

# **Evaluator Training Course**

## **Guidance on Vulnerability Search and Analysis**

**Version 4.1, July 2003**

---

This document gives guidance on how to initiate the process of identifying vulnerabilities in conjunction with other evaluation activities. Particular focus is given to the development representation evaluation activities

It is a supplement to version 4.0 (July 2000) of the evaluator training material. Versions 4.0 of the other components of the evaluator training material remain current at this time.

## Background

1. Both CEM and UKSP 05 Part III provide a methodology for identifying vulnerabilities in a TOE, based around a taxonomy of generic attack methods. Such a methodology is a requisite part of the evaluation activity of *independent vulnerability analysis*.
2. Whilst the checklist approach inherent in both methodologies is intended to enhance the reproducibility of evaluations, it is evident from experience that its effectiveness at helping evaluators find vulnerabilities could be significantly improved. This is not because of any perceived flaws with the approach itself; rather it is because its effectiveness is reliant on the checklist being applied *in conjunction with* other evaluation activities, such as design or code analysis – and neither CEM nor UKSP 05 provide any guidance on how this might be achieved in practice.
3. To provide such guidance is the fundamental aim of this document.

## Scope

4. As is implied above, this guide is applicable to both CC and ITSEC evaluations. Moreover, it is intended to be fully applicable to SYSn evaluations and Fast Track Assessments.
5. In a CC evaluation, independent vulnerability analysis is only a requirement at AVA\_VLA.2 and higher, which features in EAL4 and higher assurance levels. Nonetheless, the principles can be expected to apply at other CC assurance levels. This is because an awareness of the checklist of generic attack methods should help evaluators in identifying ‘obvious’ (or ‘encountered’<sup>1</sup>) vulnerabilities. In this sense the differences between an evaluation at AVA\_VLA.1 (e.g. EAL3) and one at AVA\_VLA.2 (e.g. EAL4) are that:
  - a) AVA\_VLA.1 does not require formal application of the generic attack method checklist, and hence does not require the ETR to demonstrate that it has been applied in a comprehensive manner. This reflects the nature of the ‘passive’ search for vulnerabilities that is expected at this level: evaluators are not required to actively search for vulnerabilities (and hence demonstrate that they have done so), but they **are** expected to be alert to any vulnerabilities that are ‘obvious’ from the evidence examined (e.g. ADV deliverables).
  - b) AVA\_VLA.1 will generally (though not necessarily) be applied where only a subset of the TSF representation evidence is available; hence (as will be seen) an evaluation at this level will not be able to give full consideration to all of the generic attack methods described in the CEM.
6. This guide focuses on the ADV activities (i.e. design and code analysis), as this is the area that is felt to be most in need of guidance. The ADV activities are the most significant of the non-testing group of activities in terms of evaluator effort, and as such ought to be a fruitful source of penetration testing ideas. Nevertheless the principles expounded here should be applied (where relevant) to other activities as well, e.g. the evaluation of guidance documentation.

---

<sup>1</sup> The term ‘encountered’ vulnerability, introduced in recent AVA proposals from the CCIMB, is arguably a better characterisation of such vulnerabilities and how an evaluator discovers them.

## Terminology

7. In line with the current trend towards CC evaluations, or approaches that are strongly based on CC principles (e.g. SYSn evaluation, FTA), this guide uses CC terminology.
8. Nonetheless, as stated above, the guide is equally applicable to ITSEC evaluations. The guide may thus be interpreted appropriately for such evaluations, generally by substitution of equivalent terms, for example:
  - a) *Architectural design* for *high-level design*
  - b) *Detailed design* for *low-level design*
  - c) *Major component* for *TSF subsystem*
  - d) *Basic component* for *module*.
9. This guide assumes that the reader is sufficiently familiar with ITSEC and CC to be able to relate the guidance to the relevant parts of an ITSEC evaluation where needed.
10. Notwithstanding the terminology used in CEM and UKSP 05 Part III, for the sake of clarity, the following terms are used in this guide:
  - a) The term *attack method* (occasionally abbreviated to just *attack*) refers to a particular means or technique by which an attacker attempts to exploit a known or suspected vulnerability in the TOE.
  - b) A *generic attack method* is one that is expressed in sufficiently abstract terms as to be applicable to many different types of TOE.
  - c) The term *vulnerability* is used strictly within the meaning defined in CEM, i.e. *a weakness in the TOE that can be used to violate a security policy* (normally helping to realise one or more threats, thereby leading to compromise of the assets).

