in those deliverables.

This approach to identification of potential vulnerabilities is to take an ordered and planned approach; applying a system to the examination. The evaluator is to describe the method to be used in terms of how the evidence will be considered; the manner in which this information is to be considered and the hypothesis that is to be created. This approach should be agreed with the evaluation authority, and the evaluation authority should provide detail of any additional approaches the evaluator should take to the vulnerability analysis and identify any additional information that should be considered by the evaluator.

Although a system to identifying potential vulnerabilities is predefined, the identification process may still be iterative, where the identification of one potential vulnerability may lead to identifying another area of concern that requires further investigation.

Subject to the SFRs the TOE is to meet in the operational environment, the evaluator's independent vulnerability analysis should consider generic potential vulnerabilities under each of the following headings:

a) Software attacks at Separation Kernel interfaces level

b) bypassing;

c) tampering;

d) direct attacks;

e) monitoring;

f) misuse.

Items b) - f) are explained in greater detail in Annex B.2.1.

The security architecture description should be considered in light of each of the above generic potential vulnerabilities. Each potential vulnerability should be considered to search for possible ways in which to defeat the TSF protection and undermine the TSF.

AVA_VAN.5-5 The evaluator shall record in the ETR the identified potential vulnerabilities that are candidates for testing and applicable to the TOE in its operational environment.

It may be identified that no further consideration of the potential vulnerability is required if for example the evaluator identifies that measures in the operational environment, either IT or non-IT, prevent exploitation of the potential vulnerability in that operational environment. For instance, restricting physical access to the TOE to authorised users only may effectively render a potential vulnerability to tampering unexploitable.

The evaluator records any reasons for exclusion of potential vulnerabilities from further consideration if the evaluator determines that the potential vulnerability is not applicable in the operational environment. Otherwise the evaluator records the

potential vulnerability for further consideration.

A list of potential vulnerabilities applicable to the TOE in its operational environment, which can be used as an input into penetration testing activities, shall be reported in the ETR by the evaluators.

4.3.2.7 Action AVA_VAN.5.4.E

AVA_VAN.5-6 The evaluator shall devise penetration tests, based on the independent search for potential vulnerabilities.

The evaluator prepares for penetration testing as necessary to determine the susceptibility of the TOE, in its operational environment, to the potential vulnerabilities identified during the search of the sources of publicly available information and the analysis of the TOE guidance and design evidence. The evaluator should have access to current information (e.g. from the evaluation authority) regarding known potential vulnerabilities that may not have been considered by the evaluator.

The evaluator is reminded that, as for considering the security architecture description in the search for vulnerabilities (as detailed in AVA_VAN.5-3), testing should be performed to confirm the architectural properties. If requirements from ATE_DPT are included in the SARs, the developer testing evidence will include testing performed to confirm the correct implementation of any specific mechanisms detailed in the security architecture description. However, the developer testing will not necessarily include testing of all aspects of the architectural properties that protect the TSF, as much of this testing will be negative testing in nature, attempting to disprove the properties. In developing the strategy for penetration testing, the evaluator will ensure that all aspects of the security architecture description are tested, either in functional testing (as considered in 14) or evaluator penetration testing.

The evaluator will probably find it practical to carry out penetration test using a series of test cases, where each test case will test for a specific potential vulnerability.

The evaluator is not expected to test for potential vulnerabilities (including those in the public domain) beyond those which required a High attack potential. In some cases, however, it will be necessary to carry out a test before the exploitability can be determined. Where, as a result of evaluation expertise, the evaluator discovers an exploitable vulnerability that is beyond High attack potential, this is reported in the ETR as a residual vulnerability.

Guidance on determining the necessary attack potential to exploit a potential vulnerability can be found in Annex B.4.

Potential vulnerabilities hypothesised as exploitable by an attacker possessing a High (or less) attack potential and resulting in a violation of the security objectives should be the highest priority potential vulnerabilities comprising the list used to direct penetration testing against the TOE.

AVA_VAN.5-7 The evaluator shall produce penetration test documentation for the tests based on the list of potential vulnerabilities in sufficient detail to enable the tests to be repeatable. The test documentation shall include:

- identification of the potential vulnerability the TOE is being tested for;
- instructions to connect and setup all required test equipment as required to conduct the penetration test;
- instructions to establish all penetration test prerequisite initial conditions;

- instructions to stimulate the TSF;
- instructions for observing the behaviour of the TSF;
- descriptions of all expected results and the necessary analysis to be performed on the observed behaviour for comparison against expected results;
- instructions to conclude the test and establish the necessary post-test state for the TOE.

The evaluator prepares for penetration testing based on the list of potential vulnerabilities identified during the search of the public domain and the analysis of the evaluation evidence.

The evaluator is not expected to determine the exploitability for potential vulnerabilities beyond those for which a High attack potential is required to effect an attack. However, as a result of evaluation expertise, the evaluator may discover a potential vulnerability that is exploitable only by an attacker with greater than High attack potential. Such vulnerabilities are to be reported in the ETR as residual vulnerabilities.

With an understanding of the potential vulnerability, the evaluator determines the most feasible way to test for the TOE's susceptibility. Specifically the evaluator considers:

- the TSFI or other TOE interface that will be used to stimulate the TSF and observe responses (It is possible that the evaluator will need to use an interface to the TOE other than the TSFI to demonstrate properties of the TSF such as those described in the security architecture description (as required by ADV_ARC). It should the noted, that although these TOE interfaces provide a means of testing the TSF properties, they are not the subject of the test.);
- initial conditions that will need to exist for the test (i.e. any particular objects or subjects that will need to exist and security attributes they will need to have);
- special test equipment that will be required to either stimulate a TSFI or make observations of a TSFI;
- whether theoretical analysis should replace physical testing, particularly relevant where the results of an initial test can be extrapolated to demonstrate that repeated attempts of an attack are likely to succeed after a given number of attempts.

The evaluator will probably find it practical to carry out penetration testing using a series of test cases, where each test case will test for a specific potential vulnerability.

The intent of specifying this level of detail in the test documentation is to allow another evaluator to repeat the tests and obtain an equivalent result.

AVA_VAN.5-8 The evaluator shall conduct penetration testing.

The evaluator uses the penetration test documentation resulting from work unit AVAVAN.5-6 as a basis for executing penetration tests on the TOE, but this does not preclude the evaluator from performing additional ad hoc penetration tests. If required, the evaluator may devise ad hoc tests as a result of information learnt during penetration testing that, if performed by the evaluator, are to be recorded in the penetration test documentation. Such tests may be required to follow up unexpected results or observations, or to investigate potential vulnerabilities suggested to the evaluator during the pre-planned testing.

Should penetration testing show that a hypothesised potential vulnerability does not exist, then the evaluator should determine whether or not the evaluator's own analysis was incorrect, or if evaluation deliverables are incorrect or incomplete.

The evaluator is not expected to test for potential vulnerabilities (including those in the public domain) beyond those which required a High attack potential. In some cases, however, it will be necessary to carry out a test before the exploitability can be determined. Where, as a result of evaluation expertise, the evaluator discovers an exploitable vulnerability that is beyond High attack potential, this is reported in the ETR as a residual vulnerability.

AVA_VAN.5-9 The evaluator shall conduct some systematic security verification test activities.

Thanks to the constant augmentation of computing power of hardware, the evaluator should be able to access a rather high computing power at reasonable price. This allows performing some systematic security verification test activities on the TOE.

Even if such tests are performed in a blackbox approach (ie without pursuing a specific attack path), they complement the tests performed via the vulnerability analysis.

In addition, Internet allows accessing freely sophisticated tools (either open source or closed source) that are useful to assess the security of the TOE.

Systematic security verification tests should include:

- Fuzzing the TSFI (ie TSF interfaces) with totally random input or pseudo-random input;
- Static and dynamic analysis of the source code to inspect potential buffer overflow and race conditions

AVA_VAN.5-10 The evaluator shall record the actual results of the penetration tests.

While some specific details of the actual test results may be different from those expected (e.g. time and date fields in an audit record) the overall result should be identical. Any unexpected test results should be investigated. The impact on the evaluation should be stated and justified.

AVA_VAN.5-11 The evaluator shall report in the ETR the evaluator penetration testing effort, outlining the testing approach, configuration, depth and results.

The penetration testing information reported in the ETR allows the evaluator to convey the overall penetration testing approach and effort expended on this sub-activity. The intent of providing this information is to give a meaningful overview of the evaluator's penetration testing effort. It is not intended that the information regarding penetration testing in the ETR be an exact reproduction of specific test steps or results of individual penetration tests. The intention is to provide enough detail to allow other evaluators and evaluation authorities to gain some insight about the penetration testing approach chosen, amount of penetration testing performed, TOE test configurations, and the overall results of the penetration testing activity.

Information that would typically be found in the ETR section regarding evaluator penetration testing efforts is:

- TOE test configurations. The particular configurations of the TOE that were penetration tested;
- TSFI penetration tested. A brief listing of the TSFI and other TOE interfaces that were the focus of the penetration testing;
- Verdict for the sub-activity. The overall judgement on the results of penetration testing.

This list is by no means exhaustive and is only intended to provide some context as to the type of information that should be present in the ETR concerning the penetration testing the evaluator performed during the evaluation.

AVA_VAN.5-12 The evaluator shall examine the results of all penetration testing to determine that the TOE, in its operational environment, is resistant to an attacker possessing a High attack potential.

If the results reveal that the TOE, in its operational environment, has vulnerabilities exploitable by an attacker possessing an attack potential less than or equal to High, then this evaluator action fails.

The guidance in B.4 and the guidance for special technical areas (e.g. AIS 26) that is relevant for the national scheme should be used to determine the attack potential required to exploit a particular vulnerability and whether it can therefore be exploited in the intended environment. It may not be necessary for the attack potential to be calculated in every instance, only if there is some doubt as to whether or not the vulnerability can be exploited by an attacker possessing an attack potential less than or equal to High.

AVA_VAN.5-13 The evaluator shall report in the corresponding ETR-part all exploitable vulnerabilities and residual vulnerabilities, detailing for each:

- its source (e.g. CEM activity being undertaken when it was conceived, known to the evaluator, read in a publication);
- the SFR(s) not met;
- a description;
- whether it is exploitable in its operational environment or not (i.e. exploitable or residual);
- the amount of time, level of expertise, level of knowledge of the TOE, level of opportunity and the equipment required to perform the identified vulnerabilities, and the corresponding values using the tables 3 and 4 of Annex B.4.

4.3.3 Recommendations for vulnerability analysis for AVA_VAN.5

The main goal of this chapter is to complement and detail the action AVA_VAN.5.3.E and especially the requirement AVA_VAN.5.4.

This chapter is based on Document 4 Appendix B.2.1 dealing with generic vulnerability guidance.

The same approach and taxonomy is used here and applied to the context of MILS systems.

4.3.3.1 Introduction

This section of the document has been written based on some assumptions:

- A MILS system is a mainly composed of 3 layers:
  - the hardware layer that physically runs the MILS system and contains the physical I/O,
  - the Separating Kernel layer (which is software) in charge of managing the hardware resources and providing services for application layer,
  - the applications (which are software) running on top of the Separating Kernel and using the service offered

- The evaluation of the MILS system is not monolithic and is performed by composition of layers: the hardware is certified and provides recommendations to be enforced by the Operating System. The Separating Kernel is certified providing that it is combined with a certified hardware and follows recommendations of the certified hardware. The Separating Kernel provides recommendations to be enforced by the upper layer (ie. applications). The applications are certified providing that it is combined with a certified Operating System and follow the recommendations of the certified Operating System.

As the evaluation of the MILS system is the combination of multiple evaluations, physical phenomenon that are independent of the Separating Kernel such as electromagnetic or optical emanation, or sensibility to external events (sensibility to modification of temperature, pressure, ….) are not studied during the Separating Kernel evaluation. They are studied during the Separating Kernel hardware evaluation.

On the other hand, in case some physical elements are managed by the Separating Kernel with no added value from the Separating Kernel hardware layer, attacks have to be studied during the Separating Kernel evaluation.

For example, in case of a USB port, the hardware layer will forward signals to the Separating Kernel. As such, it is an external interface of the Separating Kernel.

In the section below, the word TOE refers to a Separating Kernel under evaluation, fulfilling the assumptions that:

- the hardware layer is certified at least at the same assurance level as the one claimed by the Separating Kernel,
- the Separating Kernel is enforcing all recommendations of the certified hardware.

The reader shall be comfortable with two notions: perimeter defence and defence in depth:

Perimeter defence refers to a system where all security functions are enforced at the boundary of the system with external interfaces. It considers that all external threats will be handled at the boundary and that no malicious activity will be surviving the perimeter defense.

On the other hand, the defense in depth is a concept in which multiple layers of security controls (defense) are placed throughout a system. Its intent is to provide redundancy in the event a security control fails.

Finally this section considers a Seperating Kernel OS which does not contain any cryptographic functions.

4.3.3.2 About Methodical Analysis

As described in the CEM, AVA_VAN.5 requires the definition of the methodology applies to examine in a systematic way the TOE and evidences associated at different level of abstractions. As described, the methodical analysis approach takes the form of a structured examination of the evidence. This method requires the evaluator to specify the structure and form the analysis will take (i.e. the manner in which the analysis is performed is predetermined, unlike the focused identification method). The method is specified in terms of the information that will be considered and how/why it will be considered. This approach to the identification of potential vulnerabilities can be used during the independent vulnerability analysis required by Evaluation of sub-activity (AVA_VAN.4) and Evaluation of sub-activity (AVA_VAN.5).

This analysis of the evidence is deliberate and pre-planned in approach, considering all evidence identified as an input into the analysis.

All evidence provided to satisfy the (ADV) assurance requirements specified in the assurance package are used as input to the potential vulnerability identification activity.

The “methodical” descriptor for this analysis has been used in an attempt to capture the characterization that this identification of potential vulnerabilities is to take an ordered and planned approach. A “method” or “system” is to be applied in the examination. The evaluator is to describe the method to be used in terms of what evidence will be considered, the information within the evidence that is to be examined, the manner in which this information is to be considered; and the hypothesis that is to be generated.

The following provide some examples that a hypothesis may take:

a) consideration of malformed input for interfaces available to an attacker at the external interfaces;

b) examination of a security mechanism, such as domain separation, hypothesizing internal buffer overflows leading to degradation of separation;

c) analysis to identify any objects created in the TOE implementation representation that are then not fully controlled by the TSF, and could be used by an attacker to undermine the SFRs.

For example, the evaluator may identify that interfaces are a potential area of weakness in the TOE and specify an approach to the analysis that “all interface specifications provided in the functional specification and TOE design will be analyzed to hypothesize potential vulnerabilities” and go on to explain the methods used in the hypothesis.

This identification method will provide a plan of attack of the TOE that would be performed by an evaluator completing penetration testing of potential vulnerabilities in the TOE. The rationale for the method of identification would provide the evidence for the coverage and depth of exploitation determination that would be performed on the TOE.

In conclusion, a methodical approach is to define some elementary components and different kind of attacks: a systematic application of all kind of attacks to each component (mainly based on APIs of the components) provide a systematic approach to evaluate weaknesses and their potential consequences. By this way, a systematic approach is applied to cover (with identified hypotheses) the TOE.

