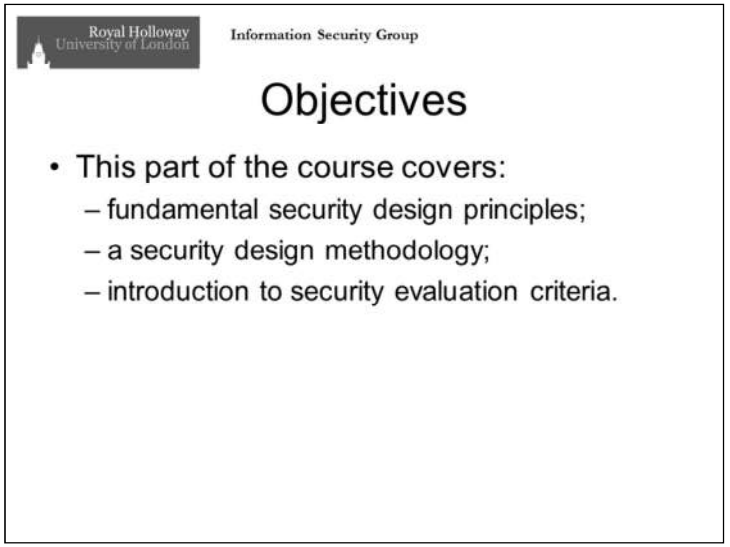
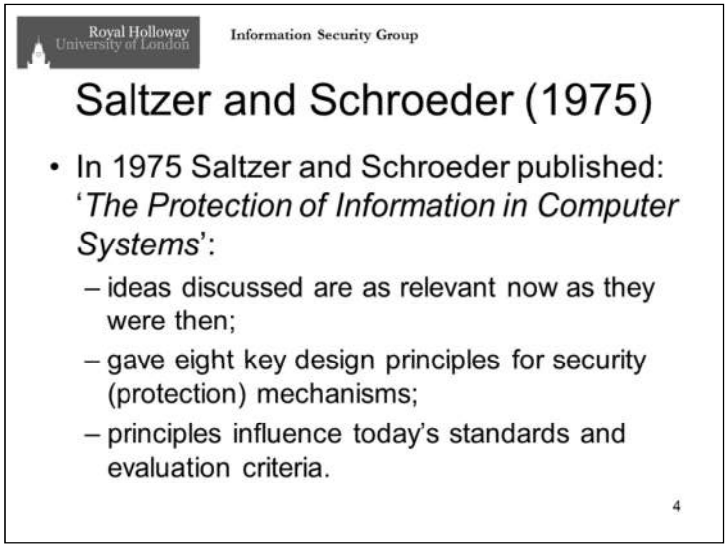
IY5512: Part 2



In this part of the course we cover the following topics. We:

- describe a set of fundamental security design principles (following the approach of Saltzer and Schroeder);
- give an example of a security design methodology;
- provide a (brief) introduction to security evaluation criteria from Orange Book to the Common Criteria.

We start by introducing the fundamental security design principles of Saltzer and Schroeder. Much of the terminology and many of the fundamental ideas they introduced have become very widely used.



In 1975, Saltzer and Schroeder published the paper: '*The Protection of Information in Computer Systems*', which has been enormously influential on thinking about computer security in the almost 40 years since then. Many of the ideas they discuss are as relevant today as they were then.

They describe eight design principles that are of particular relevance when designing security (protection) mechanisms. The influence of these principles is still felt in today's standards and evaluation criteria.

Their paper is available at:

http://www.cs.virginia.edu/~evans/cs551/saltzer/

and

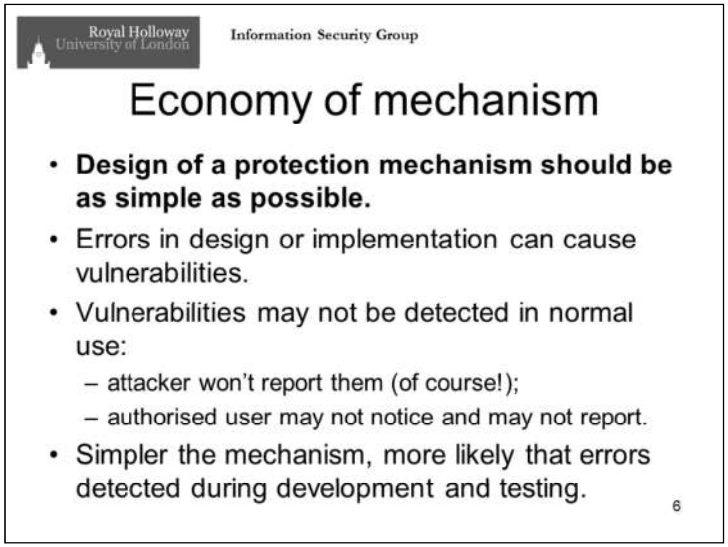
http://www.cap-lore.com/CapTheory/ProtInf/

[This is the same Schroeder (Mike Schroeder) as in the Needham-Schroeder security protocols. He is also a leading expert on the American landscape painter Gilbert Munger.]

The eight Saltzer-Schroeder principles are as follows:

1. Economy of mechanism
2. Fail-safe defaults
3. Complete mediation
4. Open design
5. Least privilege
6. Least common mechanism
7. Separation of privilege
8. Ease of use

We look at each principle in greater detail.

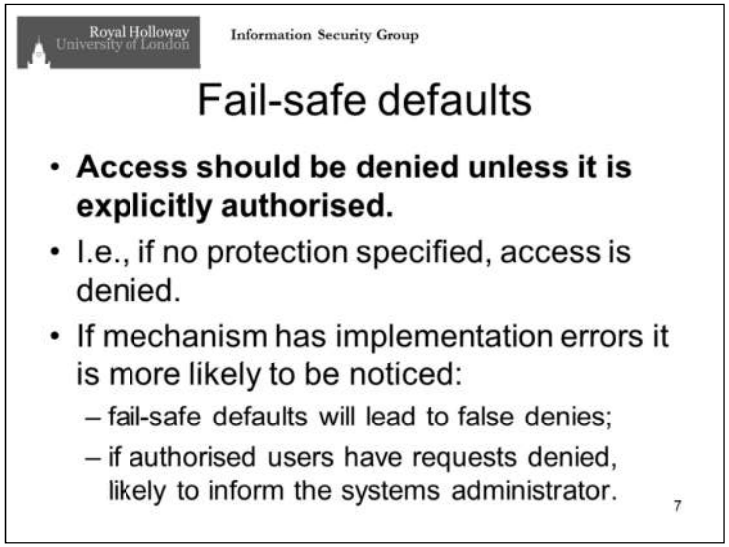


The design of a protection mechanism should be as simple as possible.

Errors in design or implementation can cause vulnerabilities which may lead to incorrect granting of access to resources. These vulnerabilities may not be noticed during normal use:

- an attacker will not report them, because it is not in the attacker's interest;
- an authorised user may not notice, and even if he/she does notice, he/she may not report it to a security administrator.

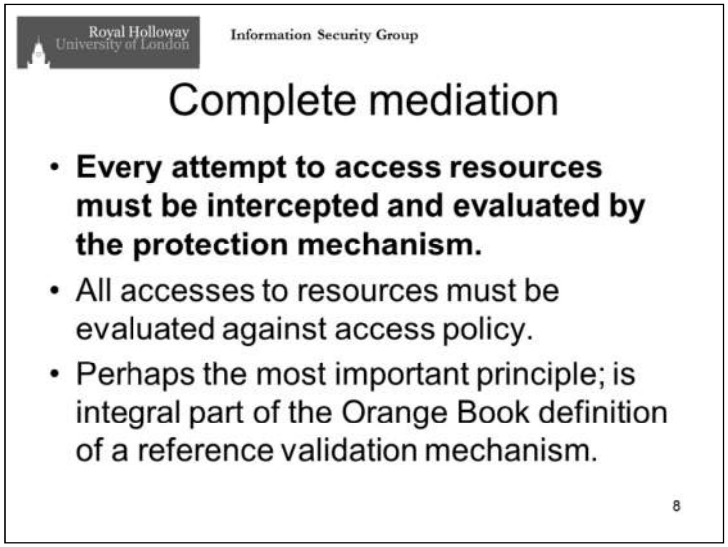
In general, the simpler the mechanism, the more likely it is that errors will be detected during development and testing.



Access should be denied unless it is explicitly authorised.

That is, if no protection is specified for a resource, access should be denied. As a result of following this principle, if the mechanism has implementation errors then they are more likely to be noticed:

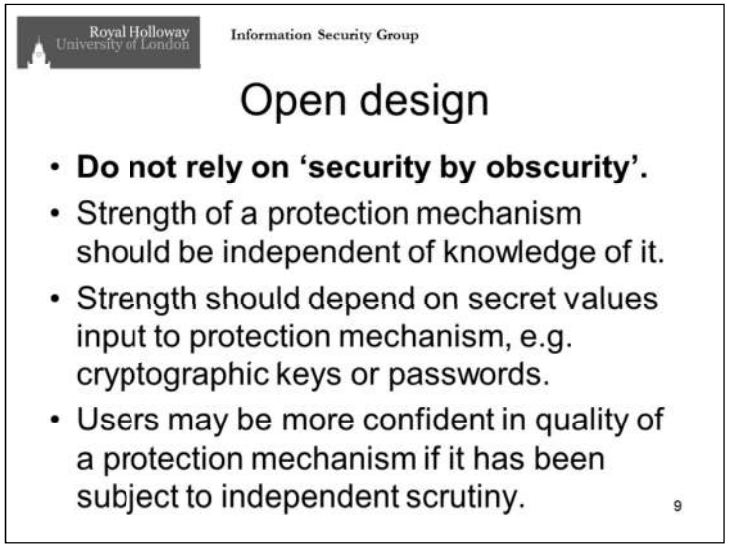
- fail-safe defaults will lead to incorrect denial of access to resources;
- if authorised users have valid requests denied, they are likely to bring it to the attention of the systems administrator.