## The Approach to Searching for Vulnerabilities

11. This guide presents two different (but complementary) views of the search for vulnerabilities:
  - a) The *Vulnerability Oriented View*, which focuses on the different types of generic attack method, as described in CEM and UKSP 05.
  - b) The *Assurance Component Oriented View*, which focuses on the activities of examining ADV deliverables in particular, and discusses the various checks that may result in the discovery of vulnerabilities by suggesting potentially fruitful attack methods.
12. We first present the *Vulnerability Oriented View*; this view should be the one that is most familiar to the reader, since it follows on from the independent vulnerability analysis approach described in CEM and UKSP 05. We build on this approach by linking the various types of generic attack method to the ADV activities where the vulnerabilities they exploit they might be first discovered.
13. We then move on to the *Assurance Component Oriented View*, which considers the various checks that are required for the examination of each TSF representation, and advises the evaluator on design and implementation features that may be the source of vulnerabilities.

## Vulnerability Oriented Search

14. This section summarises the generic attack methods described in the CEM and UKSP05 approaches, and expands on those documents by identifying their relationship to the ADV criteria checks.
15. CEM and UKSP 05 identify the four basic attack methods:
  - a) Bypassing
  - b) Tampering
  - c) Direct Attack
  - d) Misuse.
16. A key to understanding the differences between these four basic attack methods is to appreciate the following points:
  - a) Insofar as the TOE design is concerned, there exist a number of *expected routes* that an attacker may take to the protected asset. In order for the asset to be adequately protected, the appropriate SF(s) must be invoked at some point along the route.
  - b) Insofar as the SF specifications are concerned, each SF has an *expected behaviour*; that is, an SF operates in such a way as to protect the asset it is intended to defend.
17. The four basic attack methods can be characterised in terms of how they are aimed at undermining either of these properties. The following table provides a definition of each in these terms.

<b>Attack</b>	<b>Expected route taken?</b>	<b>SF behaving as expected?</b>
Bypass	×	✓
Tampering	✓	×
Direct Attack	✓	✓
Misuse	✓	×

18. This table illustrates that *Tampering* and *Misuse* attacks are closely linked, in that the attacker aims to exploit the fact that the behaviour of the SF has been altered to the attacker's advantage. They differ in respect of *who* (or *what*) caused the alteration in behaviour. In the case of *Tampering*, it is the attacker who actively causes the change in behaviour; in the case of *Misuse*, the change in behaviour arises as a result of human or other operational error. The key to identifying the potential for these attacks thus lies in discovering the ways of altering the behaviour of an SF.
19. By contrast, *Bypass* attacks do not depend on the behaviour of the SF being altered. Rather, the attacker seeks an alternative route to the asset – a route that the developer was not expecting. If a *Bypass* attack succeeds, it is because either no SFs are invoked on this alternative route, or because the SFs that are invoked are ineffective at protecting the asset. The key to identifying the potential for these attacks thus lies in discovering alternative routes to the assets that require

protection.

20. Finally, there is *Direct Attack* which (as the table illustrates) involves an attacker attempting to defeat an SF without any attempt to either select an alternative route to the asset, or to alter the behaviour of the SF. The key to identifying the potential for such attacks lies in discovering weaknesses in the underlying mechanisms used to implement a SF.