4.3.3.3 Context and overview

In the context of the MILS, the recommendations and approach defined in the CEM have to be tailored.

The cornerstone of a MILS system is the Separating Kernel which is mainly software. The class of weakness than can be exploited to lead to vulnerability is not the same as a monolithic system that contains both hardware and software.

With regards to the PP and the ST defined in the context of the EuroMils project, the definition of the TOE is not only a Separating Kernel (as PikeOS) but a generic configuration of a Separating Kernel with diverse partitions that may contain any kind of applications. As described in the ST, the type of TOE considered is mainly:

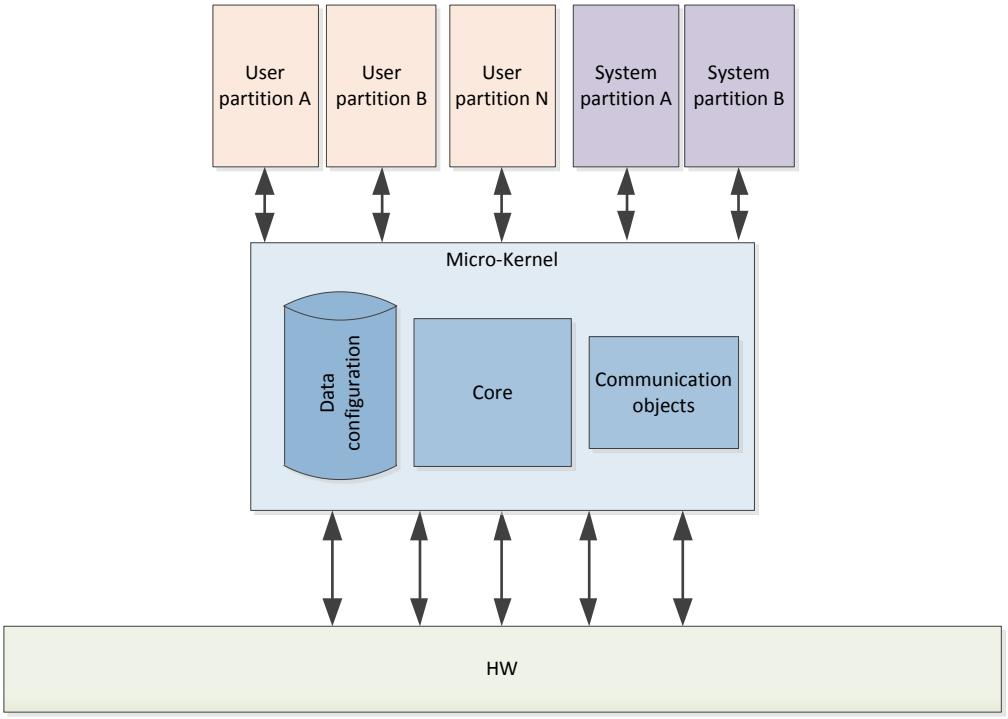


Figure 2: General MILS System

Starting from this TOE we can define arbitrary major components of the TOE. They can be abstracted to:

- The Separating Kernel composed by
  - The APIs offered by the Separating Kernel to the applications. The applications use these APIs (a.k.a hypercall) to exchange with the core kernel.
  - The communication object proposed by the Separating Kernel to the applications to support inter-partitions communications,
  - The configuration data which defines rights and properties of partitions
- The partitions hosting the applications (which can be malicious or not) and managed by the Separation Kernel.

This split of a Separating Kernel product is a suggestion and is a macro vision of the components of a Separation Kernel. It should be possible to split in a more refined way.

For example, the physical I/O (e.g; USB, network, serial,…) management functions or the RAM management functions could be considered themselves as a component on which the methodology detailed below could be applied

This list is an abstract view of the TOE and could be refined to lists more detailed modules. Each element of this list shall be considered as atomic (at a certain abstraction level) and the vulnerability analysis shall identify weakness on their.

4.3.3.4 Different methodologies for assurance level

From these first considerations, we can define two major approaches for vulnerability analysis:

- An external approach which consists in identifying vulnerabilities based on stimulus and attacks at the external interface of the TOE (i.e. the integration of the Separating Kernel and the diverse applications). In this approach, the Separating Kernel internal interfaces (i.e; internal functions, internal mechanisms) are not visible to the attacker.
- An internal approach (a.k.a white box) which consists in detailing the internal behavior of the TOE and knowing precisely the path used by every external stimulus once inside the TOE. It is similar to a tainted analysis where each data provided at an external interface is traced within the TOE in order to inspect any function (i.e. piece of code) making use of this data (that could be malicious). This allows, among other things, identifying functions using external data in a risky way (e.g; not sanitizing the data).

These two different approaches can be applied for different context:

- The external approach allows having some confidence in the robustness of the TOE based on a perimeter defense. This approach provides a first level of confidence but should not be used to reach a high assurance level.
- The internal approach provides a higher level of information and allows a defense in depth paradigm. This approach should be use when claiming a high level of assurance for the evaluation.

The annex B section 2 of the CEM presents different kind of generic attacks to be considered during the vulnerability analysis.

Due to the universal and technology agnostic level of the CEM, we need to refine and to detail the nature of the attacks proposed by the CEM for MILS technology.

This refinement is an important part of the definition of the methodology for the vulnerability analysis.

As the external methodology should not be used when claiming high assurance level, we will study directly the internal methodology.

4.3.3.5 Internal Methodology

With regards to the two major approaches defined previously we propose a systematic examination of the behavior of the atomic element. The list of components considered is:

- Kernel core functions (i.e subcomponents of the kernel core)
- Communication objects (which supports data exchanges between two partitions),
- Configuration data (which manage rules and rights of each partitions on the memory and communication memory…). It is named TSF data in the Common Criteria vocabulary
- Users application

For each components of the previous list, we applied a set of attack type as recommended in the CEM. But the nature of the attack is “specialized” with regard of the component considered and its interactions with others.

Using this internal methodology assume that the ITSEF has access to the architecture of the source code (traceability in the flow of execution), all the source code of the TOE whatever language used and the detailed toolchain.

4.3.3.5.1 Recommendations for Kernel Core vulnerability analysis

4.3.3.5.1.1.1 Attacks based on bypassing

Reminder: according to the Document 4 annex B, “bypassing” is defined by:

Bypassing includes any means by which an attacker could avoid security enforcement, by:

- exploiting the capabilities of interfaces to the TOE, or of utilities which can interact with the TOE;
- inheriting privileges or other capabilities that should otherwise be denied;
- (where confidentiality is a concern) reading sensitive data stored or copied to inadequately protected areas.

During start-up and shutdown phases of a MILS system, all internal APIs used to initiate respectively stop the core of the Separating Kernel (for example reading of configuration data, initialization of internal state machines …) shall be non by-passable.

For example, if during startup of the system, a first boot code is read directly in the hardware (e.g; chip), then execution is provided to a first low-level boot code and then high-level boot code, the TOE shall ensure at minimum that those code are not modified since the version provided by the developer.

The TOE shall also ensure that the boot chain is not altered, i.e the order of the boot sequence is not modified (e.g; some boot code are skipped or a new code is injected).

If the sequence of the call of functions is important for functional or security reasons the developer shall control the sequence and the integrity by security mechanisms (magic bytes, state machine with magic values, digital signature …).

The TOE may use various scripts that are used at the end of the boot process to fine tune the configuration. The TOE shall ensure that those script are not altered and are called following strictly the planned order.

On the other, when the system is shutting down, the TOE shall ensure that all mechanisms contributing to security (e.g; memory cleaning functions) are not bypassed before the system switches off.

*The **evaluator** checks for all init, startup and shutdown functions (and more generally for all functions identified in documentation with influence on behavior of the kerne)l than security mechanisms against bypassing are correctly implemented.*

4.3.3.5.1.1.2 Attacks based on Tampering

Reminder: according to the Document 4 annex B, “tampering” is defined by:

Tampering includes any attack based on an attacker attempting to influence the behavior of the TSF (i.e. corruption or de-activation), for example by:

- accessing data on whose confidentiality or integrity the TSF relies;
- forcing the TOE to cope with unusual or unexpected circumstances;
- disabling or delaying security enforcement;
- physical modification the TOE.

For example the configuration of the channels between the partitions could be modified in order to open new channel not expected. It can also be accessing a secret, like a cryptographic key, used during the authentication of the TOE to another server.

Forcing the TOE to cope with unusual or unexpected circumstances could be a malicious application trying to call APIs of the kernel (a.k.a hypercall) out the context of use: with bad parameters or not in

the appropriate timing. For example, a malicious application tries to read in its shared communication objects before its instantiation or write values in some memory partition of the kernel to deactivate some security mechanisms.

NB: attacks at physical level (i.e; direct access on the physical components of the TOE) should be considered during the evaluation of the MILS hardware level.

Such kind of physical modification could be for example replacing a physical component of the TOE that contributes to security mechanisms in order to downgrade TSF.

*The **evaluator** shall check for each APIs offered by the kernel to application the behavior of the kernel in case of bad parameters or incorrect use of the API to verify than security mechanisms assure non communication of sensitive data to unauthorized application.*

*The **evaluator** shall verify than all security variables of the kernel identified in the documentation (status of kernel, status of authorizations …) is protected against attacks based on tampering by specific coding values or mirroring.*

4.3.3.5.1.1.3 Direct attacks

This category of attacks pertains mainly to mathematics and cryptographic mechanisms. As such, it is not applicable in the MILS context.

4.3.3.5.1.1.4 Attacks based on monitoring

Reminder: according to the Document 4 annex B, “information” is defined by:

Information is an abstract view on relation between the properties of entities, i.e. a signal contains information for a system, if the TOE is able to react to this signal. The TOE resources process and store information represented by user data. Therefore:

- information may flow with the user data between subjects by internal TOE transfer or export from the TOE;
- information may be generated and passed to other user data;
- information may be gained through monitoring the operations on data representing the information.

Dedicated tool can be used to identify interactions between the Separating Kernel core and the hardware which support the security features. With the help of this kind of tools an attacker can identify the access to MPU or MMU and some others important security mechanisms. This information can be used to deploy efficiently other kind of attacks.

In the context of MILS Separating Kernel, energy emanation (such as electromagnetic or optical) eavesdropping is not taken into account.

4.3.3.5.1.1.5 Attacks based on Misuse

Reminder: according to the Document 4 annex B, “Misuse” is defined by:

Misuse may arise from:

- incomplete guidance documentation;
- unreasonable guidance;
- unintended misconfiguration of the TOE;
- forced exception behavior of the TOE.

Misuse of API of the kernel by the applications customers of the different services. The log files and associated data generated in case of crash of the system shall be considered and no confidential data can be recovery by analysis of the log files.

*The **evaluator** shall verify for security involved applications than the usage of the APIs of the kernel is consistent with documentation and recommendations of the guides.*

*The **evaluator** shall verify the content of the log file to check than no confidential data can be found in case of crash of the system (for example memory dump).*

4.3.3.5.1.2 Recommendations for Communication objects vulnerability analysis

4.3.3.5.1.2.1 Attacks based on bypassing

An attacker should not be able to take advantage of the expected communication objects to share other information than the one specified.

*The **evaluator** shall verify that communication objects can not be bypassed in order to exchange other information that are not expected to be exchanged.*

4.3.3.5.1.2.2 Attacks based on Tampering

The data contained by a communication object shall be preserved of unauthorized access and corruption. These features are implemented by the Separating Kernel itself. A malicious application should not be able to corrupt communication objects in transit. If this occurs, this means that the kernel core was corrupted by an attack: refers to chapter 3.5.1.

The evaluator shall verify that communication objects can not be corrupted (ie loss of integriy).

4.3.3.5.1.2.3 Direct attacks

This category of attacks pertains mainly to mathematics and cryptographic mechanisms. As such, it is not applicable in the MILS context.

4.3.3.5.1.2.4 Attacks based on monitoring

An attacker should not be able to monitor the behavior of the Separating Kernel when exchanging information from one partition to another. This would help the attacker to better understand the internal mechanisms anticipate the future behavior of the Separating Kernel.

The evaluator shall verify that communication objects can not be monitored during internal Separating Kernel transit mechanisms are active and that communication objects behavior can not be predicted.

4.3.3.5.1.2.5 Attacks based on Misuse

In case of incorrect use of tools and configuration, the communication objects and their properties are not correctly configured. It can lead to give the access rights of some communication objects to a malicious application.

*The **evaluator** shall check than the guidance is defined and applied for configuration of all communication objects.*

4.3.3.5.1.3 Recommendations for Configuration Data vulnerability analysis

4.3.3.5.1.3.1 Attacks based on bypassing

An attacker should not be able to bypass the legitimate Configuration Data with specifically crafted malicious configuration Data (a.k.a TSF data). There should be a form of authentication between the TSF and TSF data to avoid such situations.

An attacker should also not be able to simulate an empty configuration data or missing configuration data that could set the TOE in a default mode.

*The **evaluator** shall check that only legitimate Configuration Data (TSF data) are used and that the loading mechanism can not be bypassed.*

4.3.3.5.1.3.2 Attacks based on Tampering

By modification of configuration data (TSF data), a malicious application can modify the behavior of the TSF.

*The **evaluator** shall evaluate the check of consistency implemented for configuration data especially that the configuration data are not dynamically modified or than the coding of data use classic technics as magic mirror, etc…*

4.3.3.5.1.3.3 Direct attacks

This category of attacks pertains mainly to mathematics and cryptographic mechanisms. As such, it is not applicable in the MILS context.

4.3.3.5.1.3.4 Attacks based on monitoring

An attacker could intercept or investigate the configuration data in order to anticipate the behavior of the TSF and easily plan an attack.

*The **evaluator** shall verify that the configuration data (TSF data) can only be read (and modified) by the relevant TSF.*

4.3.3.5.1.3.5 Attacks based on Misuse

A misuse of the integrator of the system can lead to incorrect configuration and can permit access to a malicious application to data/functions. The integrator of the system shall be very rigorous about the rights management and the correct use of chain tools and associated documents.

*The **evaluator** shall check than the documentation for configure and use communication objects is complete and consistent. Additionally the evaluator shall checks than the documentation was correctly applied by the integrator (or/and developer) during definition and configuration of communication objects.*

4.3.3.5.1.4 Recommendations for User Application vulnerability analysis

4.3.3.5.1.4.1 Attacks based on bypassing

An attack based on bypassing a user application means that the kernel is not able performing the expected services (refers to ST).

Remark: the non-execution of a user partition can lead for example not to crypt information before sending them if the encryption algorithm is implemented in a different partition and communicate with user application by communication objects.

There should be a form of authentication between the User Application and Kernel to avoid such situations.

*The **evaluator** shall check that user application accessing the Separating Kernel can not be replaced by another malicious application.*

4.3.3.5.1.4.2 Attacks based on Tampering

In this specific case, tampering of user application (ie the content of a user partition) is out of the scope of the evaluation of the Separating Kernel.

Indeed the content of user partition is not supervised by the Separating Kernel.

The risk of a user partition being attacked and containing malware has to be taken into account at the Separating Kernel.

4.3.3.5.1.4.3 Direct attacks

This category of attacks pertains mainly to mathematics and cryptographic mechanisms. As such, it is not applicable in the MILS context.

4.3.3.5.1.4.4 Attacks based on monitoring

In this specific case, monitoring a user application (ie the content of a user partition) is out of the scope of the evaluation of the Separating Kernel.