Every attempt to access resources must be intercepted and evaluated by the protection mechanism.

That is, all accesses to resources must be evaluated against the access policy in force.

This is perhaps the most important principle, and forms an integral part of the definition of a **reference validation mechanism** (i.e. an access control mechanism) in the Orange Book.



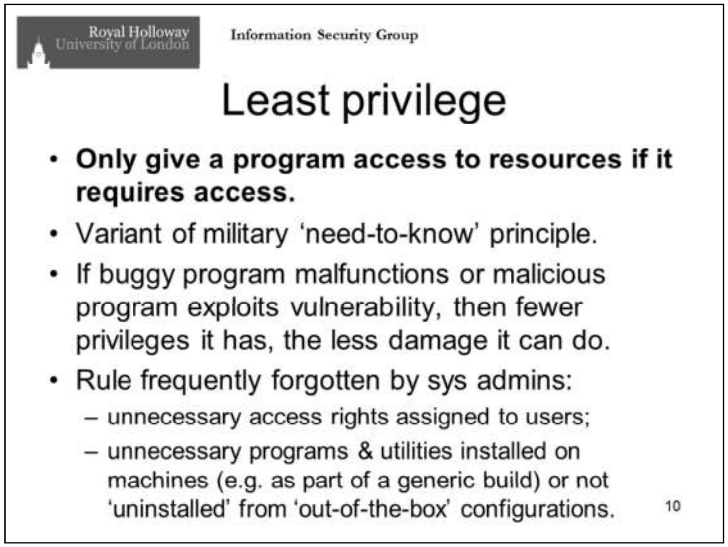
Do not rely on 'security by obscurity'.

That is, the strength of a protection mechanism should be independent of who knows how the mechanism works. Instead of secrecy, the strength of the mechanism should depend on the secrecy and strength of the secret values used as input to the protection mechanism, such as cryptographic keys or passwords.

Users are likely to feel more confident in the quality of a protection mechanism if it has been subject to independent scrutiny and been found to be secure.

This does not, of course, mean that all security mechanisms must be made public, although it can be argued that the user may gain greater assurance if this is the case. The point is that security should not **rely on** secrecy.

Independent scrutiny does not always mean that the protection mechanism must be public (a testing body may perform an analysis under a non-disclosure agreement, and only publish their findings). Also, sometimes greater security can be obtained by combining secrecy with good design – after all, if the design of mechanism is secret than this makes it harder to break. Indeed, obfuscation is an established area of study within security. Moreover, in some cases obscurity is the main form of defence, although this should perhaps be a 'last resort'.



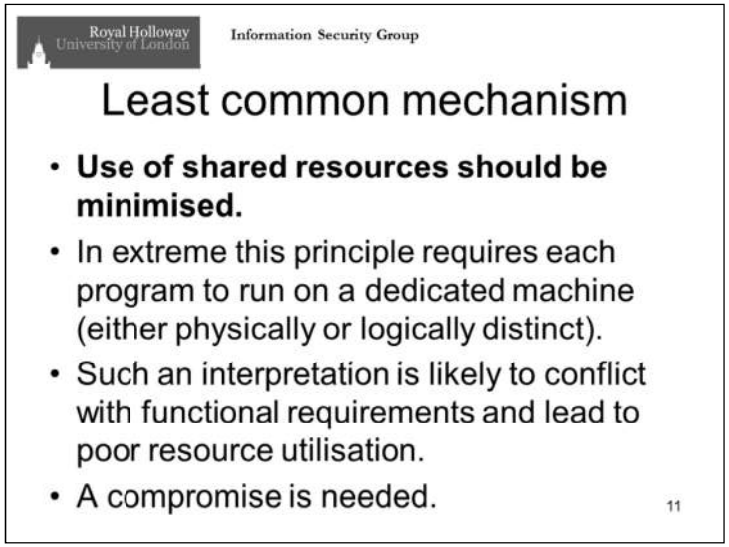
Only give a program access to resources if it requires access.

This is a variant of the military 'need-to-know' principle. If an incorrect program malfunctions, or a malicious program exploits a vulnerability, then the fewer privileges it has, the less damage it can do.

This lesson is frequently forgotten by system administrators:

• unnecessary access rights are assigned to users (and are not removed when they are no longer needed);

• unnecessary programs and utilities are installed on machines (e.g. as part of a generic build) or not 'uninstalled' from an 'out-of-the-box' configuration.



The use of shared resources should be minimised.

Taken to its extreme this principle requires that each program should run on its own dedicated machine (either physically or logically distinct). Clearly such an extreme interpretation is likely to conflict with functional requirements and lead to poor resource utilisation.

Ultimately must compromise between efficiency and security.

Wherever possible two or more independent checks should be used to confirm that a request is authorised.

Thus wherever possible a system should use two or more security mechanisms to protect a resource. For example:

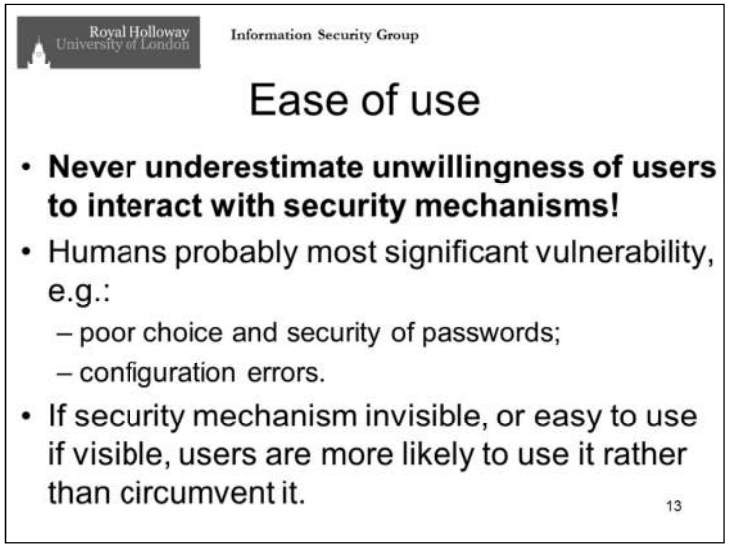
• deploy a security guard at the front door of a building and also put locks on the doors within the building;

• use two-factor authentication.

This relates closely to the notion of **dual control**: it should be impossible for a single user to perform a sequence of mission- or business-critical actions. For example:

• two different generals must separately arm and launch a nuclear missile;

• two different individuals must separately authorise cheques over £5000.

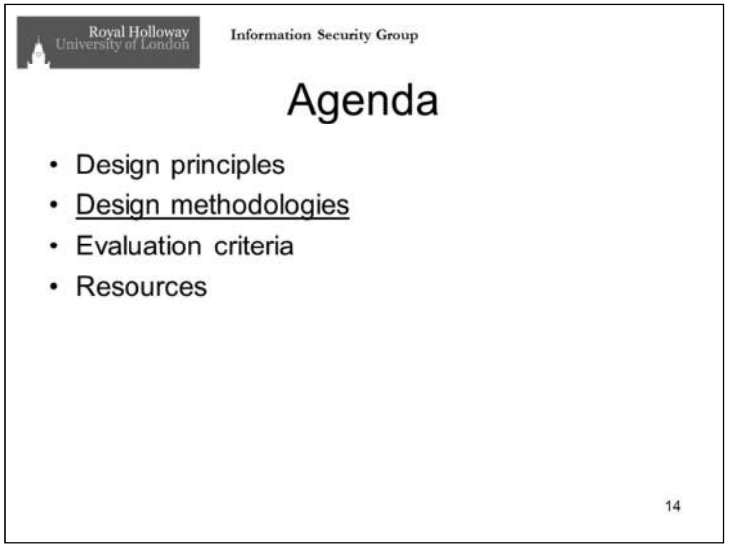


Never underestimate the unwillingness of users to interact with security mechanisms!

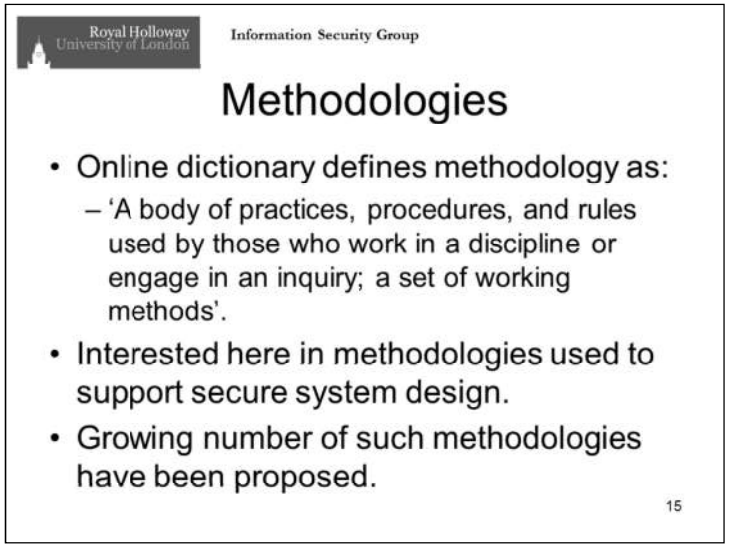
The human element of computer systems is probably the most significant vulnerability. For example:

- users and likely to choose poor passwords and also to manage them poorly;
- users are likely to make system configuration errors;
- 'good' users can be persuaded to run malicious software or do other damage to their own system.