### *Bypassing Attacks*

21. The first class of bypass attack is that which exploits the capabilities of TSF interfaces or of utilities that interact with the TOE (often at a low level), so as to find an *alternate* route to the asset requiring protection. For this class of attack, the specification of external interfaces in the FSP is the primary source of inspiration for possible penetration tests. However, it is often the case that detail in less abstract representations will help refine or focus ideas. The five attacks in this category are:
  - B1.1 Bypass by changing a pre-defined sequence order.
  - B1.2 Bypass by inserting a component into a pre-defined sequence.
  - B1.3 Bypass by using a component in an unexpected context or for an unexpected purpose.
  - B1.4 Bypass through exploitation of implementation detail in less abstract representations.
  - B1.5 Bypass through use of delay between time of check and time of use
22. It may be noted that the term *sequence* in this context refers to a set of operations that are performed in a particular order. These operations may correspond to external interface calls, actions performed by interacting TSF subsystems, or may refer to interactions between the TOE and the IT environment.
23. The second class of bypass attack is that which exploits inheritance of privilege. Here, the attacker seeks an *alternate* route to the asset that, whilst defended by SFs, does not prevent compromise of the asset because the route confers upon the attacker the privilege to circumvent those SFs. For this class of attack it is primarily the FSP, supported by information in the HLD (and possibly the guidance documentation) that provides the main source of ideas for penetration tests. The three attacks in this category are:
  - B2.1 Privilege inheritance through illicit execution of data.
  - B2.2 Privilege inheritance through generation of unexpected input.
  - B2.3 Privilege inheritance through invalidation of properties or assumptions on which lower-level components rely.
24. Finally, there are the bypass attacks that are based on reading sensitive data stored in inadequately protected areas. Here the *alternate* route to the asset is opened up by the TOE itself storing the asset (sensitive data in this case), or a copy of it, in a location that is not adequately defended by the SFs. With this class of attack, the HLD in conjunction with the FSP and guidance documentation is the likeliest source of penetration test ideas. The three attacks in this category are:
  - B3.1 Read sensitive data stored in inadequately protected areas of memory or disk storage.

B3.2 Read sensitive data by exploiting access to shared resources.

B3.3 Read sensitive data by exploiting error recovery.

25. The following table summarises the attacks and likely sources of potential vulnerabilities.

Attack Method	Source	Action
Exploit interface capabilities		
Sequence manipulation [B1.1, B1.2]	FSP	Are any security-relevant operations implemented by a sequence of interface calls?  In such a sequence, which interface call(s) do not result in the invocation of the appropriate SF?  Are there defined sequences of interactions between the TOE and the IT environment that could be susceptible to manipulation, e.g. man-in-the-middle attacks?
	HLD	Are any SFs implemented by a sequence of TSF subsystem actions?  In such a sequence, can the TSF subsystems, or communications between them, be manipulated?
Unexpected use of interface – context or purpose [B1.3]	FSP	Are there any ‘general-purpose’ TSF interfaces that could be used for an unexpected security-relevant purpose?
	LLD IMP	Are implementation details introduced in the LLD or implementation representation that provide the potential to use an interface in an unexpected manner?
Exploit implementation detail [B1.4]	HLD LLD IMP	What shared resources are evident in the TSF representation that might be exploited for illicit data flow?
Exploit time delay [B1.5]	FSP HLD	For those sequences identified as above, are any susceptible to the injection of a time delay? Could something happen in an interim period or timeslot that invalidates previous enforcement?
Inherit privileges		
Unexpected execution [B2.1]	FSP HLD	What privileged programs or applications are present in the TOE?  Is there any potential for execution of data input directly or indirectly from their external interface?
Unexpected input [B2.2]	FSP	What potential exists, as evident from the external interfaces to privileged programs and applications, for unexpected input to cause an uncontrolled exit, resulting in illicit privilege inheritance?
Invalidate assumptions/properties [B2.3]	HLD	Given the layer or level at which the SFs are enforced, are the lower layers/levels that provide access to the asset assumed not to be accessible? Are those assumptions valid, or could they be undermined in any way?

Attack Method	Source	Action
Read sensitive data		
Unprotected areas [B3.1]	HLD FSP	If the TOE processes sensitive user data, does the TOE keep copies of the data (whether temporary or not)? If so, where is it stored, and who can access those areas?
Exploit shared resource access [B3.2]	HLD FSP	What shared resources exist that might contain copies of sensitive user data? How is access controlled to such resources?
Exploit error recovery [B3.3]	FSP AGD	If error recovery is invoked, how does the TOE process sensitive user data, e.g. in recovery or temporary files? Where are these stored, and who can access them?