4.3.3.5.1.4.5 Attacks based on Misuse

In this specific case, Misuse on user application (ie the content of a user partition) is out of the scope of the evaluation of the Separating Kernel.

4.3.3.6 Black box Methodology (a.k.a external methodology)

In the context of AVA_VAN.5, the black box approach is not applicable.

With regards to the **high** potential of the attacker and the completeness required, the analysis of internal documentation, the reviews of the code audit… are required for the vulnerability analysis.

Chapter 5 Composition

This fifth chapter of this document study specifically the aspect of the Common Criteria composition between software platforms, for example like applications on top of certified platform like operating systems.

The composition is studied both on the point of view of the construction and on the point of view of the evaluation.

5.1 Composite product evaluation for software platform

5.1.1 Introduction

5.1.1.1 Preface

The Common Criteria (CC) are being widely used for software platform products security evaluation. Software platform products are represented by different types of software environments, where applications are executed.

There may be operating systems for mainframe and for personal computers, for embedded devices like electronic control units, network devices and meters, for smart phones and other mobile devices. There may be Java runtime environments.

A security composition in the area of smart cards and similar devices has already been covered by the related supporting documents:

- SOGIS: Composite product evaluation for Smartcards and similar devices [JIL COMP SC],
- CCMC: CCDB-2012-04-001 - Composite product evaluation for Smartcards and similar devices [CCDB COMP SC].

The composite methodology for smart cards and similar devices has proven itself in praxis. The current document is inspired by this methodology and re-uses it for the current purpose.

The reason for a composite approach is that almost all real products can be represented as a combination of two parts: an executing environment part (abstract machine) and an application part often developed by different actors with specific objectives.

One objective is to independently perform a dedicated security evaluation and certification of an executing environment (abstract machine, platform) to address several applications and customers. The executing environment may comprise hardware parts or may purely consist of software.

Another objective is to create one or several applications to load on one or several certified platforms and to certify the final product.

The objective for composite product integration is to install one or several applications onto one already certified platform to reduce the evaluation effort and keeping a high level of assurance.

To achieve these objectives, a transfer of knowledge and a reuse of evidences and results have been defined.

5.1.1.2 Definitions

The executing environment (abstract machine, platform) is evaluated independently as it can be used with many different software applications. The executing environment may comprise hardware parts or may purely consist of software.

The application is installed on the executing environment and is built to operate with this platform. The resulting product is the one being used in the field (mainframe, applications for personal computers, embedded devices like electronic control units, network devices and meters, apps for smart phones and other mobile devices, Java apps) and on which customers/operators/users need to gain assurance.

An important specificity of any composite product type is that the application part uses and may or has to control and/or configure the security functionality provided by the executing environment

5.1.1.3 Composite product evaluation and ACO (CC V3.1)

Although the CC version 3 introduces the specific assurance class ACO for composition, this class is not suitable for usual constellation as described above.

ACO addresses a TOE composed of two certified TOEs: the Base TOE and the Dependent TOE (see Figure 1). The evaluation of the composed TOE consists in evaluating the interaction between both TOEs, reusing evaluation results of Base TOE and Dependent TOE.

The result of this evaluation is not an EAL level, but a CAP level which is not comparable to EAL level. Furthermore, ACO class is applicable up to Extended-Basic assurance level, whereas smart cards especially in banking or signature type application require ‘High Level’ assurance.

1

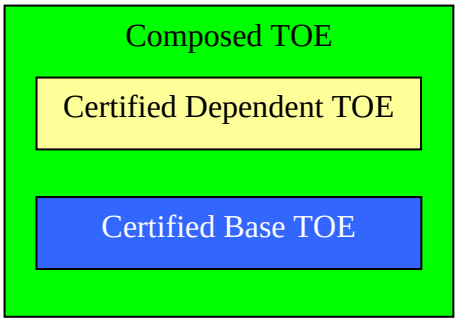


Diagram 1 ACO composed TOE (package CAP)

The composite product is the final product for which an EAL level certification is required by customers/operators. This allows a direct comparison with similar products certified after a single evaluation and facilitates building an integrated system of single components on an equal/homogeneous assurance level.

Considering the current composite approach, the final product is composed of an executing environment (platform) and a software application. In a Composite TOE evaluation, the platform is certified, the application is evaluated and the results of the platform certification are reused. See Figure 2 for security certification of the entire Composite TOE.

1

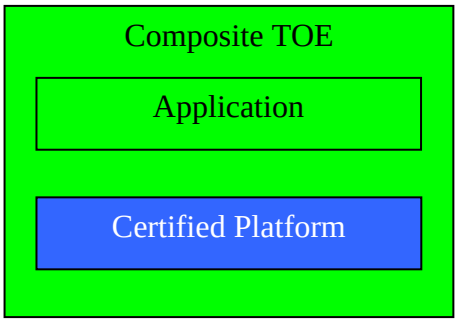


Diagram 2 Composite product evaluation (current approach)

The certified platform provides security functionality to protect the final, composite product assets, but the composite product behaviour widely depends on the software application using and perhaps having to manage this security functionality.

Therefore, the platform evaluation results provide security recommendations and conditions for the software application implementation. The composite product evaluator shall examine amongst other that the combination of both products does not lead to any exploitable vulnerability.

The current composite evaluation methodology defines precise work units with clear statement on the information needed from the platform developer and provides an agreed “framework” for information transfer from platform to composite product evaluator.

The information required is already available from the platform evaluation tasks and no additional work is required from platform developer.

- There is no need for specific details on the platform development class ADV.
- The user guidance (AGD) of the platform is considered early in the development of the application and the composite product and provides all interfaces information needed.
- The interfaces of the platform used by the application are evaluated and relied upon.
- All interfaces between platform and application are in the scope of the composite product evaluation.
- Test (ATE) and vulnerability assessment (AVA) are performed on the composite product taking advantage of platform evaluation results

Current concept of the Composite TOE evaluation does not limit the composite evaluation in EAL and resistance against attacks, i.e. up to ‘high’, whereas Composed TOE (CAP package) is limited by resistance against attacks ‘extended-basic’.

5.1.1.4 Objective and scope

The objective of this document is to precisely define tasks for the different parties involved in the composite product evaluation.

The aim is not to define an additional assurance class, but to define refinements to the existing assurance requirements for a composite product evaluation.

This document is also applicable to any other security IT technology, where an independently certified product is part of a final composite product to be evaluated

5.1.2 Definition / terminology

5.1.2.1 Definitions

A *composite product* is a product consisting of at least two different parts, whereby one of them represents a single product having already been evaluated and certified.

A *composite TOE* is defined in such a way that the already certified product is declared to be part of the composite TOE.

An evaluation of the composite TOE is a *composite evaluation*.

Usually a composite product consists of two components, whereby the first one represents an '*underlying platform*[1]' and the second one constitutes an '*application*' running on this platform. The underlying platform is the part of the composite product having already been certified (see below):

1

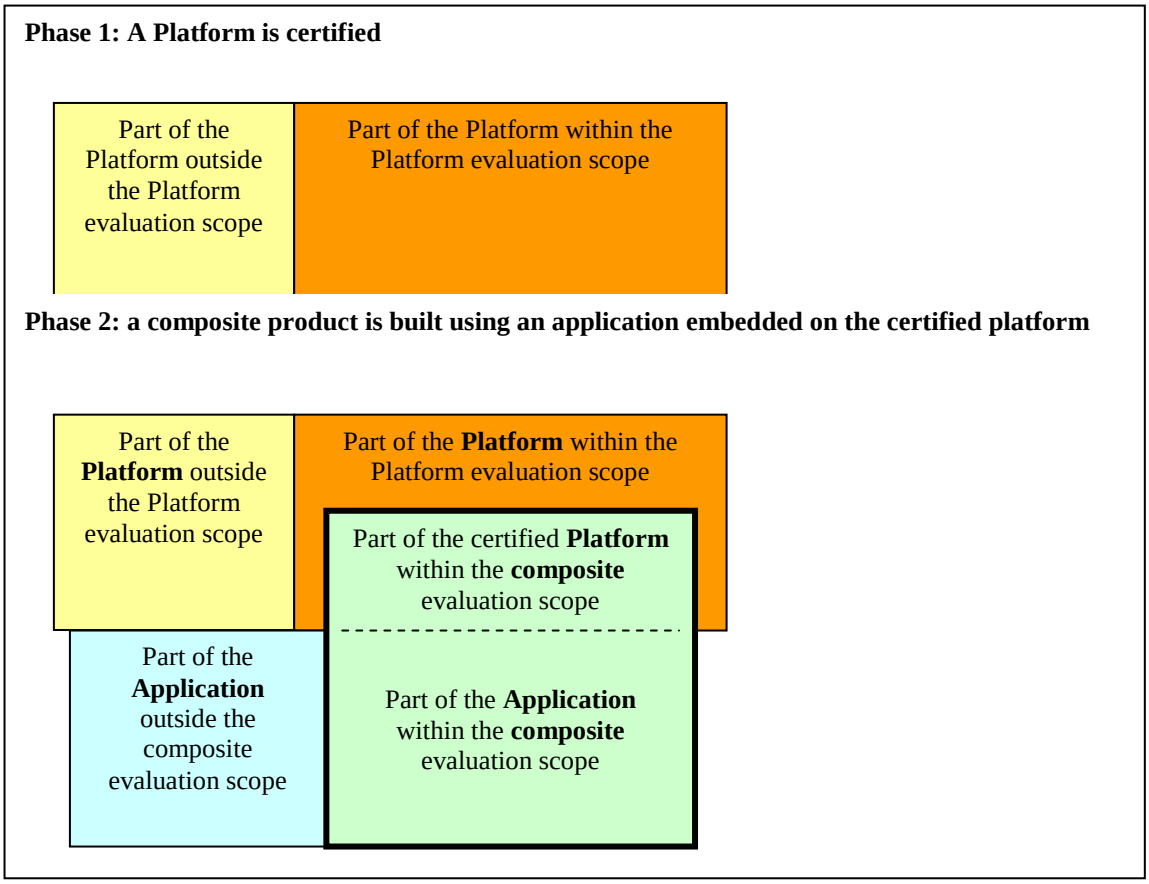


Diagram 3 Composite evaluation scope example

An 'underlying platform' usually plays a server role, whereby an 'application' – a client role. The server-client relation stresses here the constellation where the client uses already certified security services provided by the server.

[1] such a platform might also be called 'executing environment', 'abstract machine' or 'virtual machine'

Exemplifying, the following constellations can be mentioned: an application ('application') running in an executing environment ('underlying platform') like a mainframe or a personal computer operating system, or an operating system of a network device (e.g. routers, NAS, switches, modems) or of a smart meter or of an electronic control unit (ECU).

Further examples are applets ('application') running on an operating system ('underlying platform') of a smart phone or of another mobile device or Java applets ('application') running in a Java runtime environment ('underlying platform').

The next figure shows an example of a possible composition:

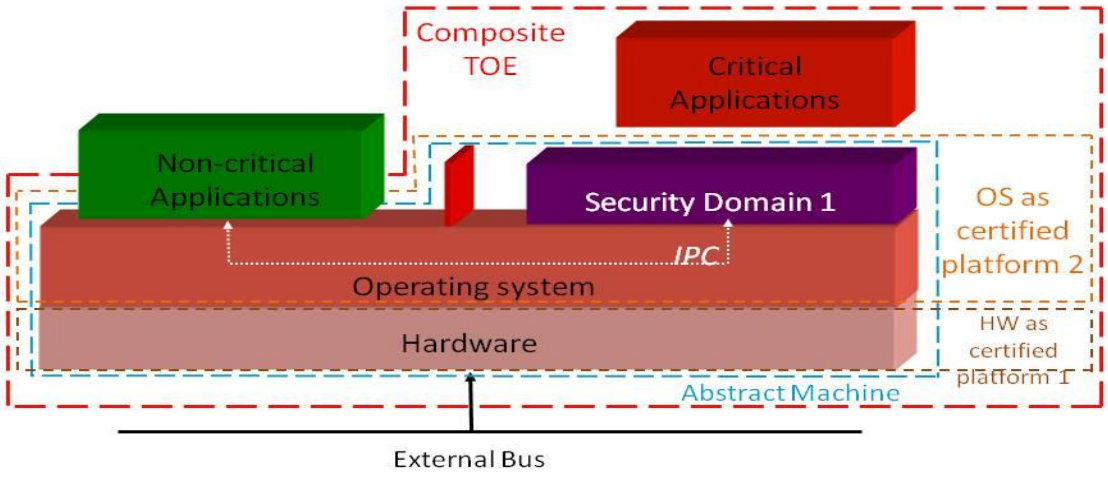


Diagram 4 Security composition example

In order to keep the document and terminology simple, we consider only the two component composite products and use the term 'platform' for the underlying certified product and the term 'application' for the software product running on the platform.

These definitions comply with ACO class definitions where:

- A platform is the base component,
- An application is the dependent component.

5.1.2.2 Roles

The following roles shall be considered in the composite evaluation activities:

- **Platform Developer**: Entity developing the platform; it might also be the sponsor of the platform evaluation.
- **Platform Evaluator**: Entity performing the platform evaluation.
- **Platform Certification Body**: Entity performing the platform certification, defined in CC V3 terminology as 'evaluation authority'.
- **Application Developer**: Entity developing the application running on the platform.
- **Composite Product Integrator** (synonym: System Integrator): Entity installing the applications on the platform.
- **Composite Product Evaluator**: Entity performing the composite product evaluation.

- **Composite Product Certification Body**: Entity performing the composite product certification, defined in CC V3 terminology as evaluation authority.
- **Composite Product Evaluation Sponsor**: Entity in charge of contracting the composite product evaluation.

Each evaluation shall associate particular organizations or persons to these generic roles.

In order to illustrate the role of the Composite Product Integrator (System Integrator) let us exemplify:

- Mainframe: The 'underlying platform' is a mainframe operating system and the Platform Developer is the related manufacturer; the 'application' is an application, e.g. a database management application and the Application Developer is the developer of the application. In this case, the role of the System Integrator is played by the operator of the mainframe system who installs the application onto the mainframe system.
- Java technology-enabled devices: The 'underlying platform' is a Java Runtime Environment (JRE) and the Platform Developer is the related manufacturer/issuer; the 'application' is a Java applet. In this case, the role of the System Integrator is often played by the manufacturer/issuer of the final device or by the operator of the final system, who installs the application onto the final device/system.
- Embedded devices like meters and ECUs: The 'underlying platform' is a hypervisor (virtualisation environment) and the Platform Developer is the related manufacturer/issuer; the 'application' is an application, in generally comprising executable code and data. An application can also be an operating system. The role of the System Integrator is often played by the manufacturer/issuer of the final device, who prepares/creates the final device binary image and possibly also installs this image on the destination hardware. The final device binary image comprises all the software components ('underlying platform', 'applications', drivers, etc.) necessary for the intended services of the embedded device.
- Smart phones and other mobile devices: The 'underlying platform' is a mobile device operating system and the Platform Developer is the related manufacturer/issuer; the 'application' is a mobile applet. In this case, the role of the System Integrator is often played by the manufacturer/issuer or by the operator/user of the final mobile device, who install the applet onto the final mobile device. If smart phones and other mobile devices would use in the near future a hypervisor (virtualisation environment) as the 'underlying platform', they can be treated like embedded devices above.