If a security mechanism is invisible, or if it is easy to use when visible, then users are more likely to use it rather than circumvent it. On the other hand, if a security mechanism is highly intrusive, then users are likely to find ways to bypass it.



We next give an example of a security design methodology. This is by no means the only example of such a methodology, but it is one that has received a lot of attention over the last five-ten years.

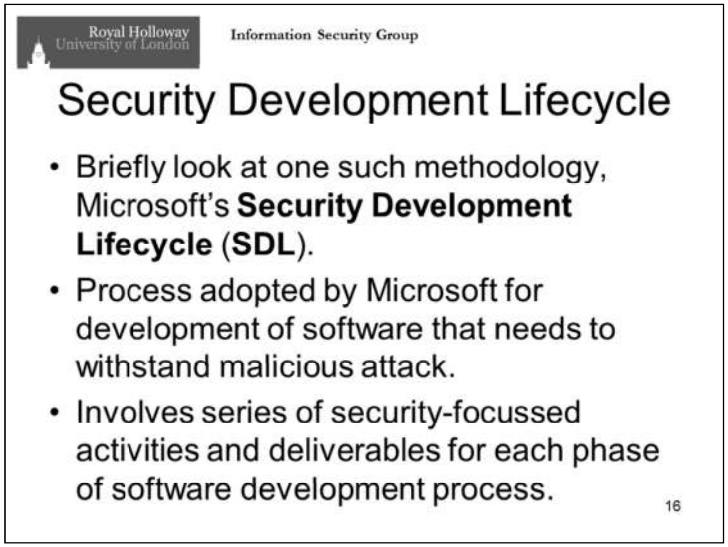


An online dictionary (http://www.thefreedictionary.com/methodology) defines a methodology as:

> 'A body of practices, procedures, and rules used by those who work in a discipline or engage in an inquiry; a set of working methods'.

We are interested here in methodologies used to support secure system design.

A growing number of such methodologies have been proposed over the last few years.



We briefly look at one such methodology, namely Microsoft's **Security Development Lifecycle** (**SDL**).

This is a process that Microsoft has adopted for the development of software that needs to withstand malicious attack. It was developed as part of the Microsoft Trustworthy Computing initiative, launched in the early 2000s with the goal of addressing Microsoft's then very poor reputation for providing reliable and trustworthy software. Some ten years on, this ongoing initiative has transformed the company's internal software development practices – see:

http://www.microsoft.com/en-us/news/features/2012/jan12/01-12TwC.aspx

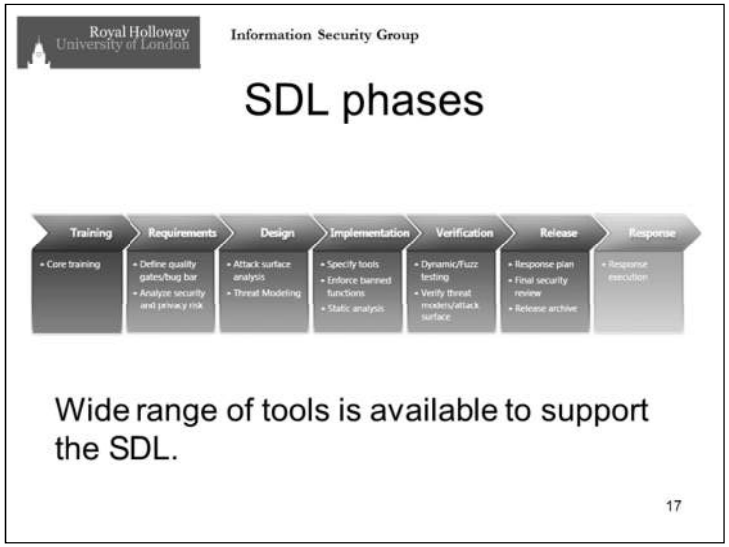
It involves a series of security-focussed activities and deliverables for each phase of the software development process.

An entire website is devoted to the SDL – see:

http://www.microsoft.com/security/sdl/

For general background on the development of the SDL, see '*The Trustworthy Computing Security Development Lifecycle*' by Steve Lipner and Michael Howard:

http://msdn.microsoft.com/en-us/library/ms995349.aspx



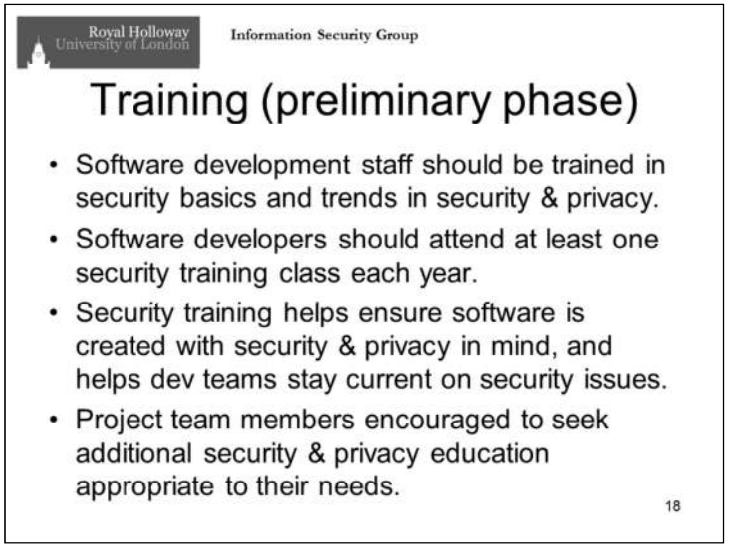
This picture (taken from the SDL website) shows the five main phases of the SDL (shown in green).

Microsoft have published a wide range of free tools designed to support various phases of the lifecycle, see:

http://www.microsoft.com/security/sdl/getstarted/tools.aspx

These include:

- the SDL process template – for the requirement and release phases;
- the SDL threat modelling tool – for the design phase;
- a range of code analysis tools and libraries which assist developers in writing more secure code – for the implementation phase;
- a range of software vulnerability analysis tools (including a fuzzer) – for the verification phase.

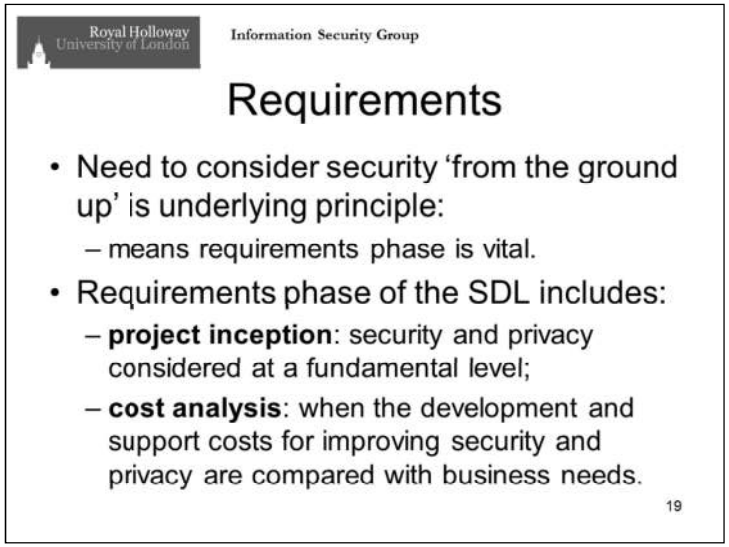


All members of a software development team should receive appropriate training to stay informed about security basics and recent trends in security and privacy.

Individuals who develop software (software developers or simply 'devs') should attend at least one security training class each year.

Security training can help ensure software is created with security and privacy in mind, and can also help development teams stay current on security issues.

Project team members are strongly encouraged to seek additional security and privacy education that is appropriate to their needs or products.



The need to consider security 'from the ground up' is a fundamental principle of secure system development. While many development projects produce 'next versions' that build on previous releases, the requirements phase and initial planning of a new release or version offers the best opportunity to build secure software.

The requirements phase of the SDL includes:

• **project inception**: in which security and privacy are considered at a fundamental level;

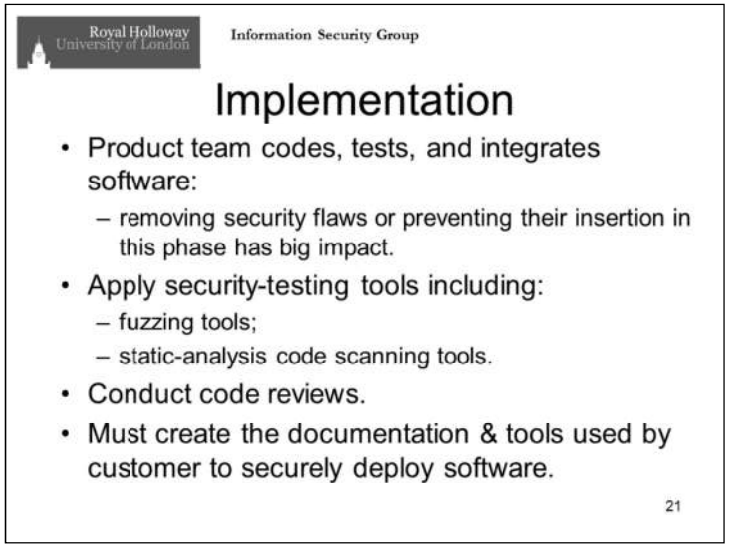
• **cost analysis**: in which the development and support costs for improving security and privacy are compared with the business needs.