### *Tampering Attacks*

26. As described above, a tampering attack can be broadly characterised as one that is based on an attacker directly or indirectly altering the behaviour of an SF in order to defeat it.
27. The key to finding such attacks is thus to identify all possible factors that could influence the behaviour of a Security Function. Such dependencies include obvious cases such as TSF data, together with those that are less obvious, such as system resource or parameter limits which may have no direct relevance to security. The former should be evident from the design, whereas identification of the latter may require more intuitive or lateral thinking on the part of the evaluator.
28. There are three basic ways of realising a tampering attack:
  - a) Identify control data or security attributes on which the security functions depend.
  - b) See what happens when the TOE is forced to cope with unexpected input or situations.
  - c) Determine whether security mechanisms can be disabled or whether security enforcement can be delayed.
29. The first class of tampering attack involves accessing data on whose confidentiality or integrity a SF relies. For this class of attack method, any TSF representation may give rise to possible penetration test ideas, though as the representations become less abstract, the methods of attack are likely to be more subtle, and thus less likely to have been addressed by the developer. The three attacks in this category are:
  - T1.1 Read, write or modify internal data directly or indirectly.
  - T1.2 Tamper through use of a component in unexpected context or for unexpected purpose.
  - T1.3 Tamper through exploitation of interference not visible at higher levels of abstraction.
30. The second class of tampering attack is based on forcing the TOE to cope with unusual or unexpected circumstances. The first of these methods relies on the FSP as the primary source of ideas, with less abstract representations helping to refine and focus those ideas. The second (as its definition might suggest) relies more on less abstract representations such as the LLD or code. The two attacks in this category are:

- T2.1 Generate unexpected input for a component.
- T2.2 Invalidate assumptions and properties on which lower-level components rely.
31. The final class of tampering attack is based on disabling or delaying security enforcement. It will be noted that this form of attack will only be effective against those SFs that do not have to complete before access is allowed to the protected asset. An example would be security audit SFs, which record security relevant events, but do not (under normal circumstances) *prevent* the event from taking place. This includes those security audit SFs that are required to intervene on detection of suspicious activity; if such intervention can be delayed, the resultant attack may succeed.
32. For this class of attack, the first two types rely more on the FSP and HLD for ideas, whereas the third relies more on the less abstract representations such as the LLD and code. The three attacks in this category are:
- T3.1 Use interrupts or scheduling functions to disrupt sequencing.
- T3.2 Disrupt concurrence.
- T3.3 Exploit interference between components not visible at a higher level of abstraction.
33. The following table summarises the various forms of tampering attacks, together with the likely sources of ideas for such attacks.

Attack Method	Source	Checklist Action
Accessing data		
Access internal data [T1.1]	FSP HLD LLD IMP	What internal TSF data is used by the SFs, and where is it stored (as evidenced in each representation)?  What entities can access, or have access to, such TSF data?
Unexpected use – context or purpose [T1.2]	FSP HLD LLD IMP	What internal TSF data is used by the SFs, and how is access controlled to it?  Can any TSF interfaces be used to bypass these controls (based on FSP and LLD information in particular)?
Exploit interference [T1.3]	FSP HLD LLD IMP	What internal TSF data is stored in a shared resource (as visible in each representation)?  What entities share these resources? Are any resources susceptible to manipulation?
Force unexpected/unusual circumstances		
Generate unexpected input [T2.1]	HLD FSP	What dependencies of the SF can be identified from the design?  Could any of these dependencies be undermined through unexpected input via a TSF interface, e.g. data in an unexpected format, or of an unexpected size or value?

Attack Method	Source	Checklist Action
Invalidate assumptions [T2.2]	LLD IMP	Do modules (LLD or code) implementing an SF make explicit or implicit assumptions about the data they are handling? Could such assumptions be undermined?
Disable/delay security enforcement		
Disrupt sequencing [T3.1]	HLD FSP	What SFs might be susceptible to this form of tampering?  Are any such SFs implemented by processes that could be manipulated through scheduling or termination?
Disrupt concurrence [T3.2]	HLD LLD	What SFs might be susceptible to this form of tampering?  Do any such SFs rely on shared resources that might be locked against use by that SF?
Exploit interference [T3.3]	HLD LLD IMP	What SFs might be susceptible to this form of tampering?  Are there any dependencies of such SFs that might be exploited to delay enforcement? Could these be influenced or manipulated by other internal entities?

#### *Direct Attacks*

34. As a general rule, direct attack is only relevant to identification and authentication SFs, and any other SFs that are implemented by a similar underlying mechanism. Direct attack penetration tests will generally be devised to confirm or disprove the SOF analysis. However (as noted below) the TOE may contain security relevant mechanisms that are susceptible to direct attack, and which could be exploited to circumvent a SF.