5.1.3 Composite evaluation concept

5.1.3.1 What are the issues?

The assets to be protected are the final composite product assets defined in the composite product Security Target.

The security functionalities involved in the protection of these assets are those provided by the platform and by the application itself.

Some of the security functionality provided by the platform may require a specific configuration, programming or activation by the application.

Therefore the **Application Developer** needs all the information (in form of a guidance or user's manual) related to the platform security functionality as well as to the platform common functionality the application has to manage, since the application may build its own security functionality using (but not security relying on) the functionality of the platform having not be considered as security one.

Furthermore, he needs the platform security target in order to build the composite product security target and to ensure consistency of security definition between both developments. Evaluation is performed and certified/validated on the final composite product.

The **Composite Product Evaluator** has to examine, whether the level of security required by the Security Target of the resulting composite product is achieved, when both parts are combined.

Therefore, the **Composite Product Evaluator** has to execute the evaluation tasks needed with respect to the Security Target of the final composite product and to provide the related ETR. In this perspective, the Composite Product Evaluator should reuse the platform's evaluation and certification result thus saving cost and time.

5.1.3.2 What information is needed?

The Composite Product Evaluator does not need all platform evaluation results. The security certificate and the certification report assure that the platform has been evaluated according to the Common Criteria. The Composite Product Evaluator will need complementary information on the assurance measures where both developments interfere. In addition to the standard amount of evaluation contributions according to the assurance package chosen for the composite evaluation (e.g. an EAL level), he will need the following (see section 'Deliveries' for further details):

- All the information delivered from the Platform Developer to the Composite Product Integrator,
- All the information delivered from the Platform Developer to the Application Developer,
- ETR for composite evaluation prepared by the Platform Evaluator, see section 'ETR for composite evaluation' (including information about vulnerability analysis and penetration testing),
- Information on compliance of the Security Targets and the designs of the platform and the application prepared by the Application Developer,
- Information on compliance of the delivery procedures of the Platform and Application Developers with the acceptance procedures of the Composite Product Integrator, and
- Information on integration of both parts using their correct certified versions and the correct configuration parameters. This information shall be prepared by the Composite Product Integrator.

Composite Product Certification Body will need the same amount of information as the Composite Product Evaluator.

5.1.3.3 Case of composite product change

In case of composite product changes due to a change of the platform or the application or both, please refer to [CC AC].

If a change comes from the platform, the assessment of the change for the platform is given by the **Platform Certification Body**. On this basis, the assessment of the change for the composite product is given by the **Composite Product Certification Body**.

If a change comes from the application, the assessment of the change for the composite product is given by the Composite Product Certification Body.

5.1.3.4 Specific case when the application is already certified

In the case where both platform and application have already been certified, a partial evaluation work may be performed regarding the results already obtained from previous application evaluation. Nevertheless, the composite evaluation tasks as defined in this document are still required.

5.1.4 Composite evaluation activities description

The current approach can be applied independent of the evaluation assurance level (EAL) for the composite product aimed. Where some evaluation activities are not applicable due to the EAL chosen, they are also not expected to be applied.

For the following paragraphs, we assume that the level of assurance of the platform is equivalent or higher compared to the composite product evaluation level.

Other cases must be discussed within the schemes.

The composite-specific developer and evaluator action elements as well as the evaluator actions (work units) belonging to the composition activities are defined as the refinements for composite evaluation, see Appendix 1: Composite-specific requirements.

5.1.4.1 Evaluation of the composite product Security Target

A Security Target for the composite product has to be written and evaluated.

The Composite Product Evaluator has to examine that the Security Target of the composite product does not contradict the Security Target of the underlying platform . In particular, this means that the Composite Product Evaluator has to examine the Composite- and the Platform- Security Targets for any conflicting elements of the security problem definitions, compatibility of security objectives, security requirements and security functionality needed by the application.

This task can be reduced, if some matching has already been checked for Protection Profiles claimed by each Security Target.

The Composite Product Evaluation Sponsor must ensure that the Platform-ST is available to the Application Developer, to the Composite Product Evaluator and to the Composite Product Certification Body. The information available in public version of the security target may not be sufficient.

5.1.4.2 Integration of the application in the configuration management system

The Composite Product Evaluator shall verify that the evaluated version of the application has been installed onto / embedded into the certified version of the underlying platform.

The Composite Product Evaluation Sponsor shall ensure that appropriate evidence generated by the Composite Product Integrator is available to the Composite Product Evaluator.

5.1.4.3 Compatibility check for delivery and acceptance procedures

The Composite Product Evaluator shall verify that delivery procedures of the Application and Platform Developers are compatible with the acceptance procedures used by the Composite Product Integrator.

The Composite Product Evaluator shall verify that all configuration parameters prescribed for the system integration process by the Application and Platform Developers are used by the Composite Product Integrator.

The Composite Product Evaluation Sponsor shall ensure that appropriate evidences generated by the Composite Product Integrator are available to the Composite Product Evaluator.

5.1.4.4 Compliance of designs

The Composite Product Evaluator shall verify that stipulations for the Application Developer imposed by the Platform Developer in its certified user guidance and referenced in the platform certification report are fulfilled by the composite product, i.e. have been taken into account by the Application Developer.

The Composite Product Evaluation Sponsor shall ensure that the following are made available to the Composite Product Evaluator:

- The platform-related user guidance,

- ETR for Composition prepared by the Platform Evaluator, see chapter 5 ‘ETR for composite evaluation’,
- The Certification Report for the platform prepared by the Platform Certification Body,
- A rationale for a secure composite product implementation including evidences prepared by the Application Developer.

5.1.4.5 Composite product functional testing

In most cases, the certified platform will be available for testing of the application. But some application functionality testing can only be performed in a specific environment deviating from the certified platform, before its embedding/integrating into the certified platform, as effectiveness of this testing (pass/fail) may not be visible using the interfaces of the final composite product. Such a specific environment may be represented by specifically instrumented versions of the platform or by a platform emulator and may require a rationale as to why this modified platform is an acceptable substitute for the certified platform with respect to the testing performed on it.

Nevertheless, functional testing of the composite product shall be performed also on composite product samples according to description of the security functions of the composite TOE and using the standard approach as required by the relevant assurance class.

The Composite Product Evaluator shall check the minimal amount of the testing necessary for the current composite evaluation having been performed on the composite product as a whole. That is what is called further in the document integration testing.

Since the amount, the coverage and the depth of the functional tests of the platform have already been validated by the platform certificate, it is not necessary to re-perform these tasks in the composite evaluation. Please note that ETR for Composition (see chapter 5 ‘ETR for composite evaluation’) does not provide any information on functional testing for the platform.

The Composite Product Evaluation Sponsor shall ensure that the following is available to the Composite Product Evaluator:

- specifically instrumented versions of the platform or/and a platform emulator and the applications in the scope of the composite evaluation suitable for functional testing, if necessary,
- composite product samples suitable for functional testing.

5.1.4.6 Composite product vulnerability analysis

The Composite Product Evaluator shall perform a vulnerability analysis for the composite product using, amongst other, the results of the platform evaluation and certification. This vulnerability analysis shall be confirmed by penetration testing.

In special cases, the vulnerability analysis and the definition of attack scenarios might be difficult, need considerable time and require extensive pre-testing, if only documentation is available. The platform may also be used in a way that was not foreseen by the Platform Developer and Platform Evaluator, or the Application Developer may not have followed the stipulations provided with the platform certification .

In such cases, the Composite Product Evaluator shall decide, whether the information available to him enables him performing vulnerability analysis as required by the assurance package chosen for the composite evaluation. The Composite Product Evaluator may use different non-mutually-excluding opportunities in order to enable or to shorten composite vulnerability analysis in such cases:

- The Composite Product Evaluator can consult the Platform Evaluator and draw on his experience gained during the platform evaluation.
- The Composite Product Evaluator can perform his own additional assurance activities to gain additional information on the platform needed for a proper vulnerability analysis of the

composite product. For example, he can load special testing software on the platform on his own discretion.

The Composite Product Evaluation Sponsor shall ensure that the following are made available to the Composite Product Evaluator:

- The ETR for Composition (ETR_COMP) prepared by the Platform Evaluator, see section 'ETR for composite evaluation' below,
- The Certification Report for the platform prepared by the Platform Certification Body,
- Composite product samples, incl. "open samples" ("open samples" are specifically instrumented versions of the platform, e.g. on which the Composite Product Evaluator can load software on his own discretion), if necessary, suitable for penetration testing

5.1.4.7 Deliveries

The tables below summarize the documentation deliveries exchanged between parties to enable the composite evaluation activities as defined in the previous paragraphs.

The Composite Product Evaluation Sponsor is in charge of the initialization of the process.

The Composite Product Evaluation Sponsor is responsible for maintaining or creating any Non Disclosure Agreement (NDA) that would be necessary between all the parties involved in the composition activities.

The Non Disclosure Agreement should be established according to the sensitivity and ownership of the information to be exchanged.

#	Document / Contribution	Description
1	Platform Security Target	Security Target of the platform as referenced in the platform certification report.
2	Platform open samples for testing, if requested by the **Composite Product Evaluator**	Specifically instrumented versions of the platform, e.g. on which the **Composite Product Evaluator** can load software on his own discretion.
3	Platform user guidance	It encompasses all platform user guidance and manuals needed for the **Application Developer** and the **Composite Product Integrator** as referenced in the platform certification report.
4	Platform ETR_COMP	ETR for composition as defined in chapter 5.1.5 and referenced in the platform certification report.
5	Platform certification report	Platform certification report issued by authorized **Platform Certification Bod**y.
6	Design compliance evidence	It enfolds evidence elements on how the requirements on the application design, imposed by the platform's guidance and certification report, are fulfilled in the composite product. If such a requirement was not followed, a rationale that the chosen composite product implementation is still secure shall be given here.

#	Document / Contribution	Description
7	Composite configuration evidence	It comprises (i) Identification elements of the composite product - proving that the correct, certified version of the platform is used in the composite product, - proving that the correct, evaluated version of the application has been integrated; and (ii) Evidence elements that configuration parameters prescribed by the **Platform** and **Application Developers** are actually being used by the **Composite Product Integrator**.
8	Delivery and acceptance procedures evidence	Evidence elements how the delivery procedures of the **Platform** and **Application Developers** are compatible with the acceptance procedure of the **Composite Product Integrator**

Table 10 Definition of specific composition contributions

The following table shows which contributions of Table 1 shall be provided to which actor within the composite evaluation process

		Actors				
#	**Documents/contributions having to be provided to**	**Composite product evaluation Sponsor**	**Composite product integrator (System Integrator)**	**Applica-tion developer**	**Composite product Evaluator**	**Composite product Certifica-tion Body**
1	Platform Security Target	No	No	Yes	Yes	Yes
2	Platform open samples[2]	No	No	No	Yes	No
3	Platform user guidance	No	Yes	Yes	Yes	Yes
4	Platform ETR_COMP	No	No	No	Yes	Yes
5	Platform certification report	Yes	Yes	Yes	Yes	Yes
6	Design compliance evidence	No	No	No	Yes	Yes
7	Composite configuration evidence	No	No	No	Yes	Yes

[2] Only if requested by composite product evaluator

		Actors				
#	**Documents/contributions having to be provided to**	**Composite product evaluation Sponsor**	**Composite product integrator (System Integrator)**	**Application developer**	**Composite product Evaluator**	**Composite product Certification Body**
8	Delivery and acceptance procedures evidence	No	No	No	Yes	Yes

Table 11 Main deliveries between actors

The next table shows some example of Composite TOE use cases with definition of the components and the roles

	Composite TOE example		
Components & roles definitions	**Mainframe** The composite TOE is built of - a mainframe operating system and an application, e.g. a data base management application, running in the executing environment of the OS.	**Embedded devices like meters and ECUs** as well as **smart phones and other mobile devices, if use a hypervisor** The composite TOE is built of - a hypervisor (virtualisation environment, separation kernel) - an application, in generally comprising executable code and data; an application can also be an operating system.	**Smart phones and other mobile devices without hypervisor** The composite TOE is built of - a mobile device operating system - a mobile applet
The **Platform** is	The mainframe operating system	The hypervisor (virtualisation environment, separation kernel)	The mobile device operating system
The **Application** is	Application(s) running on the OS, e.g. a data base management application	Application(s) running on the hypervisor, e.g. a smart meter application and Linux OS	The mobile applet
The **Platform Developer** is	The mainframe OS manufacturer: - develops, manufactures and delivers the mainframe OS to the System	The hypervisor Manufacturer: - develops, manufactures and delivers the hypervisor to the System Integrator	The mobile device OS manufacturer/issuer: - develops manufactures and delivers the mobile device OS to the

	Composite TOE example		
	Integrator		System Integrator
The **Application Developer** is	The application software developer: - develops the application; - delivers the application to the System Integrator	The application software developer: - develops the application; - delivers the application to the System Integrator	The applet developer: - develops the applet; - delivers the applet to the System Integrator
The **System Integrator** (**Composite Product Integrator)** is	The operator of the mainframe system: - is in charge of installing the application onto the mainframe system; - usually delivers the Composite TOE to be evaluated	The manufacturer/issuer of the final embedded device: - is in charge of preparing/creating the final device binary image; the latter comprises all the software components (the hypervisor, the smart meter application and the Linux OS, drivers, hardware abstraction layer, other necessary system components) and configuration data necessary for the intended services of the embedded device; - possibly also installs this image on the destination hardware. - usually delivers the Composite TOE to be evaluated	The manufacturer / issuer or the operator / user (less probably) of the final mobile device: - installs the applet onto the final mobile device; - delivers the Composite TOE to be evaluated.

Table 12 Example of composite TOE use cases

5.1.5 ETR for composite evaluation

5.1.5.1 Objective of the document

A standard Evaluation Technical Report (ETR) contains proprietary information that cannot be made public. The ETR for composite evaluation (ETR_COMP) document is compiled from the ETR in order to provide sufficient information for composite product evaluation with a certified platform. It should enable the Composite Product Evaluator and the respective Certification Body to understand the considered attack scenarios, the performed tests and the effectiveness of countermeasures implemented by the platform.

A template for an ETR_COMP document is given in Appendix 2: ETR for composite evaluation template.

5.1.5.2 Generic rules

The *ETR for composite evaluation* should be produced by the **Platform Evaluator** based on the platform evaluation results. This task should be considered when determining the evaluation work program to reduce additional cost and effort.

The content of ETR_COMP has to strike the right balance between protecting platform developer's and/or **Platform Evaluator's** proprietary information and providing sufficient information for the **Composite Product Evaluator** and the respective **Certification Body**, cf. Table 11 Main deliveries between actors above.

ETR_COMP shall not include information affecting national security.

The information provided must be approved by all parties involved in the platform evaluation (i.e. the Evaluator, the Certification Body, the developer and sponsor of the evaluation). The platform Certification Body shall validate its consistency with the original ETR. The platform certification report shall reference the *ETR for composite evaluation*.

If the current ETR_COMP itself relies on a previous composite evaluation, and if there is direct interface with the previous platform, the reference to this previous composite evaluation ETR_COMP must be supplied.