The design phase identifies the overall requirements and structure for the software. The design phase involves building the plan for taking the project through the rest of the SDL process, including from implementation, to verification, to release.

During the design phase, the team members must:

• establish best practices that will be followed in the functional and design specifications;

• perform a **risk analysis** (or, in SDL terminology, 'risk modelling') to identify threats to, and vulnerabilities in, the software.



During the implementation phase, the product team codes, tests, and integrates the software. Steps taken to remove security flaws or prevent their initial insertion during this phase have a high impact — they significantly reduce the likelihood that security vulnerabilities will make their way into the final version of the software that is released to customers.

The Implementation phase involves establishing development best practices to detect and remove security and privacy issues early in the development cycle. In this phase, security-testing tools including fuzzing tools should be applied. 'Fuzzing' supplies structured but invalid inputs to software application programming interfaces (APIs) and network interfaces so as to maximize the likelihood of detecting errors that may lead to software vulnerabilities. Static-analysis code scanning tools are also applied that can detect some types of coding flaws that result in vulnerabilities, including buffer overruns, integer overruns, and uninitialized variables. Microsoft has made a major investment in the development of such tools (the two that have been in longest use are known as PREfix and PREfast) and continually enhances those tools as new kinds of coding flaws and software vulnerabilities are discovered.

Code reviews should be conducted, which supplement automated tools and tests by applying the efforts of trained developers to examine source code and detect and remove potential security vulnerabilities. They are a crucial step in the process of removing security vulnerabilities from software during the development process.

In this phase, the development team must create the documentation and tools to be used by the customer to make informed decisions about secure deployment of the software.

IY5512: Part 2

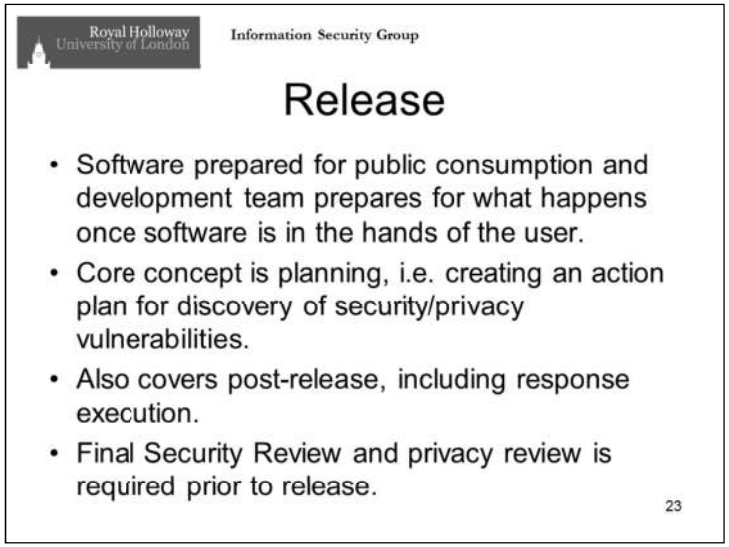
The verification phase is the point at which the software is functionally complete and enters user beta testing. During this phase, while the software is undergoing beta testing, the product team conducts a "security push" that includes security code reviews beyond those completed in the implementation phase as well as focused security testing.

The verification phase involves ensuring that the code meets the security and privacy rules established in the previous phases. This involves:

• security and privacy testing,

• a security push, i.e. a team-wide focus on updating the threat model, reviewing the code, code testing, and a thorough review and edit of the system documentation.

A public release privacy review is also completed during this phase.



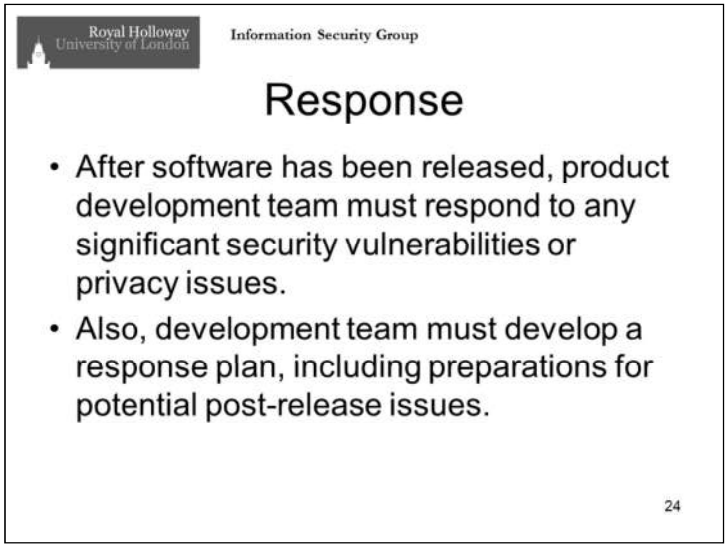
During the release phase, the software should be subject to a Final Security Review (FSR). The goal of the FSR is to answer one question: 'From a security viewpoint, is this software ready to deliver to customers?'. A privacy review is also required prior to release.

During the release phase the software is prepared for public consumption, and the development team prepares for what happens once their software is in the hands of the user.

A core concept for this phase is planning, and in particular the creation of an action plan that should be followed if and when security or privacy vulnerabilities are discovered.

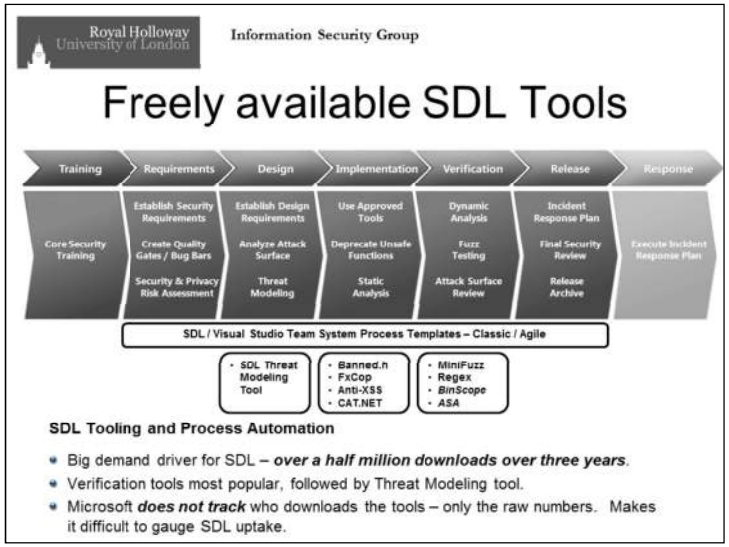
This phase also includes the post-release period, including response execution.

.



After the software has been released, the product development team must be available to respond to any security vulnerabilities or privacy issues that a sufficiently significant to warrant a response.

In addition, the development team must develop a response plan that includes preparations for potential post-release issues.

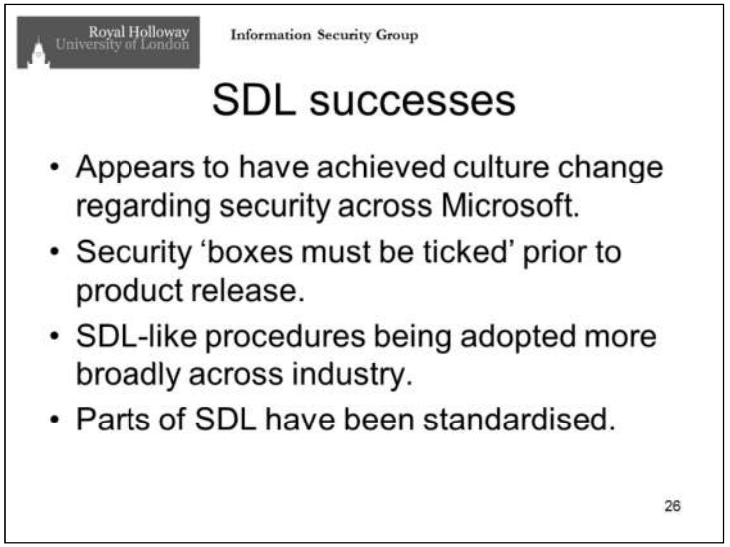


This slide summarises some of the tools which support use of the SDL and that are available for free download.

As of mid 2011, there had been over a half million downloads of SDL tools and materials since April 2008. At that time the verification tools had been the most popular, followed by the Threat Modeling tool. Microsoft does not track who downloads the tools – only the raw numbers. This makes it difficult to gauge the overall industry uptake of SDL.

Likely that numbers have continued to increase since then.

[This slide was taken from a slide deck prepared by David Ladd (a Principal Program Manager at Microsoft) in August 2011.]

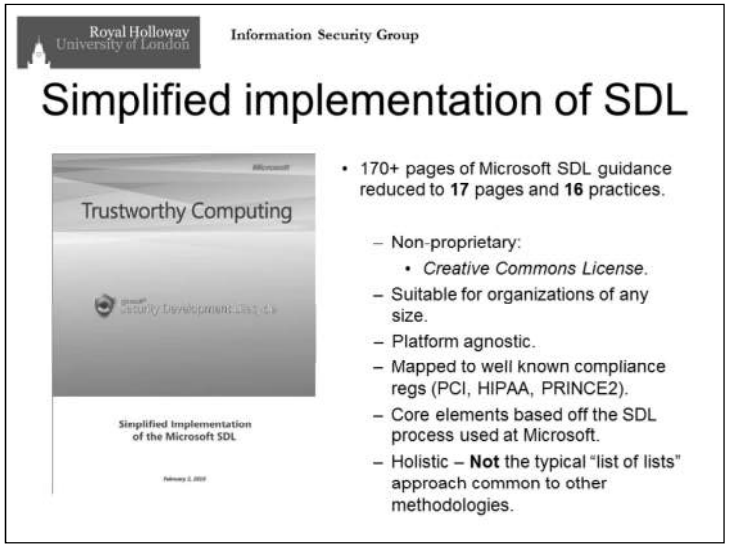


The SDL appears to have achieved a culture change regarding security across large parts of Microsoft.