#### *Misuse Attacks*

35. In a similar way, this form of attack will be encapsulated in penetration tests designed to confirm or disprove the misuse analysis. However, it is worth noting that consideration of tampering attacks as described above may give rise to misuse attacks. That is to say, the evaluator may determine that the behaviour of an SF can only be altered by suitably privileged users; but the evaluator should also consider whether the SF behaviour could be *unintentionally* altered as a result of unwitting action (or inaction) by such users.

#### *Composite Attacks and Attack Paths*

36. It is sometimes the case that a chosen attack can be categorised differently according to how it is viewed. To some extent this is because of redundancy in the vulnerability analysis approach, which has the benefit of helping to ensure potential vulnerabilities are not overlooked. Ultimately, of course, the categorisation is very much a secondary issue: what matters is that the evaluators uncover the vulnerability if it is there. However, it can sometimes be helpful to think of such attacks as *composite* attacks.
37. A *composite* attack is one that involves two (or possibly more) different attack methods. For example, a tampering attack that forces a privileged program to cope with unexpected input, in

order to execute a bypassing attack (privilege inheritance). Or, an 'alternate' route to bypass an access control mechanism may be defended by a security relevant mechanism that is itself susceptible to direct attack.

38. More generally, one can consider an *attack path* as a route an attacker may take to compromise an asset, requiring two or more 'hurdles' to be overcome – where each step along the path involves defeating an SF or a security relevant mechanism. Different attack methods may be chosen for each step (or hurdle) along the path.

### Assurance Component Oriented Search

39. In this section we focus on the ADV activities, and provide guidance on how an evaluator might identify potential vulnerabilities during the course of performing the various work units required for these activities.
40. The general approach of this part of the guide is to take each TSF representation in turn, and identify the generic attack methods that might be suggested by the content of that representation. We then relate such identifiable attack methods to the appropriate criteria checks where an evaluator might be expected to encounter them. The expectation is therefore that, when performing such checks, the evaluator is alert to the types of potential vulnerabilities that might be in evidence.

#### *Functional Specification*

41. From the perspective of the search for vulnerabilities, the most significant part of this TSF representation is the specification of external TSF interfaces.
42. Almost any generic attack might be suggested by an external interface specification, for the simple reason that, whatever penetration tests will be identified, there will be one or more external interfaces involved in the exploitation of any vulnerabilities or the execution of a specific attack. The point is that not all such vulnerabilities or attacks will be evident from the FSP. This section deals with only those attack methods that might be suggested from consideration of the FSP alone; it does not address (for example) attack methods suggested by the content of the HLD, LLD or code, that require subsequent consideration of the FSP to convert the ideas into actual penetration tests.
43. The following cases are of particular interest when performing the ADV\_FSP evaluator actions, and as such merit further investigation:
  - a) **Security relevant operations that are carried out by a sequence of interface calls.** If the enforcement of a particular SF is associated with one interface call in the sequence, there may be a possibility of a *sequence manipulation* attack, e.g. changing the order, inserting another call, or missing out a call. Also, the potential for insertion of time delay between time of check and time of use should be considered where relevant (e.g. because the TOE or another process may, in the intervening period, perform some action that renders the previous check invalid or irrelevant).
  - b) **Interfaces that manipulate TSF data.** Whilst the controls over who can call such interfaces are obviously of interest, evaluators should consider (in the detail about the *effects* of calling the interface) where such data is stored, and whether an attacker can gain access to such areas. Evaluators should also consider whether the inputs to these

interfaces have assumed limits or assumed formats; and if so, whether inputs can be supplied that do not comply with such assumptions.