5.1.5.3 Content of the ETR for composite evaluation

The information required is focused on:

a) Formal information about the platform like its exact identification, reference to the certification report etc.

b) Information about the Platform design.

c) Information about the evaluated configuration of the Platform.

d) Information on delivery procedures and data exchange.

e) Information about vulnerability analysis for and penetration testing of the Platform including the considered attack scenarios and summary of test results. This information shall cover all attack scenarios and vulnerabilities have been considered as well as why they were considered non-exploitable.

f) Observations and recommendations for users.

5.1.5.3.1 Formal information

This section of ETR_COMP shall provide formal information on the platform evaluation as:

- product identification,
- sponsor and developer identities,
- identities of the evaluation facility and the certification body,
- assurance level of the evaluation,
- formal evaluation and certification results like pass/fail,
- references to the ETR.

5.1.5.3.2 Platform design

This section of ETR_COMP shall provide a high-level description of the IT product and its major components based on the deliverables required by the assurance class ADV of the Common Criteria. The intent of this section is to characterize the degree of architectural

separation of the major components and to show possible technical dependencies between the platform and an application using the platform.

5.1.5.3.3 Evaluated configuration

This section of ETR_COMP shall provide information about the evaluated configuration of the Platform based on the developer's configuration list or relevant parts as needed or on a case by case basis. The platform must unambiguously be identifiable and this identification shall be commensurate with the evaluated configuration as stated in the platform certification report.

Setting configuration parameters being security relevant for the Platform shall be explained and their effect on the resistance against considered attack scenarios be outlined (e.g. limits of platform resources).

Evidence about the evaluation of the configuration management coverage (ALC_CMC) can be necessary for a specific type of Platform, if it is relevant for supporting composite evaluations (e.g. evidence about integration of an application in the configuration management of a combined platform/application manufacturing). Therefore, beside the evaluation evidence about the principle capability of the configuration management system, specific composite configuration evidence may be necessary.

More precisely, as a platform may have several evaluated configurations, one dedicated configuration list for composition shall be delivered to the **Composite Product Evaluator**. With this document the evaluator should be able to make the link between the evaluated platform configuration items and the composite product configuration.

An example of a hypervisor platform and an application configuration list document is shown in the table below:

Hypervisor platform evaluated configurations list document
Identification of the TOE:
Pre-configured options/parameters (hardware abstraction layer, APIs):
Identification of already integrated applications, if any:
Binary file:
Additional information if any

Table 13 Configuration list example for hypervisor platform

5.1.5.3.4 Delivery procedures and data exchange

For supporting composite evaluation, evaluation evidence can be necessary for delivery of the platform to the **System Integrator**.
If applicable, additional evaluation evidence can be necessary for delivery and acceptance procedures of an application to be integrated during a combined platform/application manufacturing. Therefore, evaluation evidence about AGD_PRE[3] and ALC_DEL + AGD_PRE[4] might be relevant.

[3] [1.2C]

[4] [1.1C]

5.1.5.3.5 Penetration Testing

This section of ETR_COMP shall provide information about the independent vulnerability analysis performed by the **Platform Evaluator** with the attack scenarios having been considered, the penetration testing having been performed and the reference to the corresponding rating (quotation) of the attack potential.

Information about penetration testing should include details necessary for understanding the attack scenarios and the assessment of penetration results.

The attack scenarios description should provide sufficient details to reproduce attacks, which require additional countermeasures in the composite TOE.

In accordance with the requirements of CEM, this information is available within the ETR. So it can be compiled for ETR_COMP.

This section shall also address the rating of access to 'open samples' of the platform, if they exist (public/restricted/sensitive/critical). The availability of 'open samples' shall be considered in the assessment of the attack scenarios. Please note that 'open samples' are evaluation tools, but do not represent a TOE.

5.1.5.3.6 Observations and recommendations

The evaluated user guidance documentation shall contain all information required to use the TOE in a secure way as defined in the Platform-ST including recommendations on how to avoid residual vulnerabilities and unexpected behaviour.

However, in specific cases detailed information might be required in addition to the guidance documents such as:

- Observations on the evaluation results (e.g. specific TOE configuration for the evaluation),
- Recommendations/stipulations for the **Composite Product Evaluator**: specific information on use of the evaluation results (e.g. about specific testing necessary during a composition evaluation).

Any such observation or recommendation/stipulation may come from both the **Platform Evaluator** or the **Platform Certification Body.**

5.1.6 Evaluation/Certification reports and Platform certificate validity

Results of a composite evaluation shall be provided to the **Composite Product Certification Body** in form of an Evaluation Technical Report for the composite product. This Composite Product ETR shall contain, amongst others, the final overall verdict for the composite evaluation based on the partial verdicts for each assurance component being in scope of the current composite evaluation. There shall be a reference to this CC supporting document in the Composite Product ETR and the Composite Product Certification Report.

As the composite product certificate covers also the platform, the composite product certificate validity is linked to the validity of the platform certificate.

The **Composite Product Certification Body** needs an up-to-date certificate or an assessment from the **Platform Certification Body** on the status of the platform certificate in question.

As a general rule, the **Composite Product Certification Body** will ask for a reassessment of the platform, if the date of the platform's ETR for Composition is more than 18 months before the submission of the report containing the full results of the composition penetration tests. This reassessment consists of either a re-evaluation of the platform focussing on a renewal of the

vulnerability analysis (surveillance task) or, alternatively, a confirmation statement of the **Platform Certification Body** may be requested.

Note that in the case the entire composite product is set up as a chain of composite products constructed on top of each other (e.g. the platform itself is already a composite product), the maximum validity period of 18 months is related to the oldest ETR for Composition used in this chain of composite products. In addition, dependencies from a lower level ETR for Composition to a higher level ETR for Composition need to be considered when reusing the results in the composite evaluation on top.

Note also that if the platform's ETR for Composition was issued less than 18 months before submission of the related composite evaluation tasks, but there was a major change in the state of the art in performing relevant attacks on the platform (e.g. a major change in applicable attack methods or attack ratings), then the **Composite Product Certification Body** has the right to require a reassessment focusing on the new attack methods.

Validity and relevance of the platform certificate for the current composite product certification shall be acknowledged by the **Composite Product Certification Body** and includes the determination of equivalence of
- single assurance components (and, hence, of assurance levels) belonging to different CC versions, if the platform certification was according to another CC version/revision, than the current composite certification is, and
- the assurance packages used for the platform certification and chosen for the current composite product certification, if at least one of them is extended; it is applicable, even if these both assurance packages are derived from same CC version/revision. Such equivalence shall be established / acknowledged by the **Composite Product Certification Body**.

The **Composite Product Certification Body** can issue a security certificate for the composite product, if

- the verdict for the Composite Product ETR is PASS and
- validity and topicality of the platform certificate for the current composite product is acknowledged by the **Composite Product Certification Body**.

Note that, if the **Composite Product Evaluator** detects some failures resulting from Platform 'open samples' testing, the results are communicated to the **Composite Product Certification Body**. The **Composite Product Certification Body** shall then take appropriate steps together with the **Platform Certification Body**

5.1.7 References

5.1.7.1 CC V3.1 documents

[CC31] CCMB-2012-09-001 : Part 1 Introduction and general model, Version 3.1, Revision 4

CCMB-2012-09-002 : Part 2 Security Functional components, Version 3.1, Revision 4

CCMB-2012-09-003 : Part 3 Security Assurance Components, Version 3.1, Revision 4

[CEM31] CCMB-2012-09-004 : Evaluation Methodology, Version 3.1, Revision 4

5.1.7.2 Supporting documents

[JIL COMP SC] Joint Interpretation Library: Composite product evaluation for Smartcards and similar devices, v. 1.2, January 2012

[CCDB COMP SC] CCDB-2012-04-001 - Composite product evaluation for Smartcards and similar devices, v. 1.2, April 2012

[CC AC] CCIMB-2004-02-009 Assurance continuity: CCRA requirements

5.1.8 Appendix 1: Composite-specific requirements

In the following, the composite-specific developer and evaluator action elements as well as the evaluator actions (*work units*) belonging to the composition activities (cf. chapter 5.1.4 above) are defined. They require the evidences as listed in section 5.1.4.7.

These refinements to the assurance requirements aim to give the **Composite Product Developer** and **Evaluator** a precise guidance on which relevant aspects have to be described and assessed in the context of a composite evaluation and the tasks to be performed.

It allows the **Composite Product Certification Body** to check using the composite product ETR that the required (mandatory) tasks have properly been performed.

All composite-specific evaluator actions have to be documented according to the scheme rules and finalised by one of the verdicts PASS, FAIL or INCONCLUSIVE. As these actions are refinements of the traditional actions focused on the composition activities, these verdicts have to be integrated to the overall verdict.

This approach can be applied independent of the aimed evaluation assurance level (EAL) for the composite product. Where some evaluation activities are not applicable due to the EAL chosen, the related composite-specific tasks are also not expected to be performed.

The composite evaluation methodology described by the current document is in principle independent of the CC version (v2.x or v3.1). However, due to the fact that CC v2.x is hardly still in use, the composite-specific tasks below are defined only for CC v3.1 in section 5.1.8.1 independent of a concrete revision.

For convenience of composite-specific activities and associated work units identification, each refinement is named as *_COMP, where * is the name of the assurance class it is related to.

5.1.8.1 Composite-specific evaluation tasks in CC V3.1

5.1.8.1.1 Consistency of composite product Security Target (ASE_COMP)

The composite-specific work units defined in this chapter are intended to be integrated as refinements to the evaluation activities of the ASE class listed in the following table. The other activities of ASE class do not require composite-specific work units.

CC assurance family	Evaluation activity	Evaluation work unit	Composite-specific work unit
ASE_OBJ	ASE_OBJ.2.1C	ASE_OBJ.2-1	ASE_COMP.1-5
	ASE_OBJ.2.1C	ASE_OBJ.2-1	ASE_COMP.1-8
	ASE_OBJ.2.3C	ASE_OBJ.2-3	ASE_COMP.1-8
ASE_REQ	ASE_REQ.1.6C	ASE_REQ.1-10	ASE_COMP.1-1
	ASE_REQ.2.9C.	ASE_REQ.2-13	ASE_COMP.1-1
	ASE_REQ.1.6C	ASE_REQ.1-10	ASE_COMP.1-2
	ASE_REQ.2.9C.	ASE_REQ.2-13	ASE_COMP.1-2
	ASE_REQ.2.8C	ASE_REQ.2-12	ASE_COMP.1-3
	ASE_REQ.2.3C	ASE_REQ.2-4	ASE_COMP.1-4
ASE_SPD	ASE_SPD.1.1C	ASE_SPD.1-1	ASE_COMP.1-6
	ASE_SPD.1.3C	ASE_SPD.1-3	ASE_COMP.1-6
	ASE_SPD.1.4C	ASE_SPD.1-4	ASE_COMP.1-7

NB: If the level of the assurance requirement chosen is higher than those identified in this table, the related composite-specific work unit is also to apply.

ASE_COMP.1 Consistency of Security Target

Objectives

The aim of this activity is to determine whether the Security Target of the composite product[5] does not contradict the Security Target of the underlying platform[6].

Application notes

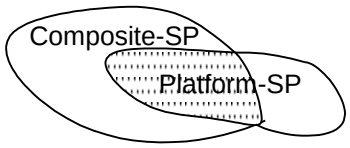
These application notes aid the developer to create as well as the evaluator to analyze a composite Security Target and describe a general methodology for it. For detailed information / guidance please refer to the single work units below.

In order to create a composite Security Target the developer should perform the following steps:

Step 1: The developer formulates a preliminary Security Target for the composite product (the Composite-ST) using the standard code of practice. The Composite-ST can be formulated independently of the Security Target of the underlying platform (Platform-ST) – at least as long as there are no formal PP conformance claims.

In order to be able to maintain specific security domains for critical applications (i.e. those that may be included or already are in scope of a security evaluation), an effective separation of such security domains is necessary. It is expected that such domain separation mechanisms will be provided and enforced to the greatest extent by the underlying platform. A software platform can provide such a domain separation in a *logical* way probably using some helpful properties of the underlying hardware.
While choosing an appropriate underlying platform (analysing Platform-STs) and formulating the Composite-ST, it is important to pay attention that the domain separation property is modelled by a set of concrete security functionalities, i.e. by a set of appropriate SFRs, but not merely as a pure 'architectural' property in ADV_ARC.

Step 2: The developer determines the intersection of the Composite-ST and the Platform-ST analysing and comparing their TOE Security Functionality (TSF) [7][8]:



[5] denoted by Composite-ST in the following

[6] denoted by Platform-ST in the following. Generally, a Security Target expresses a security policy for the TOE defined.

[7] because the TSF enforce the Security Target (together with organisational measures enforcing security objectives for the operational environment of the TOE).

[8] The comparison shall be performed on the abstraction level of SFRs. If the developer defined security functionality groups (TSF-groups) in the TSS part of his Security Target, the evaluator should also consider them in order to get a better understanding for the context of the security services offered by the TOE.

Step 3: The developer determines under which conditions he can trust in and rely on the Platform-TSF being used by the Composite-ST without a new examination.

Having undertaken these steps the developer completes the preliminary Security Target for the composite product.

It is not mandatory that the platform has been and the composite TOE is being certified according to same version/revision of the CC. It is due to the fact that the application can rely on some security services of the platform, if (i) the assurance level of the platform covers the intended assurance level of the composite TOE and (ii) the platform's security certificate is valid and up-to-date. Equivalence of single assurance components (and, hence, of assurance levels) belonging to different CC versions/revisions shall be established / acknowledged by the Composite Product Certification Body, cf. chapter 5.1.6.

If a PP conformance is claimed (e.g. the Composite-ST claims conformance to a PP that claims conformance to a PP for the platform), the consistency check can be reduced to the elements of the Security Target having not already been covered by these Protection Profiles.

The fact of compliance to a PP is not sufficient to avoid inconsistencies. Assume the following situation, where → stands for "complies with":

Composite-ST → Application-PP → Platform-PP ← platform-ST

The Application-PP may require any kind of conformance[9], but this does not change the 'additional elements' that the Platform-ST may introduce to the Platform-PP. In conclusion, these additions are not necessarily consistent with the Composite-ST additions to the Application-PP: There is no scenario that ensures the consistency 'by construction'.

Note that consistency may be not directly matching: e.g. objectives for the platform environment may become objectives for the composite TOE.

Dependencies:

No dependencies.

Developer action elements:

ASE_COMP.1.1D

The developer shall provide a statement of compatibility between the Composite Security Target and the Platform Security Target. This statement can be provided within the Composite Security Target.

Content and presentation of evidence elements:

ASE_COMP.1.1C

The statement of compatibility shall describe the separation of the Platform-TSF into relevant Platform-TSF being used by the Composite-ST and others.

ASE_COMP.1.2C

The statement of compatibility shall show (e.g. in form of a mapping) that the Security Targets for the composite product and for the underlying platform match, i.e. that there is no conflict between the security problem definitions, security objectives, and security requirements of the Composite-ST and the Platform-ST. It can be provided by indicating of the concerned elements directly in the Composite-ST followed by explanatory text, if necessary.

[9] e.g. "strict" or "demonstrable" according to CC V3.1.