Prior to release of a major product, the SDL procedures must be completed and the product signed off by the security team.

SDL-like procedures are being adopted more broadly across the software industry.

Parts of the SDL have has a major influence on at least one ISO/IEC standard.



One key SDL publication has been the *Simplified Implementation of the Microsoft SDL*. This reduces the SDL guidance to 17 pages and 16 practices.

It is non-proprietary, and has been released under the Creative Commons Licence. It is designed to be suitable for organisations of any size, is platform agnostic, and has been mapped to well known compliance regulations (including PCI, HIPAA, PRINCE2).

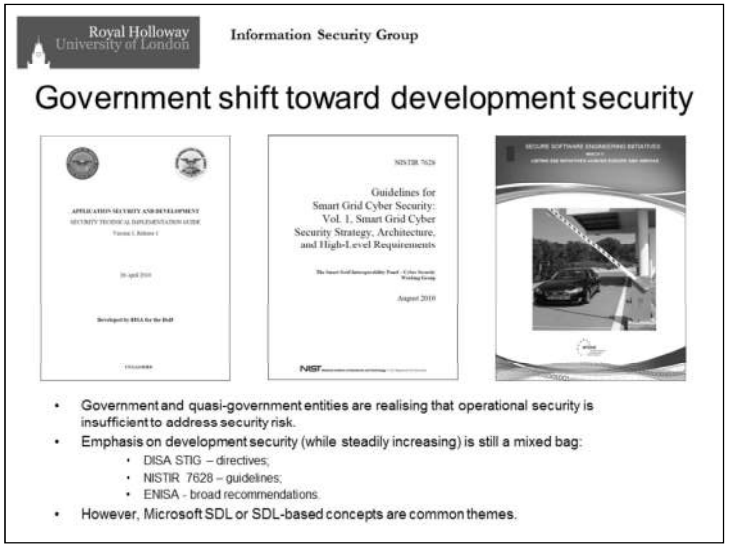
The core elements are based on the SDL process used at Microsoft. It is intended to form a complete and integrated process, and not just a 'list of lists' approach common to other methodologies.

Between April 2008 and mid 2011 there were nearly a quarter million downloads.

[This slide was taken from a slide deck prepared by David Ladd (a Principal Program Manager at Microsoft) in August 2011.]

See the link:

http://www.microsoft.com/en-us/download/details.aspx?id=12379



Government and quasi-government entities have realised that operational security is insufficient to address the complete set of security risks. There is an increasing emphasis on development security, although the picture is still not clear. Initiatives include:

- DISA STIG – directives;
- NISTIR 7628 – guidelines;
- ENISA – broad recommendations.

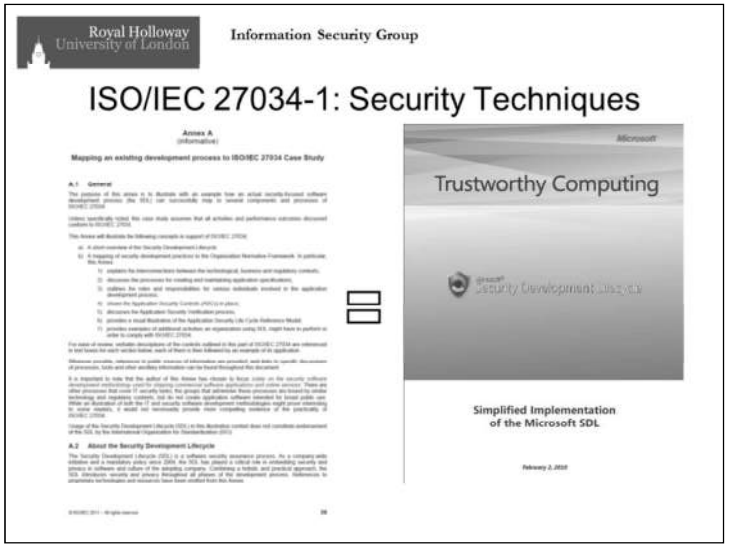
However, Microsoft SDL or SDL-based concepts are common themes.

[This slide was taken from a slide deck prepared by David Ladd (a Principal Program Manager at Microsoft) in August 2011.]

There are three common themes across the diverse industry players shown in the slide:

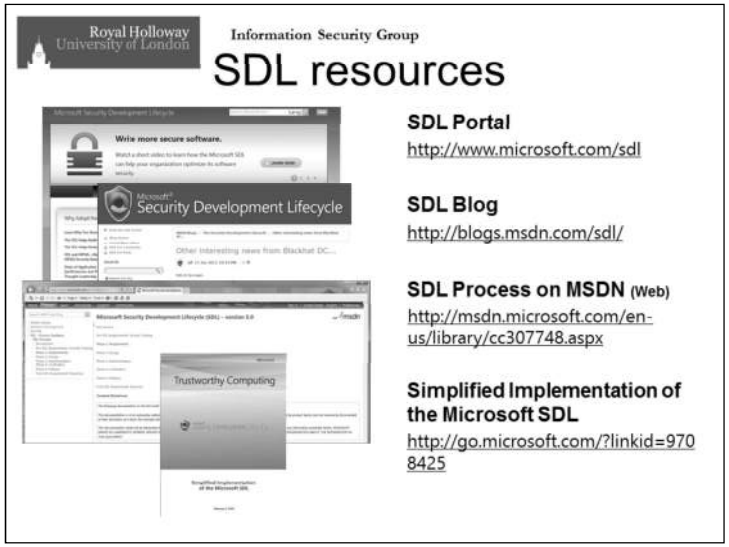
- reduction of risk (for both the company and its customers) is a top priority;
- there is solid executive support for these initiatives;
- demonstrable results have been achieved for both internal and external stakeholders;
- these large companies are starting to push security development concepts on to their suppliers.

[This slide was taken from a slide deck prepared by David Ladd (a Principal Program Manager at Microsoft) in August 2011.]



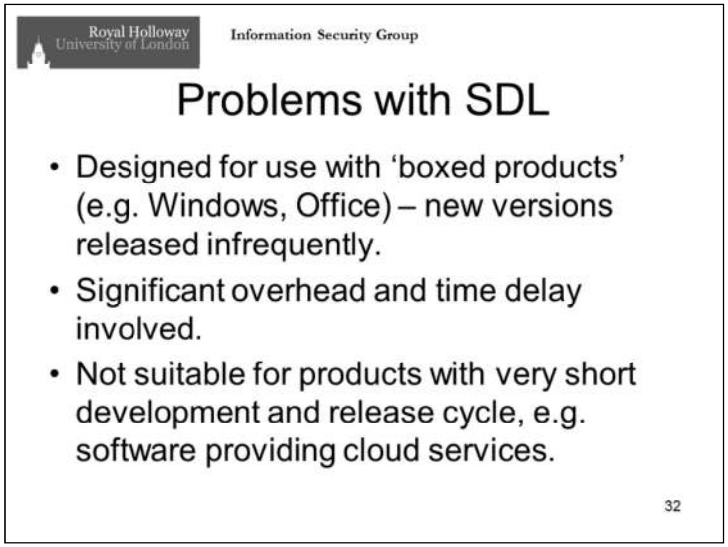
ISO/IEC 27034-1 (Application security – Part 1: Overview and concepts) was published in November 2011. It takes on board many of the ideas developed in the SDL. Indeed, Annex A (shown on the slide), based on a contribution from Microsoft, contains a version of the simplified SDL, although this is an informative annex (i.e. for information only) rather than a normative annex (i.e. forming part of the standard).

[This slide is loosely based on one taken from a slide deck prepared by David Ladd (a Principal Program Manager at Microsoft) in August 2011.]



The slide summarises publicly available SDL resources.

[This slide was taken from a slide deck prepared by David Ladd (a Principal Program Manager at Microsoft) in August 2011.]

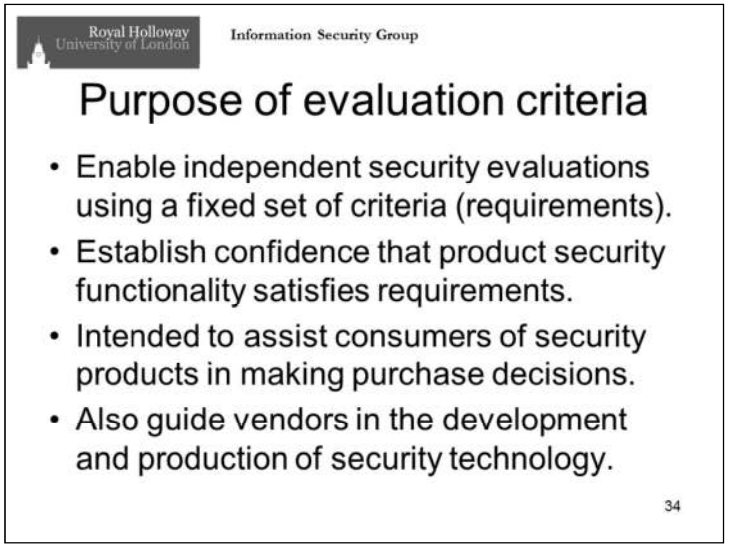


The SDL was designed for use with 'boxed products', e.g. Windows and Office, new versions of which are released only infrequently.