- c) **Interfaces to privileged programs/applications.** The FSP should identify such programs or applications that run with privilege. The interfaces to these may be susceptible to attacks based on *privilege inheritance*. Evaluators should consider the possibility of supplying unexpected input to such interfaces (with the aim of forcing an uncontrolled exit, or forcing it to execute a series of commands). Evaluators should also consider whether there are any other features that might be exploited, e.g. to get to the command line interface whilst the program is running.
  - d) **Interfaces that process sensitive user data.** It is worth considering any detail about the effects of calling such interfaces, in terms of any user-visible information relating to internal processing, e.g. where temporary copies of sensitive files are stored.
  - e) **Interfaces that access user-visible shared resources.** Such interfaces should be considered for the potential afforded by accesses to such resources, such as illicit access to sensitive user or TSF data, or manipulating the content of such resources to affect the operation of another process or cause an illicit data flow. In this case, it is likely that evaluators will need to use other sources of information (e.g. HLD, guidance) to identify all such resources.
  - f) **General purpose interfaces.** Such interfaces should be considered for potential use in an unexpected way (e.g. with specially constructed inputs), with a view to performing an action that would otherwise be subject to a SF, i.e. achieve bypass. Such interfaces may also be susceptible to attacks based on supplying unexpected inputs. In both cases, the potential for attack may be difficult to identify based on FSP information alone. Evaluators may need to consider information from other TSF representations (particularly the LLD or code if available) in order to formulate attacks against such interfaces.
44. The ADV\_FSP work units that are most likely to reveal potential vulnerabilities as described above involve the following checks on the content and presentation of evidence.
45. *Inconsistencies* (e.g. ADV\_FSP.2-2) may help realise any attack, depending on the nature of the inconsistency.
46. *Identification of external interfaces* (e.g. ADV\_FSP.2-3) focuses on direct and indirect interfaces to the TSF. An indirect interface that can be called up directly (e.g. a command line interface used by a GUI interface) is worthy of investigation for bypass potential: there may be fewer controls over input values, thereby providing the potential for attacks based on unexpected input to or use of an interface.
47. *Description of all external interfaces* (e.g. ADV\_FSP.2-4) focuses on the scope and completeness of the interface specification, and ensuring that each external interface is adequately described, to the extent that its security relevance can be determined. This includes all programs that execute with privilege.
48. *Description of complete behaviour* (e.g. ADV\_FSP.2-5) considers the documentation of input parameters, externally visible effects, and error codes returned. This is ideal for attacks such as (for example) providing input values that are out of range. Descriptions of protocols and user-

accessible databases that direct the activities of programs may indicate possible tampering attacks directed against the dependencies of an SF.

### *High-Level Design*

49. Here we see the top-level decomposition of the TOE into major structural units or TSF subsystems. Of particular interest is the way the subsystems work together to provide the Security Functions. Evaluators should look for any dependencies or shared resources, including both TSF data and less tangible dependencies such as resource availability.
50. The following cases are of particular interest when performing the ADV\_HLD evaluator actions, and as such merit further investigation:
  - a) **Availability of interfaces to ‘low-level’ subsystems.** In this context, a ‘low-level’ subsystem is one that implements ‘low level’ functions that are relevant to security (in other words, functionality that is subject to one or more SFs). An example would be a subsystem that provides functions to access the contents of files stored on disk, where the ST requires the enforcement of file access controls. The availability of the interfaces to such subsystems should be considered for potential bypass of the relevant SFs (especially where those SFs are implemented by another subsystem).
  - b) **Security relevant operations that are carried out by a sequence of subsystem actions.** If the enforcement of a particular SF is associated with subsystem in the sequence, there may be a possibility of a *sequence manipulation* attack. For example, one or more subsystems may be processes that can be killed, or affected by scheduling functions. Or, there may be communications links between interacting subsystems that offer the potential for exploitation (e.g. through man-in-the-middle attacks).
  - c) **Storage of TSF data relied upon by subsystems.** The HLD should identify where TSF data that its subsystems rely on is stored, both permanent and temporary. The potential for attackers gaining access to such storage areas should be investigated.
  - d) **Other subsystem dependencies.** Other, less tangible, dependencies should be considered where these are evident in the HLD, e.g. availability of system resources, or dependencies on system-wide configuration parameters.
  - e) **Operation of privileged programs/applications.** The operation of programs or applications that run with privilege should be considered, particularly for the potential for attacks based on *privilege inheritance*. Evaluators should pay particular attention to ‘indirect inputs’ that may affect the operation of such privileged entities (e.g. configuration or initialisation files) and whether these can be manipulated.
  - f) **Subsystems that process sensitive user data.** Evaluators should investigate whether such subsystems make temporary copies of the data they process (e.g. for subsequent processing by another subsystem), and if so whether they offer the potential for illicit access.
  - g) **Shared resources.** The HLD should identify resources shared between different subsystems or processes. The evaluators should consider the potential afforded by accesses to such resources, such as illicit access to sensitive user or TSF data, or manipulating the content of such resources to affect the operation of another process or cause an illicit data flow.