Evaluator action elements:

ASE_COMP.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

Evaluator actions:

Action ASE_COMP.1.1E

ASE_COMP.1.1C

ASE_COMP.1-1 The evaluator ***shall check*** that the statement of compatibility describes the separation of the Platform-TSF into relevant Platform-TSF being used by the Composite-ST and others.

Please note that TSF means 'TOE Security Functionality' in CC V3.1, whereby the TSF content is represented by SFRs[10]. The respective TOE summary specification (TSS) shall provide, for each SFR, a description on how each SFR is met[11]. The evaluator shall use this description in order to understand the contextual frame of the SFRs.
If the developer defined security functionality groups (TSF-groups) in the TSS part of his Security Target as such contextual frame of the SFRs, the evaluator should also consider them in order to get a better understanding for the context of the security services offered by the TOE.

This work unit relates to the *Step 2* of the *Application Notes* above. In order to determine the intersection area the evaluator considers the list of the Platform-SFRs (given in the Platform-ST) as the single properties of the platform's security services.
To give an example, let us assume that there are the following Platform-SFRs: No illicit information flaws FDP_IFF.5, automated recovery FPT_RCV.2, failure handling FPT_FLS.1 as well as resource allocation FRU_RSA.2.

These Platform-SFRs shall be separated in two groups:

– **IP_SFR**: Irrelevant Platform-SFRs not being used by the Composite-ST, and

– **RP_SFR**: Relevant Platform-SFRs being used by the Composite-ST.

The second group *RP_SFR* exactly represents the intersection area in question. For example, *IP_SFR* = {FPT_RCV.2} and *RP_SFR* = {FDP_IFF.5, FPT_FLS.1, FRU_RSA.2}, i.e. the automated recovery service of the platform is not used by the composite TOE, but all other Platform-SFRs are used.

The amount of the intersection area (i.e. the content of the group *RP_SFR*) results from the concrete properties of the Platform-ST and the Composite-ST. If the Composite-ST does not use any property of the Platform-ST and, hence, the intersection area is an empty set (*RP_SFR* = {∅}), no further composite evaluation activities are necessary at all: In such a case there is a purely functional, but not a security composition.

The result of this work unit shall be integrated to the result of ASE_REQ.1.6C / ASE_REQ.1-10 (or the equivalent higher components, if a higher assurance level is selected) and ASE_REQ.2.9C / ASE_REQ.2-13.

[10] security functional requirements

[11] cf. CC part 3, ASE_TSS.1.1C

ASE_COMP.1-2 The evaluator ***shall examine*** the statement of compatibility to determine that the Platform-TSF being used by the Composite-ST is complete and consistent for the current composite TOE.

In order to determine the completeness of the list of the Platform-TSF being used by the Composite-ST, the evaluator shall verify that:

– Platform-SFR = IP_SFR ∪ RP_SFR and

– elements belonging to RP_SFR actually reflect the composite TOE to the intended extent.

In order to determine the consistency of the list of the Platform-TSF being used by the Composite-ST, the evaluator shall verify that there are no ambiguities and contradictory statements.

More details on the consistency analysis can be found in common CC documents.

The result of this work unit shall be integrated to the result of ASE_REQ.1.6C / ASE_REQ.1-10 (or the equivalent higher components, if a higher assurance level is selected) and ASE_REQ.2.9C / ASE_REQ.2-13.

ASE_COMP.1.2C

ASE_COMP.1-3 The evaluator ***shall check*** that the security assurance requirements of the composite evaluation represent a subset of the security assurance requirements of the underlying platform.

This work unit relates to the *Step 2* of the *Application Notes* above. In order to ensure a sufficient degree of trustworthiness of the Platform-TSF the evaluator compares the TOE security assurance requirements[12] of the composite evaluation with those of the underlying platform. The evaluator decides that the degree of trustworthiness of the Platform-TSF is sufficient, if the Composite-SAR represent a subset of the Platform-SAR:

$$\text{Platform-SAR} \supseteq \text{Composite-SAR},$$

e.g. the EAL chosen for the composite evaluation does not exceed the EAL applied to the evaluation of the platform.

The result of this work unit shall be integrated to the result of ASE_REQ.2.8C / ASE_REQ.2-12.

ASE_COMP.1-4 The evaluator ***shall examine*** the statement of compatibility to determine that all performed operations on the relevant TOE security functional requirements of the platform are appropriate for the Composite-ST.

This work unit relates to the *Step 3* of the *Application Notes* above. The *relevant* TOE security functional requirements of the platform are exactly the elements of the group *RP_SFR* (cf. the work unit ASE_COMP.1-1).

In order to perform this work unit the evaluator compares single parameters of the *relevant* SFRs of the platform with those of the composite evaluation. For example, the evaluator compares the properties of the respective components FRU_RSA.2/communication_objects and determines that the Composite-ST requires the maximal amount of physical memory that can be allocated to the communication object and the Platform-ST can enforce this security service

[12] denoted by SAR in the following

with a configurable physical memory range from 1/64 to the maximum. Hence, this parameter of the platform is appropriate for the Composite-ST.
Note, that the Composite-SFRs need not necessarily be the same as the Platform-SFRs, e.g. a trusted channel (FTP_ITC.1) in the composite product can be built using data exchange integrity (FDP_UIT.1) and data exchange confidentiality (FDP_UCT.1) of the platform.

The result of this work unit shall be integrated to the result of ASE_REQ.2.3C / ASE_REQ.2-4.

ASE_COMP.1-5 The evaluator ***shall examine*** the statement of compatibility to determine that the relevant TOE security objectives of the Platform-ST are not contradictory to those of the Composite-ST.

This work unit relates to the *Step 3* of the *Application Notes* above. The *relevant* TOE security objectives of the Platform-ST are those that are mapped to the *relevant* SFRs of the Platform-ST (cf. the work unit ASE_COMP.1-4).

In order to perform this work unit the evaluator compares the *relevant* TOE security objectives of the Platform-ST with those of the Composite-ST and determines whether they are not contradictory.

The result of this work unit shall be integrated to the result of ASE_OBJ.2.1C / ASE_OBJ.2-1.

ASE_COMP.1-6 The evaluator ***shall examine*** the statement of compatibility to determine that the relevant elements of the security problem definition of the Platform-ST - threats and OSPs - are not contradictory, also crosswise, to those of the Composite-ST.

This work unit relates to the *Step 3* of the *Application Notes* above. The evaluator compares the *relevant* elements of the security problem definition (i.e. threats and OSPs being mapped to the relevant TOE security objectives, cf. the work unit ASE_COMP.1-5) of the Platform-ST with those of the Composite-ST and determines, whether they are not contradictory, also crosswise. Crosswise means relevant Platform-threats vs. Composite-OSPs and vice versa.
The evaluator can decide on non-contradiction, if the threats and OSPs of the Composite-ST referring to the platform-share of the composite product are covered, also crosswise, by the *relevant* threats/OSPs of the Platform-ST. For example, there may be a threat T.Forgery in the Composite-ST covered by the threat T.Modification in the Platform-ST.

An example for contradictive items: The organisational security policy of the Platform-ST “Cryptographic algorithms used shall be in accordance with *international standards*” is contradictory to the threat of the Composite-ST “An attacker discloses the secrets being used by the TOE *proprietary* cryptographic algorithm”.

Beyond it, a special organisational security policy OSP.Composite could be formulated within the Composite-ST, e.g. ‘The application is running on a certified platform and compatible with it’. Then the developer can define the special security objectives for the TOE and its environment exactly reflecting the conditions and restrictions of the certification report of the underlying platform.

The result of this work unit shall be integrated to the result of ASE_SPD.1.1C / ASE_SPD.1-1 and ASE_SPD.1.3C / ASE_SPD.1-3.

ASE_COMP.1-7 The evaluator ***shall examine*** the statement of compatibility to determine that the environment-upheld elements of the security problem definition

(assumptions, OSPs, threats) of the Platform-ST being significant for the Composite-ST are complete and consistent for the current composite TOE.

This work unit relates to the *Step 3* of the *Application Notes* above. Environment-upheld elements of the security problem definition are those being upheld by the platform operational environment, i.e. by the environmental security objectives of the platform. The environment-upheld elements of the security problem definition include all the assumptions as well as OSPs and threats assigned to the environmental security objectives of the platform.
In order to determine which environment-upheld elements of the security problem definition of the Platform-ST are *significant* for the Composite-ST the evaluator analyses them and their separation in the following groups:

– **IrPA**: The environment-upheld elements being not relevant for the Composite-ST, e.g. the assumptions about the developing and manufacturing phases of the platform.

– **CfPA**: The environment-upheld elements being fulfilled by the Composite-ST *automatically*.
Such environment-upheld elements of the Platform-ST can always be assigned to the TOE security objectives of the Composite-ST. Due to this fact they will be fulfilled either by the Composite-SFR or by the Composite-SAR automatically.
To give an example, let there be an assumption A.Resp-Appl of the Platform-ST: 'Any user partition content is owned by user applications residing in this partition. Therefore, it must be assumed that security relevant user data are treated by the user applications as defined for the specific application context' and a TOE security objective OT.Key_Secrecy of the Composite-ST: 'The secrecy of the signature private key used for signature generation is reasonably assured against attacks with a high attack potential.' If the private key is the only sensitive data element of the user application, then the Platform-assumption A.Resp-Appl is covered by the Composite-objective OT.Key_Secrecy automatically.

– **SgPA**: The remaining environment-upheld elements of the Platform-ST pertain neither to the group *IrPA* nor *CfPA*. **Exactly this group makes up the *significant* environment-upheld elements of the security problem definition (assumptions, OSPs, threats) for the Composite-ST**, i.e. that shall also be included into the environmental branch of the Composite-ST.

The result of this work unit shall be integrated to the result of ASE_SPD.1.4C / ASE_SPD.1-4.

ASE_COMP.1-8 The evaluator ***shall examine*** the statement of compatibility to determine that the significant security objectives for the operational environment of the Platform-ST are not contradictory to those of the Composite-ST.

This work unit relates to the *Step 3* of the *Application Notes* above. The *significant* security objectives for the operational environment of the Platform-ST are those being assigned to the elements of the security problem definition (assumptions, OSPs, threats) classified as the group *SgPA* of the Platform-ST (cf. the work unit ASE_COMP.1-7).

In order to accomplish this work unit the evaluator compares the *significant* security objectives for the operational environment of the Platform-ST with those of the Composite-ST and determines whether they are not contradictory.

If necessary, the *significant* security objectives for the operational environment of the Platform-ST shall be included into the Composite-ST and assigned to the elements of the security problem definition (assumptions, OSPs, threats) from the group *SgPA*, cf. the work unit ASE_COMP.1-7. The inclusion is not

necessary, if the Composite-ST already contains equivalent (or similar) security objectives (covering all the relevant aspects).

Since assurance of the development and manufacturing environment of the platform is confirmed by the platform certificate, the respective platform-objectives, if any, belong to the group *IrPA*.

Assurance of development and manufacturing environment is usually completely addressed by the assurance class ALC, and, hence, requires no explicit security objective.

The result of this work unit shall be integrated to the result of ASE_OBJ.2.1C / ASE_OBJ.2-1 and ASE_OBJ.2.3C / ASE_OBJ.2-3.

5.1.8.1.2 Integration of composition parts and Consistency of delivery procedures (ALC_COMP)

The composite-specific work units defined in this chapter are intended to be integrated as refinements to the evaluation activities of the ALC class listed in the following table. The other activities of ALC class do not require composite-specific work units.

CC assurance family	Evaluation activity	Evaluation work unit	Composite-specific work unit
ALC_CMS	ALC_CMS.12C	ALC_CMS.1-2	ALC_COMP.1-1
ALC_DEL	ALC_DEL.1.1C	ALC_DEL.1-1	ALC_COMP.1-3
AGD_PRE	AGD_PRE.1.2C	AGD_PRE.1-4	ALC_COMP.1-2

NB: If the level of the assurance requirement chosen is higher than those identified in this table, the related composite-specific work unit is also to apply.

ALC_COMP.1 **Integration of the application into the underlying platform and Consistency check for delivery and acceptance procedures**

Objectives

The aims of this activity are to determine, whether

– the correct version of the application is installed onto/into the correct version of the underlying platform, and

– the delivery procedures of **Platform** and **Application Developers** are compatible with the acceptance procedure of the **Composite Product Integrator**.

Application notes

These application notes serves to stress the importance of
- the specific role and responsibility of the **Composite Product Integrator** (**System Integrator**) for maintaining security properties of the Composite-TOE and, hence, for enforcing the security policy defined in the Composite-ST during the operation of the Composite-TOE later on, and
- the *documental* interface between the underlying platform and applications. Note that the *functional* interface between the underlying platform and applications is subject to the specific activities within ADV_COMP.1 and ATE_COMP.1 below.

The **System Integrator** (= **Composite Product Integrator**) plays an important role in the life cycle of the Composite-TOE:

- 1) he decides on the immediate TOE operational environment, in which the Composite-TOE will be executed, and on applications to be integrated into the Composite-TOE,
- 2) he (pre- and post-)configures and composes (integrates) the Composite-TOE.

In a first step – selection of the immediate TOE operational environment – the **System Integrator** selects hardware, and if applicable, firmware and bootloader the Composite-TOE shall run on.
He selects all the necessary privileged components (drivers, management applications, etc.) in order to make the Composite-TOE and the final system executable in its operational environment and manageable.

Because of possible special requirements on the hardware by the platform of the Composite-TOE (e.g. on a hardware-controlled memory management) and special capabilities of the privileged components, the **System Integrator** has to select them carefully and to approve them for a secure integration into the Composite-TOE.

The **System Integrator** also selects all the (user) applications intended to be integrated into the Composite-TOE. The content of any user application may usually be arbitrary and not require any *security* approval by the **System Integrator**.

In a second step – dependent on the concrete type of the Composite-TOE and its integration capabilities – the **System Integrator** may either
- pre-configure the Composite-TOE and then create the final system as a dedicated binary image (rather a case for a hypervisor as platform) or
- install all the applications on the platform and then perform the post-configuration task (rather a case of mainframe platform and devices with an OS-own capability for the installation of applications).

Pre- and post-configuration tasks encompass setting all the parameters as prescribed by the **Platform** and **Application Developers** and in accordance with a **System Integrator** own *System Security Policy*.

It is expected that the system integration procedure having to be performed by the **System Integrator** is completely and accurately described and evaluated within the assurance component AGD_PRE.1 [AGD_PRE.1.2C].

It is also expected that the *documental* interface between the underlying platform and applications (platform and application user guidance, ETR_COMP) completely and accurately reflects their security-relevant interaction according to the requirements of the assurance class AGD.
The amount of security (and, if necessary, functional) services provided by platform to applications and by the applications to users determines the width of the documental interface.

Hence, it is expected, especially from the **Platform Developer**, providing a consolidated 'Security Guidance' (within the assurance class AGD) comprising all the CC evaluation relevant requirements on applications.

Since the standard assurance elements of the class AGD completely cover all the aspects addressed in this Application Note, there is no necessity for any additional composite activities / refinements for the assurance class AGD.

Dependencies:

No dependencies.

Developer action elements:

ALC_COMP.1.1D

The developer shall provide components configuration evidence; cf. item #7 in Table 10 Definition of specific composition contributions, section 5.1.4.7.