Use of the SDL involves a significant work overhead and time delay.

The SDL is not suitable for products with a very short development and release cycle, e.g. software that is used to provide cloud services, which may be updated almost on a weekly basis.

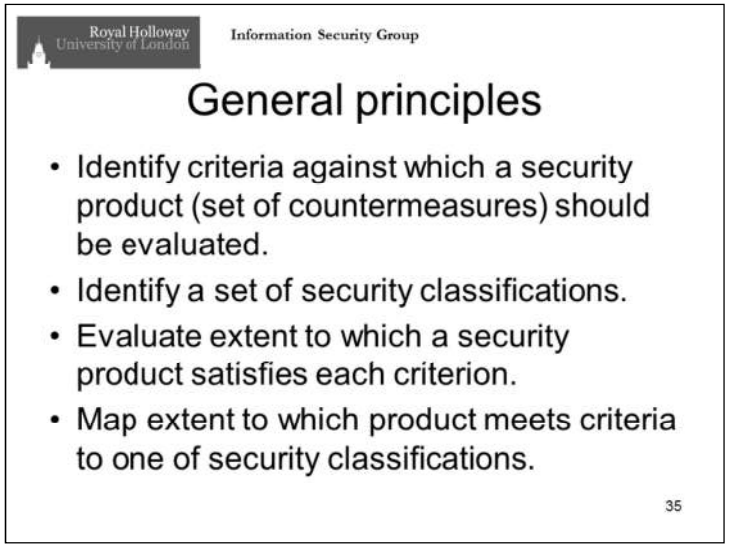
The subject of evaluation criteria has become a large and complex topic in its own right. We provide only a brief introduction to the subject.



The main purpose of evaluation criteria is to provide a basis for independent evaluations of security products using a fixed set of criteria (requirements).

The purpose of such an evaluation is to establish a level of confidence that the security functionality of a product satisfies the criteria. Ultimately, this is intended to assist consumers of security technology in making an informed decision.

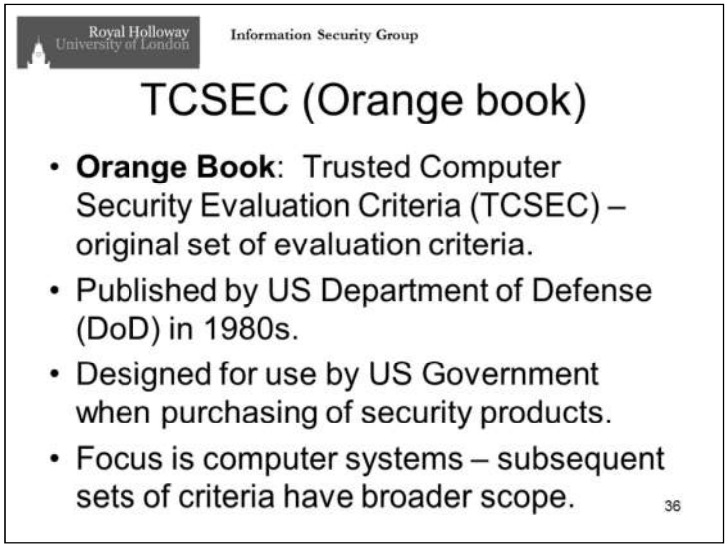
They are are also intended to help guide vendors in the development and production of security technology.



The general principles behind any set of security evaluation criteria (and several sets have been defined, starting with the Orange Book and culminating in the Common Criteria) are as follows.

1. Identify the criteria against which a security product (or, more generally, a set of countermeasures) should be evaluated.
2. Identify a set of security classifications (typically this consists of an ordered sequence of *security levels*).
3. Evaluate the extent to which the security product being evaluated satisfies each criterion.
4. Map the extent to which the product satisfies the criteria to one of the security classifications.

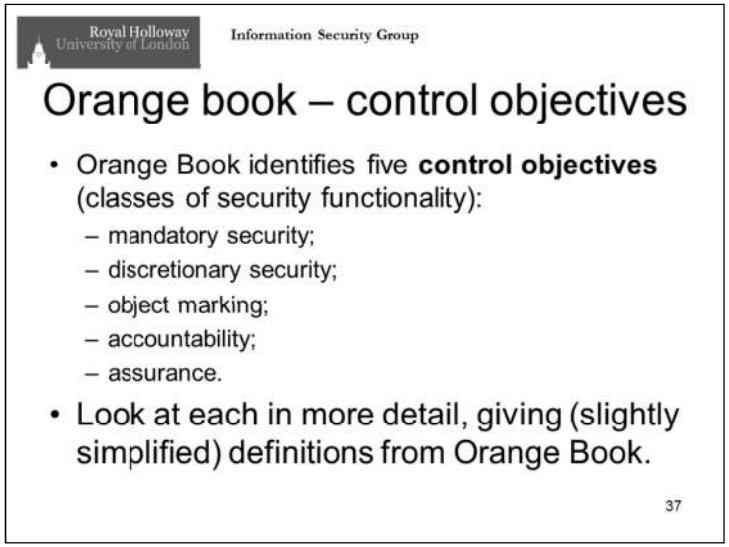
IY5512: Part 2



The **Trusted Computer Security Evaluation Criteria** (**TCSEC)** or the **Orange Book** (so named because of the covers in which the TCSEC were published) were the first set of evaluation criteria. They were published by the US Department of Defense (DoD) in the 1980s (although they reflect ideas dating back to the 1960s).

They were designed for use in assisting US Government departments in purchasing security products.

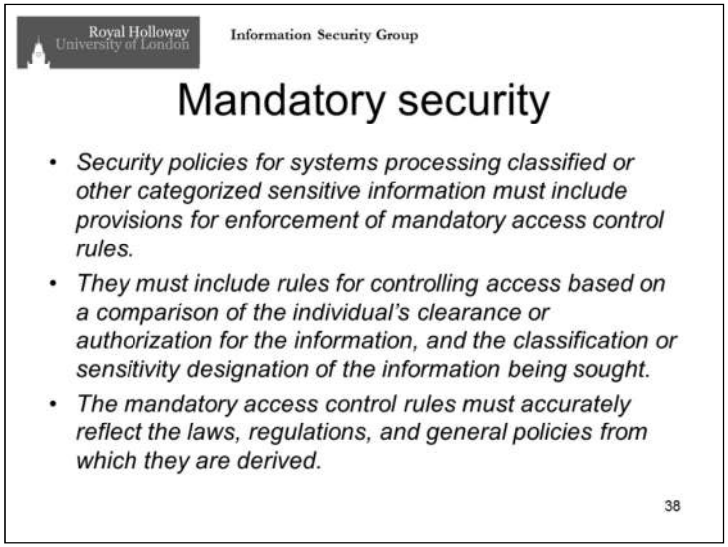
The Orange Book focuses on computer systems. Subsequently published sets of evaluation criteria have broader scope (e.g. anti-virus technology, firewalls, biometrics, etc.).



The Orange Book identifies five **control objectives** (i.e. desirable classes of security functionality), as follows:

- mandatory security;
- discretionary security;
- object marking;
- accountability;
- assurance.

We next look at each of these objectives in greater detail, in each case quoting the definitions given in the Orange Book.

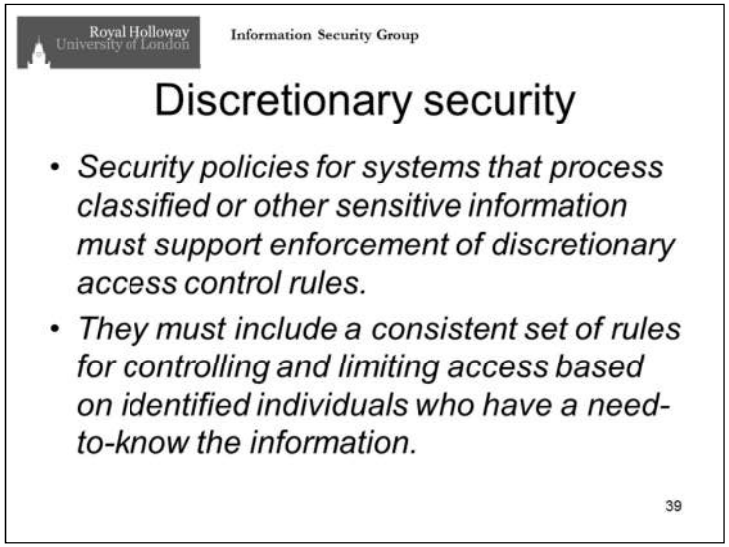


The following definition is taken from section 5.3.1.1 of the Orange Book.

Security policies defined for systems that are used to process classified or other specifically categorized sensitive information must include provisions for the enforcement of mandatory access control rules.

That is, they must include a set of rules for controlling access based directly on a comparison of the individual's clearance or authorization for the information and the classification or sensitivity designation of the information being sought …

The mandatory access control rules must accurately reflect the laws, regulations, and general policies from which they are derived.



The following definition is taken from section 5.3.1.2 of the Orange Book.