51. The ADV\_HLD work units that are most likely to reveal potential vulnerabilities as described above are as follows.
52. *Inconsistencies* (e.g. ADV\_HLD.2-2) may help realise any attack, depending on the nature of the inconsistency.
53. *Description of TSF in terms of subsystems* (e.g. ADV\_HLD.2-3) is not a particular source of vulnerabilities, but more a prerequisite to evaluator understanding.
54. *Description of the subsystem security functionality* (e.g. ADV\_HLD.2-4) identifies the security responsibilities of a subsystem and possible effects e.g. on security databases. The dependencies of the SFs can be determined, and the potential for manipulation of those dependencies established. The evaluators should also be able to formulate a picture of the security architecture, whereby it is established where SFs are implemented, and where the low-level functions that handle the assets to be protected are implemented. This may highlight the potential for bypass attacks which the next work units may consider.
55. *Hardware, firmware and software requirements* (e.g. ADV\_HLD.2-5) and *presentation of functions of supporting protection mechanisms* (e.g. ADV\_HLD.2-6) are not direct sources of vulnerability, but provide a context in which certain attacks (bypassing, tampering) may be ruled out.
56. *Details of subsystem interfaces* (e.g. ADV\_HLD.2-7 through ADV\_HLD.2-9) complement earlier checks by detailing potential interrelationships between subsystems. The evaluators should investigate the availability of interfaces to subsystems that implement low-level functions. If such interfaces are externally visible, it may be possible to bypass any SFs that the ST requires be invoked prior to the execution of such functionality. Also worthy of investigation are apparent sequences that could be manipulated, e.g. subsystems that are processes could be killed, or affected by scheduling functions.
57. *Description of separation* (e.g. ADV\_HLD.2-10) may rely on protection mechanisms. Evaluators should note in particular any assumptions that might be undermined by detail introduced at lower representational levels (if the LLD and code are to be evaluated).
58. *Traceability analysis* (e.g. ADV\_HLD.2.2E) considers subsystem functionality and interrelationships i.e. how subsystems cooperate to provide an SF. This builds on the details already examined, considering the 'overall picture' of how the security functions are provided.

#### *Low-Level Design*

59. This TSF representation continues the decomposition of the TSF to the granularity of individual modules that can be related to the source code implementation of the SFs. The search for vulnerabilities follows similar principles to that described above for the HLD, focusing on the additional detail introduced as a result of the refinement process.
60. The following cases are of particular interest when performing the ADV\_LLD evaluator actions, and as such merit further investigation:
  - a) **Availability of interfaces to 'low-level' modules.** In this context, a 'low-level' module is one that implements 'low level' functions that are relevant to security (in

other words, functionality that is subject to one or more SFs). An example of such modules would be those that read data from, or write data to, a file, where the ST requires enforcement of file access controls. The availability of the interfaces to modules that implement such 'low-level' functions should therefore be considered for potential bypass of the relevant SFs (which will, typically, be implemented by different modules).

- b) **Modules that provide externally visible interfaces.** Whilst external interfaces will have been considered in the FSP, the internal LLD for such modules may highlight the possibility of attacks that are not visible at the higher level. This may include inadequate input parameter validation (suggesting possible *interface input* attacks), or the processing of such inputs may suggest ways of constructing inputs to achieve some unexpected purpose, bypassing an SF.
  - c) **Storage of TSF data relied upon by modules.** The LLD should describe any global data structures used to provide storage for temporary copies of TSF data. The potential for attackers gaining access to such storage areas should be investigated.
  - d) **Shared resources.** The LLD should identify resources shared between different modules. The evaluators should consider the potential afforded by accesses to such resources, such as illicit access to sensitive user or TSF data, or manipulating the content of such resources to affect the operation of another process or cause an illicit data flow.
61. It may be noted that *module* sequence attacks are not explicitly considered here since these are not (or should not) be susceptible to manipulation, being by definition internal sequences that are under the control of the TSF. This contrasts with the situation in the HLD representation, where the subsystems in an identified 'sequence' may have some external manifestation that can be manipulated, e.g. processes are visible to user, or there may be physical interfaces between TOE components.
62. The ADV\_LLD work units that are most likely to reveal potential vulnerabilities as described above are as follows.
63. *Description of module purpose and functionality* (e.g. ADV\_LLD.1-4 and ADV\_LLD.1-6) will inform the evaluator as to how an SF or security relevant functionality is provided, highlighting dependencies (e.g. data structures in which TSF data is stored) that might be undermined. This information may also reveal unexpected effects of the execution of a module that might be exploited, e.g. to interfere with the operation of an SF. Descriptions of how input parameters are handled may reveal the potential for attack through the external interfaces (e.g. providing unexpected input, or attempted use of an interface for an unexpected purpose).
64. *Description of module interrelationships* (e.g. ADV\_LLD.1-5) may help identify the potential for bypassing attacks (possibly revealing routes through a series of module calls that may lead to compromise of an asset without the effective enforcement of security) or tampering attacks (where more indirect interrelationships are concerned, for example through global data structures that could offer the potential for interference with an SF).
65. *Identification and description of module interfaces* (e.g. ADV\_LLD.1-7 and ADV\_LLD.1-9) may provide a supporting role in helping to identify possible attack methods, e.g. possible