ALC_COMP.1.2D

The developer shall provide an evidence for delivery and acceptance compatibility; cf. item #8 in Table 10 Definition of specific composition contributions, section 5.1.4.7.

Content and presentation of evidence elements:

ALC_COMP.1.1C

The components configuration evidence shall show that

(i) the evaluated version of the application has been installed onto / embedded into the certified version of the underlying platform and

(ii) the configuration parameters evidence shall show that configuration parameters prescribed by the Platform and Application Developers are actually being used by the Composite Product Integrator.

ALC_COMP.1.2C

The evidence for delivery and acceptance compatibility shall show that the delivery procedures of the Platform and Application Developers are compatible with the acceptance procedure of the Composite Product Integrator.

Evaluator action elements:

ALC_COMP.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

ALC_COMP.1.2E

The evaluator shall confirm that the evidence for delivery compatibility is complete, coherent, and internally consistent.

Evaluator actions:

Action ALC_COMP.1.1E

ALC_COMP.1-1 The evaluator ***shall check*** the evidence that the evaluated version of the application has been installed onto / embedded into the correct, certified version of the underlying platform.

The *general* information on the CM capabilities is represented and has to be examined in the context of the assurance family ALC_CMC.
The *special* composite evaluator activity is to check the evidence of the version correctness for both parts of the composite product.

For the underlying platform, the evaluator shall determine that the actual identification of the platform is commensurate with the respective data in the platform security certificate.

For the application(s), the relevant task is trivial due to the fact that the **Composite Product Evaluator** has to perform this task in the context of the assurance family ALC_CMC.

Components identification evidence can be supplied in two different ways: *technical* and *organisational*. A technical evidence of version correctness is being generated by the composite product itself: the platform and the application return – in each case – strings containing an unambiguous version reference as answers to the respective commands. E.g. it can be the return string of a command or the hard copy of the GUI-information (like 'About').

If also hardware is in scope (e.g. as part of the platform), a technical evidence of version correctness for hardware can also be supplied, if applicable, by reading off an unambiguous inscription on its surface or by triggering an answer from fixed code stored in the hardware by its manufacturer.

Technical evidence is recommended to be provided.

An organisational evidence of version correctness is being generated by the **Composite Product Integrator** on the basis of his configuration lists containing unambiguous version information of the platform and the application(s) having been composed into the final composite product.

For example, it can be acknowledgement statements (e.g. configuration lists) of the platform and application(s) manufacturers to the **Composite Product Integrator** containing the evidence for the versions of the platform, the application(s) and their pre-configured options / parameters[13].

Organisational evidence is always possible and, hence, shall be provided.

The result of this work unit shall be integrated to the result of ALC_CMS.1.2C / ALC_CMS.1-2 (or the equivalent higher components if a higher assurance level is selected).

ALC_COMP.1-2

The evaluator ***shall examine*** the evidence for using configuration parameters to determine that the Composite Product Integrator uses the configuration parameters as prescribed by the Platform and Application Developers.

The *general* information on the configuration parameters required is represented and has to be examined in the context of the assurance family AGD_PRE [1.2C].
The *special* evaluator activity is to examine the developer's evidence and to decide whether the **Composite Product Integrator** appropriately treats this *special subset* of the configuration parameters.

For example, for an embedded or mobile device as composite TOE, the manufacturer/issuer of the final device has to set all parameters as prescribed by the Platform- and the Applet- Developers while installing the applet onto the platform; cf.Table 12 Example of composite TOE use cases, section 5.1.4.7.

The result of this work unit shall be integrated to the result of AGD_PRE.1.2C /AGD_PRE.1-4.

Action ALC_COMP.1.2E

The evaluator ***shall examine*** the evidence for compatibility of delivery interfaces to determine that delivery procedures of the Platform and Application Developers are compatible with the acceptance procedure of the Composite Product Integrator.

The *general* information on the delivery procedures is represented and has to be examined in the context of the assurance families ALC_DEL and AGD_PRE [1.1C].
The *additional* composite activity of the evaluator is to examine each delivery interface between the **Platform Developer** and the **Composite Product Integrator** on the one side and between the **Application Developer** and the

[13] e.g. pre-configured hardware abstraction layer and APIs of the platform

Composite Product Integrator on the other side. As a result, the evaluator confirms or disproves the justification for delivery compatibility.

If there are no delivery interfaces between the **Platform** and **Application Developers** and the **Composite Product Integrator** or the assurance package chosen does not contain the family ALC_DEL (e.g. EAL1), this work unit is not applicable.

The result of this work unit shall be integrated to the result of ALC_DEL.1.1C / ALC_DEL.1-1.

5.1.8.1.3 Composite design compliance (ADV_COMP)

The composite-specific work units defined in this chapter are intended to be integrated as refinements to the evaluation activities of the ADV class listed in the following table. The other activities of ADV class do not require composite-specific work units.

CC assurance family	Evaluation activity	Evaluation work unit	Composite-specific work unit
ADV_ARC	ADV_ARC.1.1E	ADV_ARC.1.1C/ ADV_ARC.1-1	ADV_COMP.1-1
ADV_IMP	ADV_IMP.1.1E	ADV_IMP.1.1C/ ADV_IMP.1-1	ADV_COMP.1-1
ADV_INT	ADV_INT.2.1E	ADV_INT.2.1C/ ADV_INT.2-1	ADV_COMP.1-2
ADV_TDS	ADV_TDS.1.2E	ADV_TDS.1-7	ADV_COMP.1-1

NB: If the level of the assurance requirement chosen is higher than those identified in this table, the composite-specific work unit is also to apply.

ADV_COMP.1 **Design compliance with the platform certification report, guidance and ETR_COMP**

Objectives

The aim of this activity is to determine whether the requirements on the application, imposed by the underlying platform, are fulfilled in the Composite-TOE.

Application notes

The requirements on the application, imposed by the underlying platform, can be formulated in the relevant certification report (e.g. in form of constraints and recommendations), user guidance and ETR_COMP (in form of observations and recommendations) for the platform. The manufacturer/issuer of the composite product shall regard each of these sources, if available (cf. Table 11 Main deliveries between actors, 5.1.4.7), and implement the composite product in such a way that the applicable requirements are fulfilled.

The TSF of the composite product is represented at various levels of abstraction in the families of the development class ADV. Experiential, the appropriate levels of design representation for examining, whether the requirements of the platform are fulfilled by the composite product, are the TOE design (ADV_TDS), security architecture (ADV_ARC) and the implementation (ADV_IMP). In case, these design representation levels are not available (e.g. due to the assurance package chosen is EAL1), the current activity is not applicable (see the next paragraph for the reason).

Due to the definition of the composite TOE (cf. section 5.1.2.1') the interface between the underlying platform and the application is the *internal* one, hence, a functional specification (ADV_FSP) as representation level is not appropriate for analysing the design compliance.

Security architecture ADV_ARC as assurance family is dedicated to ensure that integrative security services like domain separation, self-protection and non-bypassability properly work. It is impossible and not the sense of the composite evaluation to have an insight into the architectural internals of the underlying platform (it is a matter of the platform evaluation). What the **Composite Evaluator** has to do in the context of ADV_ARC is
(i) to determine, whether the application uses services of the underlying platform within its own Composite-ST to provide domain separation, self-protection, non-bypassability and protected start-up; if no, there is no further composite activities for ADV_ARC; if yes, then
(ii) the evaluator has to determine, whether the application uses these platform-services in an appropriate/secure way (please refer to the platform user guidance, cf. item #3 in Table 10 Definition of specific composition contributions, section 5.1.4.7.).

Since consistency of the composite product security policy has already been considered in the context of the Security Target in the assurance family ASE_COMP (see page 122 above), there is no compulsory necessity to consider non-contradictoriness of the security policy model (ADV_SPM) of the Composite-TOE and the security policy model of the underlying platform.

Dependencies:

No dependencies.

Developer action elements:

ADV_COMP.1.1D

The developer shall provide a design compliance justification; cf. item #6 as well as items #3, #4, #5 Table 10 Definition of specific composition contributions, section 5.1.4.7.

Content and presentation of evidence elements:

ADV_COMP.1.1C

The design compliance justification shall provide a rationale for design compliance – on an appropriate representation level – of how the requirements on the application, imposed by the underlying platform, are fulfilled in the Composite-TOE.

Evaluator action elements:

ADV_COMP.1.1E

The evaluator shall confirm that the rationale for design compliance is complete, coherent, and internally consistent.

Evaluator actions:

Action ADV_COMP.1.1E

ADV_COMP.1-1 The evaluator ***shall examine*** the rationale for design compliance to determine that all applicable requirements on the application, imposed by the underlying platform, are fulfilled by the Composite-TOE.

In order to perform this work unit the evaluator shall use the rationale for design compliance as well as the TSF representation on the ADV_TDS, ADV_ARC and ADV_IMP levels on the one side and the input of the platform developer in form of the certification report, user guidance and ETR_COMP on the other side.

The evaluator shall analyse which platform requirements are applicable for the current Composite-TOE.
The evaluator shall compare each of the applicable requirements with the actual specification and/or implementation of the Composite-TOE and determine, for each requirement, whether it is fulfilled.
As result, the evaluator confirms or disproves the rationale for design compliance.

For example, platform guidance may require an application to perform a special start-up sequence testing the current state of the platform. Such information might be found in the description of secure architecture ADV_ARC of the composite TOE; see also the *Application Note* above.

The appropriate representation level (ADV_TDS, ADV_ARC and/or ADV_IMP), what the analysis is being performed on, can be chosen and mixed flexibly depending on the concrete Composite-TOE and the requirement in question. Where it is not self-explaining, the evaluator shall justify why the representation level chosen is appropriate.

The evaluator activities in the context of this work unit can be spread over different single evaluation aspects (e.g. over ADV_TDS and ADV_IMP). In this case the evaluator performs the partial activity in the context of the corresponding single evaluation aspect. Then, the notation for this work unit shall be ADV_COMP.1-1-TDS, ADV_COMP.1-1-ARC and ADV_COMP.1-1-IMP, respectively.

If the assurance package chosen does not contain the families ADV_TDS, ADV_ARC or ADV_IMP (e.g. EAL1), this work unit is not applicable (cf. *Application Note* above).

The result of this work unit shall be integrated to the result of ADV_TDS.1-2E / ADV_TDS.1-7, ADV_ARC.1.1E / ADV_ARC.1.1C / ADV_ARC.1-1, ADV_IMP.1.1E / ADV_IMP.1.1C / ADV_IMP.1-1 (or the equivalent higher components if a higher assurance level is selected).

ADV_COMP.1-2 The evaluator ***shall check*** the TSF internals of the Composite-TOE to determine that they do not contradict any design requirement imposed by the underlying platform.

The TSF internals are represented and evaluated in the context of the assurance family ADV_INT. The evaluator shall compare the internal structure of the TSF with the design requirements of the platform and search for obvious contradictions.

If there are no requirements of the platform concerning the TSF internal structure or the assurance package chosen does not contain the family ADV_INT, this work unit is not applicable.

The result of this work unit shall be integrated to the result of ADV_INT.2.1E / ADV_INT.2.1C / ADV_INT.2-1 (or the equivalent higher components if a higher assurance level is selected).

5.1.8.1.4 Composite functional testing (ATE_COMP)

The composite-specific work units defined in this chapter are intended to be integrated as refinements to the evaluation activities of the ATE class listed in the following table. The other activities of ATE class do not require composite-specific work units.

CC assurance family	Evaluation activity	Evaluation work unit	Composite-specific work unit
ATE_COV	ATE_COV.1.1C	ATE_COV.1-1	ATE_COMP.1-1
	ATE_COV.1.1C	ATE_COV.1-1	ATE_COMP.1-2
ATE_FUN	ATE_FUN.1.2C	ATE_FUN.1-3	ATE_COMP.1-1
ATE_IND	ATE_IND.1.2E	ATE_IND.1-5	ATE_COMP.1-3

NB: If the level of the assurance requirement chosen is higher than those identified in this table, the composite-specific work unit is also to apply.

ATE_COMP.1 Composite product functional testing

Objectives

The aim of this activity is to determine whether the Composite-TOE *as a whole* exhibits the properties necessary to satisfy the security functional requirements of its Security Target.

Application notes

A composite product can be tested *separately* and *integrative*. Separate testing means that the platform and the application are being tested independent of each other. A lot of tests of the platform may have been performed within the scope of its accomplished evaluation. The application may be tested on a specifically instrumented version of the platform or/and in a platform emulator representing a virtual abstract machine.
Integration testing means that the composite product is being tested as it is: the application is running on the intended real platform.

Behaviour of implementation of some SFRs can depend on properties of the underlying platform as well as of the application, e.g. correctness of the measures of the composite product to avoid unintended information flow between different applications of the Composite-TOE or between the Composite-TOE and its outside environment. In such a case the SFR implementation shall be tested on the final composite product, but not on a specifically instrumented version of the platform or in an emulator.

This activity focuses exclusively on testing of the composite product *as a whole* and represents merely *partial efforts* within the general test approach being covered by the assurance class ATE. These integration tests shall be specified and performed, whereby the approach of the standard[14] assurance families of the class ATE shall be applied.

- A correct behaviour of the Platform-TSF being relevant for the Composite-ST (corresponding to the group *RP_SFR* in the work unit ASE_COMP.1-1 above), and
- absence of exploitable vulnerabilities (sufficient effectiveness) in the context of the Platform-ST

are confirmed by the valid Platform Certificate, cf. chapter 5.1.6 above.

Dependencies:

No dependencies.

Developer action elements:

[14] i.e. as defined by CEM

ATE_COMP.1.1D

The developer shall provide a set of tests as required by the assurance package chosen.

ATE_COMP.1.2D

The developer shall provide Composite-TOE samples suitable for testing.

The developer shall provide, if necessary, specifically instrumented versions of the platform or/and a platform emulator and the applications in the scope of the composite evaluation suitable for testing.

Content and presentation of evidence elements:

ATE_COMP.1.1C

Content and presentation of the specification and documentation of the *integration* tests shall correspond to the standard[15] requirements of the assurance families ATE_FUN and ATE_COV.

ATE_COMP.1.2C

The Composite-TOE provided shall be suitable for testing.

If necessary, specifically instrumented versions of the platform or/and a platform emulator and the applications provided shall be suitable for testing.

Evaluator action elements:

ATE_COMP.1.1E

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

ATE_COMP.1.2E

The evaluator shall specify, perform and document a set of own *integration* tests to confirm that the Composite-TOE operates as specified.

Evaluator actions:

Action ATE_COMP.1.1E

ATE_COMP.1-1 The evaluator ***shall examine*** that the developer performed the *integration* tests for all SFRs having to be tested on the Composite-TOE as *a whole*.

In order to perform this work unit the evaluator shall analyse, for each SFR from the Composite-ST, whether it directly depends on security properties of the platform and of the application, simultaneously.
Then the evaluator shall verify that the *integration* tests performed by the developer cover at least all such SFRs.

If the assurance package chosen does not contain the families ATE_FUN and ATE_COV (e.g. EAL1), this work unit is not applicable.

The result of this work unit shall be integrated to the result of ATE_COV.1−1C / ATE_COV.1-1 and ATE_FUN.1.2C / ATE_FUN.1-3 (or the equivalent higher components if a higher assurance level is selected).