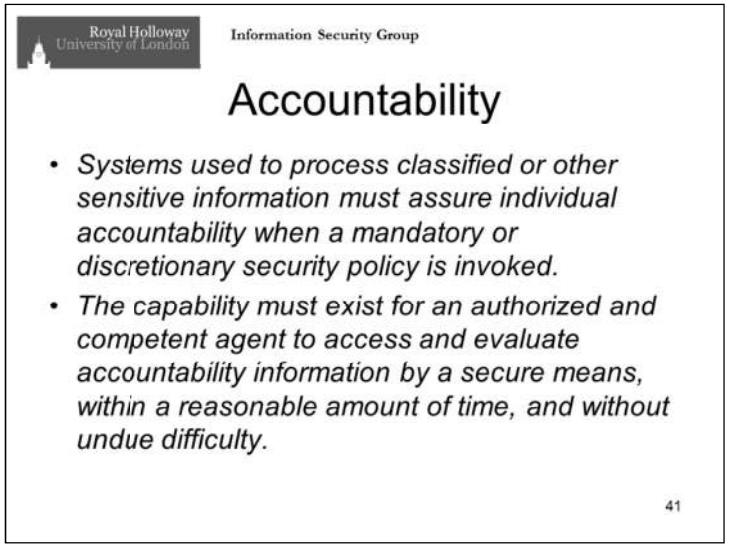
Security policies defined for systems that are used to process classified or other sensitive information must include provisions for the enforcement of discretionary access control rules.

That is, they must include a consistent set of rules for controlling and limiting access based on identified individuals who have … a need-to-know … the information.

The following definition is taken from section 5.3.1.3 of the Orange Book.

Systems that are designed to enforce a mandatory security policy must store and preserve the integrity of classification or other sensitivity labels for all information.

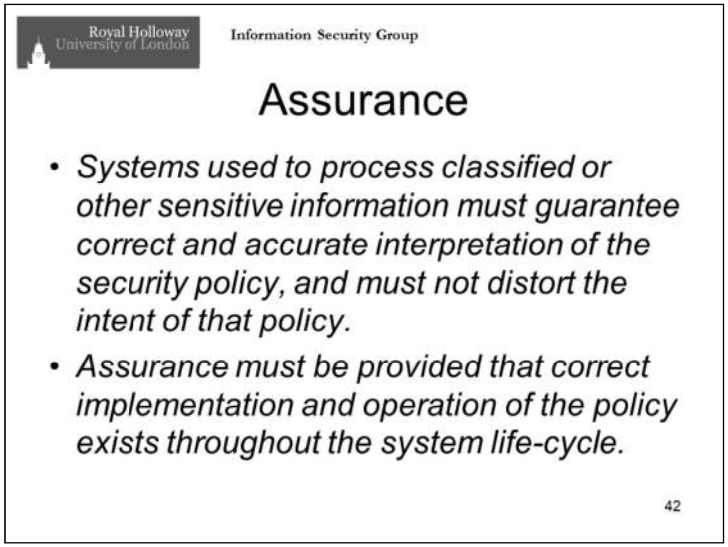
Labels exported from the system must be accurate representations of the corresponding internal sensitivity labels being exported.



The following definition is taken from section 5.3.2 of the Orange Book.

Systems that are used to process … classified or other sensitive information must assure individual accountability when … a mandatory or discretionary security policy is invoked.

Furthermore, to assure accountability, the capability must exist for an authorized and competent agent to access and evaluate accountability information by a secure means, within a reasonable amount of time, and without undue difficulty.



The following definition is taken from section 5.3.3 of the Orange Book.

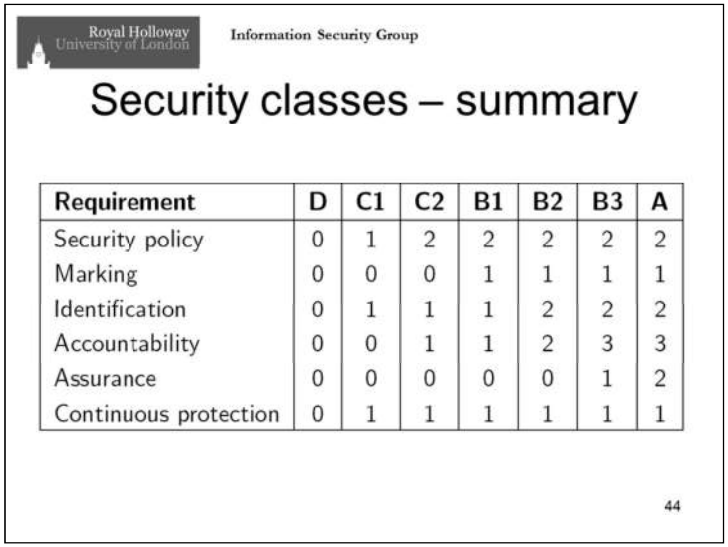
Systems that are used to process or handle classified or other sensitive information must be designed to guarantee correct and accurate interpretation of the security policy and must not distort the intent of that policy.

Assurance must be provided that correct implementation and operation of the policy exists throughout the system's life-cycle.

The five control objectives are used to define six security requirements, covering the following topics.

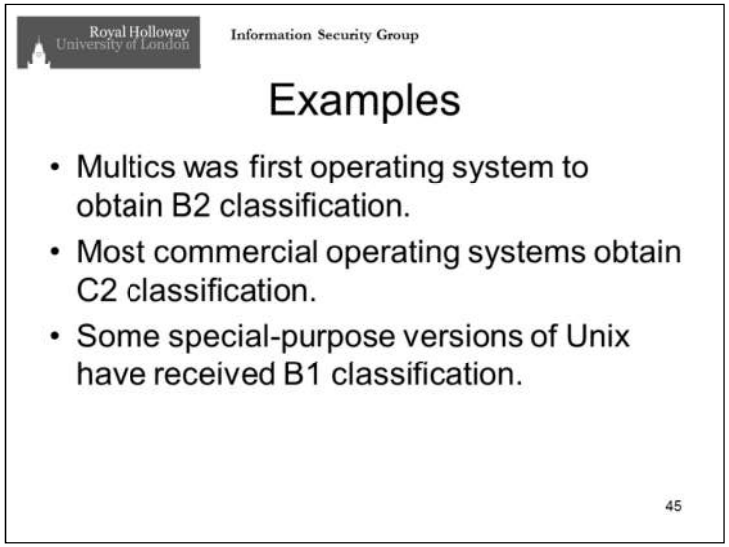
- Security policy;
- Marking;
- Identification;
- Accountability;
- Assurance;
- Continuous protection.

IY5512: Part 2



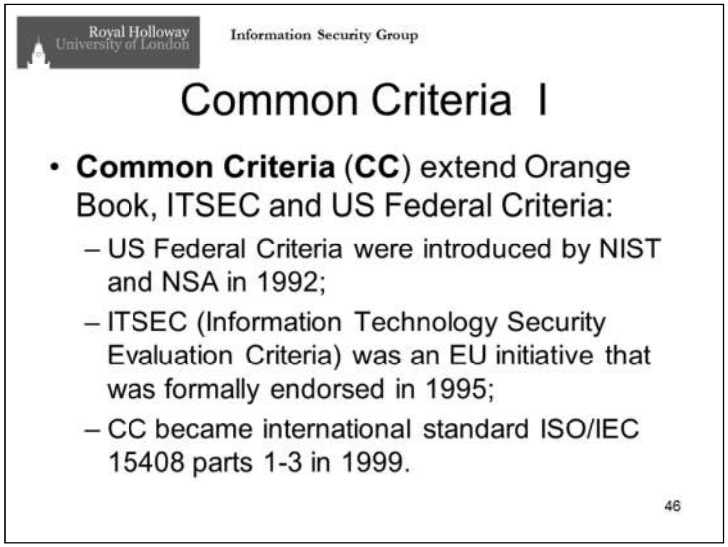
Requirement	D	C1	C2	B1	B2	B3	A
Security policy	0	1	2	2	2	2	2
Marking	0	0	0	1	1	1	1
Identification	0	1	1	1	2	2	2
Accountability	0	0	1	1	2	3	3
Assurance	0	0	0	0	0	1	2
Continuous protection	0	1	1	1	1	1	1

This table summarises the six protection classes defined in the Orange book, ranging from D (no protection) to A (highest level of protection). The level of requirements of each type (0=no requirement) in order to obtain each classification level is indicated in the table entries. That is, the higher the number indicated, the higher the degree to which the requirement must be met in order to obtain the classification.



Multics was the first operating system to obtain the B2 classification. Most commercial operating systems obtain the C2 classification.

Some special-purpose versions of Unix have received a B1 classification.

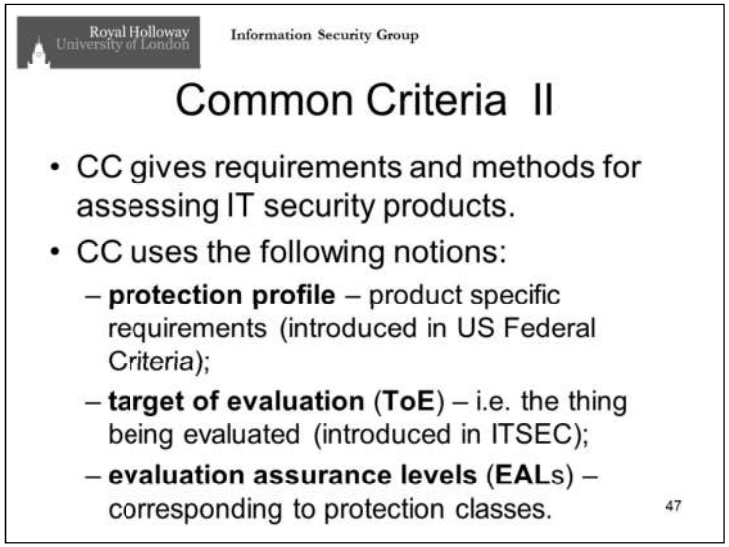


The **Common Criteria** (**CC**) extend the Orange Book, ITSEC and the US Federal Criteria:

• the US Federal Criteria were introduced by NIST and NSA in 1992 (to replace the Orange Book);

• ITSEC (Information Technology Security Evaluation Criteria) was an EU initiative that was formally endorsed in 1995;

• the CC, developed as a result of an international collaborative effort (the Common Criteria Editorial Board, or CCEB) involving the US, Europe and Asia, became international standard ISO/IEC 15408 parts 1-3 in 1999.