interference with SFs through the unexpected manipulation of data structures containing TSF data.

66. *Identification of externally visible interfaces* (e.g. ADV\_LLD.1-8) helps link the LLD to the FSP, and thus helps to identify possible susceptibility to attacks based on the supply of inputs to the TSF, e.g. thorough inadequate parameter validation.
67. *Traceability analysis* (e.g. ADV\_LLD.1.2E) considers module functionality and interrelationships i.e. how modules cooperate to provide an SF. As with the HLD traceability analysis, this evaluator action builds on the details already examined, considering the 'overall picture' of how the security functions are provided.

### *Code Analysis*

68. In this section we consider vulnerabilities that might be found in the implementation representation. The description which follows is presented in terms of code analysis, although examination of hardware drawings might provide similar approaches.
69. The code implementation is closely linked with the LLD: the LLD provides an overview of the implementation representation that is uncluttered by implementation details, particularly from the perspective of module interrelationships. The LLD may also point to those parts of the code that are likeliest to contain vulnerabilities, e.g. as a result of highlighting areas where the design is unduly complex.
70. As with the search of other TSF representations, the focus at this level is on the potential exploitation of implementation detail not visible at higher levels.
71. The following cases are of particular interest when performing the ADV\_IMP evaluator actions, and as such merit further investigation:
  - a) **Code implementation of externally visible interfaces.** Evaluators should examine the code for external TSF interfaces to determine whether there is a lack of adequate parameter validation or error handling. Such vulnerabilities could help realise *interface input* attacks.
  - b) **Storage of TSF data.** The LLD should describe any global data structures used to provide storage for temporary copies of TSF data. The potential for attackers gaining access to such storage areas should be investigated at the code level.
  - c) **Shared resources.** The LLD should identify resources shared between different modules. The evaluators should consider the potential afforded by accesses to such resources, where this is visible at the code level, e.g. illicit access to sensitive user or TSF data, or manipulating the content of such resources to affect the operation of another process or cause an illicit data flow.
  - d) **Validation checks on TSF data.** Evaluators should investigate whether the code authors have made any unreasonable assumptions (whether explicit or implicit) of prior validation of TSF data that has been directly input by the user, or read from a file or database in which it is stored. The consequences of invalidating such assumptions should be investigated.
  - e) **Security critical code routines.** It is expected that such routines (those that lie at the

‘heart’ of the implementation of any SF, e.g. implementing access control rules, or collating and formatting audit information into an audit record) will be examined as part of the code sampling. The implementation detail may reveal problems that could give rise to vulnerabilities.

72. The ADV\_IMP work units that are most likely to reveal potential vulnerabilities as described above are as follows.
73. *Inconsistencies* (e.g. ADV\_IMP.1-3) may help identify a variety of vulnerabilities arising from inconsistent handling of data (e.g. input parameters, variables holding TSF data). It may be noted that the CEM does not give much guidance on the types of inconsistency to search for; the above list may be used to give focus to the search required by this work unit. Fundamentally, of course, evaluator resources are best deployed towards searching for those inconsistencies that are most likely to result in a vulnerability.
74. *Traceability analysis* (e.g. ADV\_IMP.1-4) may help identify vulnerabilities arising from the introduction of implementation detail, whether it be those that arise from an unduly complex implementation of the algorithm underpinning an SF, or through additional implementation detail that undermines explicit or implicit assumptions made in the LLD.