Action ATE_COMP.1.2E

ATE_COMP.1-2 The evaluator ***shall determine*** the minimal amount of the *integration* tests being necessary for the current composite evaluation.

[15] i.e. as defined by CEM

The evaluator shall perform the following steps:

1) The evaluator determines the share of the platform part of the TOE in enforcing the Composite-ST.
In order to do this, the evaluator refers to the TOE design (ADV_TDS and ADV_ARC might possess an appropriate representation level for this purpose) as well as to the Composite-ST and lists all the Composite-SFRs using the security services of the platform[16]. Hereby the evaluator shall understand and document, for each such Composite-SFR, what concretely the platform does (so called *platform's share*).

For example, there is a Composite-SFR FCS_CKM.1 'Key Generation'. The share of the platform in implementing this SFR may be generating random numbers used for the key generation.

2) The evaluator checks, for each such Composite-SFR, whether the *platform's share* has fully been covered by the Platform Certificate; in order to do this the evaluator refers to the platform user guidance, ETR_COMP and the platform certification report.

For our example with the random number generator (RNG), the evaluator might find an advice in ETR_COMP that (i) the RNG is being tested before each use (online tests) and (ii) generated random numbers possess a sufficient (for the generation of static keys) quality. In such a case the evaluator can decide that no *integration* tests are necessary for FCS_CKM.1/Key Generation[17].

The next example represents a situation, where additional *integration* tests are necessary.
There let be a Composite-SFR FPR_UNO.2 (unobservability of operations and unobservable storage of secrets) fulfilled by the following TSF-portions:
- 'Key Generation': the *platform's share* in implementing the related Composite-SFR (FCS_CKM.1) is disguising information about the value of the key being generated. Important information sources there are observing power consumption by the underlying hardware (may be outside the Composite-TOE) as well as processing time and memory management by the platform;
- 'Signature Creation': the *platform's share* in implementing the related SFR (FCS_COP.1) is disguising information about the value of the signature key being used. Important information sources there are observing power consumption by the underlying hardware (may be outside the Composite-TOE) as well as processing time and memory management by the platform;

For processing time by the platform, the evaluator might find an advice in ETR_COMP that independence of the platform processing time has been evaluated, whereby this platform security property is application-independent.

For memory management by the platform, the evaluator might find an advice in ETR_COMP that cleaning of CPU registers and memory caches upon their deallocation from all objects has been evaluated, but no advice on a spread

[16] Such security services of the platform are represented by the group *RP_SFR, cf.* the work unit ALC_COMP.1-1 above

[17] Of course, the evaluator might (and, perhaps, would) decide to test the 'Key Generation'-TSF as an integration test on the final TOE. A methodological reason for this test would then be rather a general check of the respective functionality, but not the confirmation of security behaviour of the platform-RNG.

storage of the related objects. Hence, the evaluator has to assume that the spread-storage-of-private-keys behaviour of the platform was not included in the consideration of the platform evaluation. Therefore, the evaluator has to specify an integration test for 'spread-storage-of-private-keys' behaviour while generating and using private keys.
For disguising against the power consumption by the underlying hardware, the evaluator might find no advice in ETR_COMP, because the underlying hardware is out of the scope of the Composite-TOE. Hence, the evaluator has to assume that this behaviour of the platform was not included in consideration. Therefore, the evaluator has to specify an integration test for power consumption while generating and using private keys.

3) The evaluator refers to ETR_COMP and checks for any explicit requirements for performing tests in the context of the composite evaluation. Such explicit requirements might sound like 'Such test has to be performed during the application / composite evaluation'.

All platform tests being necessary for the current composite evaluation, but not covered by the Platform Certificate, and all composite tests explicitly required by ETR_COMP, encompass together the minimal amount of the integration tests being necessary for the current composite evaluation.

This work unit is the complementary part to the work unit 1-1.COMP_ASE In 1-1.COMP_ASE the evaluator determines, on which part of the Platform-ST the Composite-ST can rely (the group *RP_SFR*); in the current work unit the evaluator determines, whether the Composite-ST can also rely on the platform's *functional* behaviour being not covered by the Platform-ST.

The result of this work unit shall be integrated to the result of ATE_COV.1.1C / ATE_COV.1-1 (or the equivalent higher components if a higher assurance level is selected).

ATE_COMP.1-3 The evaluator ***shall perform*** the standard[18] evaluator actions in the context of the assurance family ATE_IND on the set of the *integration* tests using the Composite-TOE as *a whole*.

The set of the *integration* tests for this activity shall embrace at least the minimal amount of the integration tests as determined in the previous work unit.

The result of this work unit shall be integrated to the result of ATE_IND.1.2E / ATE_IND.1-5) (or the equivalent higher components if a higher assurance level is selected).

5.1.8.1.5 Composite vulnerability assessment (AVA_COMP)

The composite-specific work units defined in this chapter are intended to be integrated as refinements to the evaluation activities of the AVA class listed in the following table. The other activities of AVA class do not require composite-specific work units.

[18] i.e. as defined by CEM

CC assurance family	Evaluation activity	Evaluation work unit	Composite-specific work unit
	AVA_VAN.1.3E	AVA_VAN.1-5	AVA_COMP.1-1
	AVA_VAN.1.3E.	AVA_VAN.1-6	AVA_COMP.1-2
	AVA_VAN.1.3E	AVA_VAN.1-7	AVA_COMP.1-2
AVA_VAN	AVA_VAN.1.3E	AVA_VAN.1-8	AVA_COMP.1-2

NB: If the level of the assurance requirement chosen is higher than those identified in this table, the composite-specific work unit is also to apply.

AVA_COMP.1 Composite product vulnerability assessment

Objectives

The aim of this activity is to determine the exploitability of flaws or weaknesses in the Composite-TOE *as a whole* in the intended environment.

Application notes

This activity focuses exclusively on vulnerability assessment of the composite product *as a whole* and represents merely *partial efforts* within the general approach being covered by the standard[19] assurance family AVA_VAN.

The results of the vulnerability assessment for the underlying platform represented in the ETR_COMP can be reused, if they are up to date and all composite activities for correctness – ASE_COMP.1, ALC_COMP.1, ADV_COMP.1 and ATE_COMP.1 – are finalised with the verdict PASS.

Due to composing the platform and the application induces a new quality arises, which can cause additional vulnerabilities of the platform which might be not mentioned in the ETR_COMP.

Dependencies:

No dependencies.

Developer action elements:

AVA_COMP.1.1D

The developer shall provide the Composite-TOE for penetration testing.

Content and presentation of evidence elements:

AVA_COMP.1.1C

The Composite-TOE provided shall be suitable for penetration testing as *a whole*.

Evaluator action elements:

AVA_COMP.1.1E

The evaluator shall conduct - building on evaluator's own vulnerability analysis - penetration testing of the Composite-TOE as *a whole* to ensure that the vulnerabilities being relevant for the Composite-ST are not exploitable.

Evaluator actions:

Action AVA_COMP.1.1E

[19] i.e. as defined by CEM

AVA_COMP.1-1 The evaluator ***shall examine*** the results of the vulnerability assessment for the underlying platform to determine that they can be reused for the composite evaluation.

The results of the vulnerability assessment for the underlying platform are usually represented in the ETR_COMP. They can be reused, if they are up to date and all the composite activities for correctness – ASE_COMP.1, ALC_COMP.1, ADV_COMP.1 and ATE_COMP.1 – are finalised with the verdict PASS.
The evaluator shall also consider the relevant determinations in the Platform Certification Report. For validity of the platform security certificate please refer to chapter 5.1.6 above.

The result of this work unit shall be integrated to the result of AVA_VAN.1.3E / AVA_VAN.1-5 (or the equivalent higher components if a higher assurance level is selected).

AVA_COMP.1-2 The evaluator ***shall specify, conduct and document*** penetration testing of the Composite-TOE as *a whole* using the standard approach of the assurance family AVA_VAN.

If the correctness-related composite-specific activities – ASE_COMP.1, ALC_COMP.1, ADV_COMP.1 and ATE_COMP.1 – are finalised with the verdict PASS and the certificate for the platform covers all security properties needed for the Composite-TOE, composing the platform and the application must not create additional vulnerabilities of the platform.

If the evaluator determined that composing the platform and the application creates additional vulnerabilities of the platform[20], a contradiction to the verdict PASS for the correctness activities (see paragraph 0 above) has to be supposed or the platform certificate does not cover all security properties needed for the current Composite-TOE.

The result of this work unit shall be integrated to the result of AVA_VAN.1.3E / AVA_VAN.1-6, AVA_VAN.1-7, AVA_VAN.1-8 (or the equivalent higher components if a higher assurance level is selected).

5.1.9 Appendix 2: ETR for composite evaluation template

The related document “ETR-template-for-composition” shall be used as a template by the Platform Evaluator to issue the ETR_COMP. Please note that the layout can be customized according to the evaluation facilities company standard, but the contents and structure are mandatory.

[20] i.e. not mentioned in the ETR_COMP

5.2 AVA class in the context of composition

This sub-section illustrates vulnerability analysis activities (called AVA in CC terms) in the context of a composition.

5.2.1 Software composition

This point has already been studied in the section 5.1.4.6 of the document.

5.2.2 Hardware composition

The composition as it is used in the hardware community, that is to say, to compose an application layer on top of a pre-certified hardware layer (e.g; a smartcard) is defined in the document “Composite product evaluation for Smart Cards and similar devices, v1.4” issued by JIL consortium (refered as [Composite] in the appendix).

It was written in the context that the hardware part and associated libraries (if applicable) is evaluated independently as it can be used with many different software applications

The software is embedded in the hardware and is built to operate with this hardware. The resulting product is the one which is used in the field (cellular phones, banking cards, health cards, Identity, digital signature , e-pass, e-ticketing etc.) and on which customers/users need to gain confidence.

Software applications may be built to operate with the support of an OS. The OS provides a separation mechanism between itself and the software applications as well as services to the software applications.

Appendix A. Cross reference of EALs and assurance components

The table below describes the relationship between the evaluation assurance levels and the assurance classes, families and components.

Key:

- Cells in black letter are covered by the CEM
- Cells in red letter are covered by the present document

Assurance class	Assurance Family	Assurance Components by Evaluation Assurance Level						
		EAL1	EAL2	EAL3	EAL4	EAL5	EAL6	EAL7
Development	ADV_ARC		1	1	1	1	1	1
	ADV_FSP	1	2	3	4	5	5	6
	ADV_IMP				1	1	2	2
	ADV_INT					2	3	3
	ADV_SPM						1	1
	ADV_TDS		1	2	3	4	5	6
Guidance documents	AGD_OPE	1	1	1	1	1	1	1
	AGD_PRE	1	1	1	1	1	1	1
Life-cycle support	ALC_CMC	1	2	3	4	4	5	5
	ALC_CMS	1	2	3	4	5	5	5
	ALC_DEL		1	1	1	1	1	1
	ALC_DVS			1	1	1	2	2

Assurance class	Assurance Family	Assurance Components by Evaluation Assurance Level						
	ALC_FLR							
	ALC_LCD			1	1	1	1	2
	ALC_TAT				1	2	3	3
Security Target evaluation	ASE_CCL	1	1	1	1	1	1	1
	ASE_ECD	1	1	1	1	1	1	1
	ASE_INT	1	1	1	1	1	1	1
	ASE_OBJ	1	2	2	2	2	2	2
	ASE_REQ	1	2	2	2	2	2	2
	ASE_SPD		1	1	1	1	1	1
	ASE_TSS	1	1	1	1	1	1	1
Tests	ATE_COV		1	2	2	2	3	3
	ATE_DPT			1	1	3	3	4
	ATE_FUN		1	1	1	1	2	2
	ATE_IND	1	2	2	2	2	2	3
Vulnerability assessment	AVA_VAN	1	2	2	3	4	5	5

Appendix B. List of Abbreviations

ADV	Development
ATE	Tests
ANSSI	Agence Nationale de la Sécurité des Systèmes d'Information – French Certification Body
AIS	Application note and Interpretation of the Scheme
API	Application Programming Interface
ARG_CDS	An argumentation on the link between the formal model and the security target
ARG_FSP	A correspondence between the formal model and the specification of the TOE
ARG_PROOF	A complete list of the hypotheses of the model (with justifications)
ARG_SPM	Explanation of the SPM formal model (design, conception, …)
ARG_TOOL	Argumentation on the formal methodology considered.
BSI	Bundesamt für Sicherheit in der Informationstechnik – German Certification Body
CB	Certification Body
CC	Common Criteria
CCMB	Common Criteria Management Board
cPP	Collaborative Protection Profile
CCRA	Common Criteria Recognition Agreement
CEM	Common Evaluation Methodology
DTR	Detailed Technical Report
ITSEF	Information Technology Security Evaluation Facility
EAL	Evaluation Assurance Level
ETR	Evaluation Technical Report
HW	Hardware
HSM	Hardware Security Module
JHAS	JIL Hardware-related Attacks Subgroup
JIWG	Joint International Working Group
JTEMS	JIL Terminal Evaluation Methodology Subgroup
MILS	Multiple Independent Levels of Security
OR	Observation Report
POI	Point Of Interaction
PP	Protection Profile
PROOF	The proofs associated to the formal model
SAR	Security Assurance Requirement

SK	Separating Kernel
TOE	Target of Evaluation
TSF	TOE Security Functionality
TSFI	TSF Interface
SFR	Security Functional Requirement
SOG-IS	Senior Officials Group – Information Systems Security
ST	Security Target
SRC	Source code of the formal model

Appendix C. Bibliography

[AIS34] Application Notes and Interpretation of the Scheme (AIS), AIS34, v3, September 2009

[CC3.1] Common Criteria for Information Technology Security Evaluation. Version 3.1, revision 4, vol. 1--3, September, 2012,

[CCOPP-OS] COTS Compartmentalized Operations Protection Profile Operating Systems, v2.0, 2008

[CEM] Common Methodology for Information Technology Security Evaluation, Evaluation methodology, September 2012, Version 3.1, revision 4

[SOG-IS] Mutual Recognition Agreement of Information Technology Security Evaluation Certificates, v3.0, January 2010

[Note 12] ANSSI : Note d'application, réf. 12.1, Modélisation formelle des politiques de sécurité d'une cible d'évaluation, March 2008

[Tenix ST] Tenix Datagate Inc, Interactive link data diode device: Common Criteria security target, no. 9126P01000001, August, 2005, http://www.commoncriteriaportal.org/files/epfiles/st_vid9512-st.pdf.

[WR MILS] Wind River VxWork MILS Platform, PO_VE_MILS_Platform.pdf, Rev 08/2010, www.windriver.com

[AAPHDSB] Application of Attack Potential to Hardware Devices with Security Boxes, Version 1.0, May 2012

[AAPP] Application of Attack Potential to POIs, Version 1.0, June 2011

[CEM Refinements] CEM Refinements for POI Evaluation, Version 1.0, May 2011

[SECDT] Security Evaluation and Certification of Digital Tachygraphy

[SWI] INTERPRETATION DU VLA.4/VAN.5 DANS LE DOMAINE DU LOGICIEL, VUL/I/01.1, Version 1

[Note 18] ANSSI : PRISE EN COMPTE DES OUTILS DANS LES EVALUATIONS LOGICIELLES

[Composite] Composite product evaluation for Smart Cards and similar devices, v1.4

END OF THE DOCUMENT