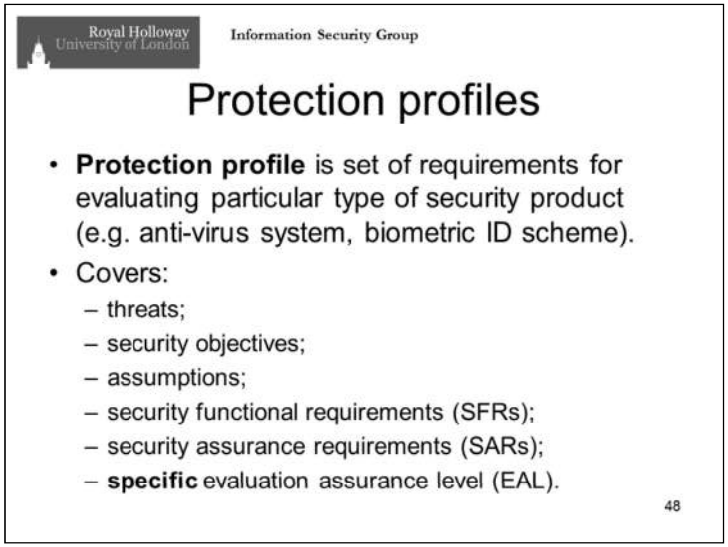
The CC provide a set of requirements and methods for assessing the security functionality of IT products.

The CC use the notions of:

• protection profile – originally introduced in the US Federal Criteria;

• target of evaluation (ToE) – originally introduced in ITSEC, i.e. the product (or, more generally, the set of functionality) being submitted for evaluation;

• evaluation assurance levels (EALs) – corresponding (roughly) to the Orange Book protection classes.



A **protection profile** is a set of requirements designed to enable the evaluation of a security product of a particular type. Examples of categories of security product include anti-virus systems and biometric identification/authentication system, etc.).

A protection profile covers:

• threats to be addressed by product;

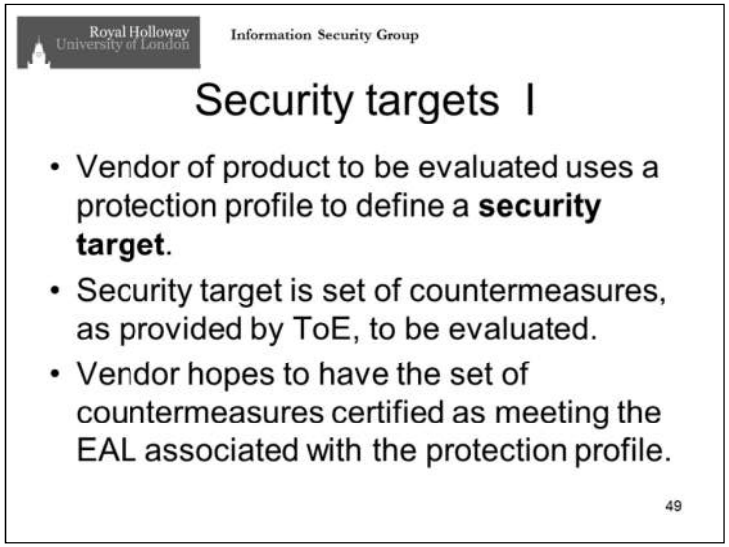
• security objectives of the features of the product;

• assumptions;

• security functional requirements (SFRs);

• security assurance requirements (SARs);

• a specific evaluation assurance level (EAL), i.e. for a particular class of products a different protection profile may exist for each of the EALs defined by the CC.



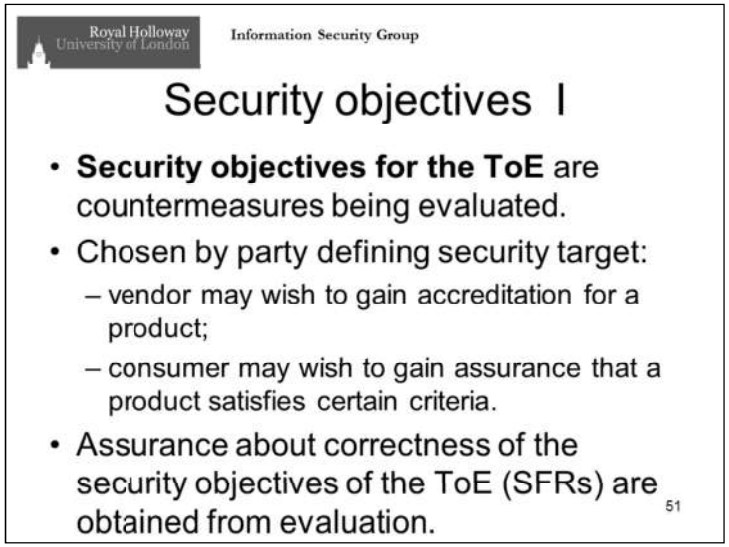
A vendor of a product to be evaluated (the ToE) uses an appropriate protection profile (i.e. one matching the type of product or set of functionality being evaluated) to define a **security target**, which specifies the security functionality of the ToE which is to be evaluated.

The security target specifies a set of countermeasures to security threats, as provided by the ToE.

The goal of the vendor is to have the set of countermeasures certified as meeting the specific EAL associated with the protection profile.

A security target defines:

- the assets to be protected by the ToE;
- the threats to which the assets might be exposed;
- the security objectives for the ToE, also known as the **security functional requirements** (**SFRs**);
- the security objectives for the **operational environment**.

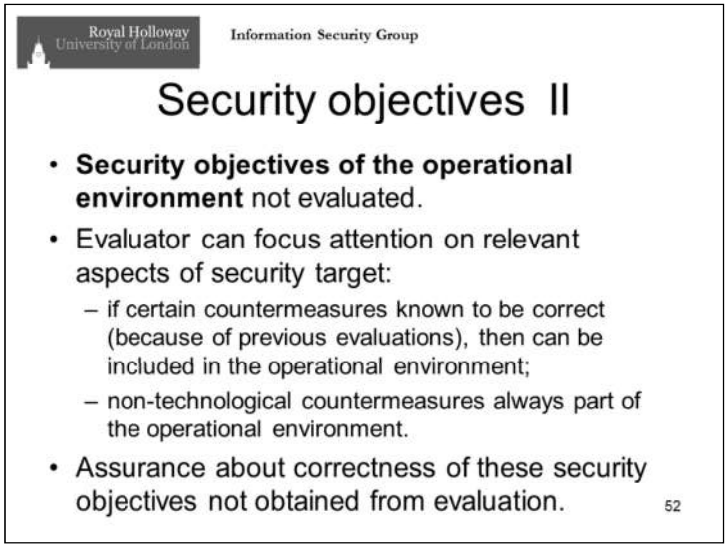


The **security objectives for the ToE** are the countermeasures for which an evaluation is sought. They are chosen by the party that defines the security target. For example:

• a vendor may wish to obtain accreditation for a product;

• a consumer may wish to obtain assurance that a product satisfies certain criteria.

Assurance about the correctness of the security objectives of the ToE (SFRs) are obtained as a result of the evaluation. Here 'correctness' captures a range of notions, including the appropriateness of the countermeasures in meeting the specified threats, the robustness of the chosen security measures (e.g. strength of encryption algorithm), and assurance in implementation.

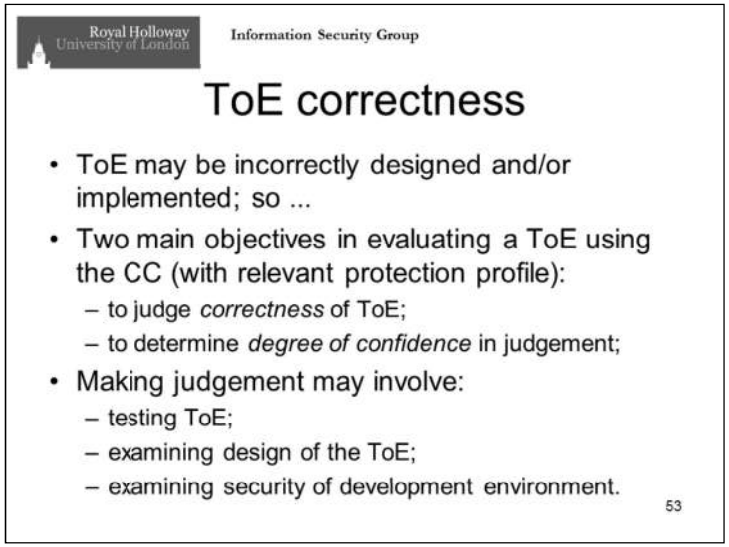


The **security objectives of the operational environment** are not evaluated. They allow the evaluator to focus its attention on those aspects of the security target that are relevant. For example:

• if certain countermeasures are already known to be correct (e.g. because of previous evaluations) then they can be included in the operational environment;

• non-technological countermeasures always form part of the operational environment.

Assurance about the correctness of the security objectives of the operational environment are not obtained as a result of the evaluation.



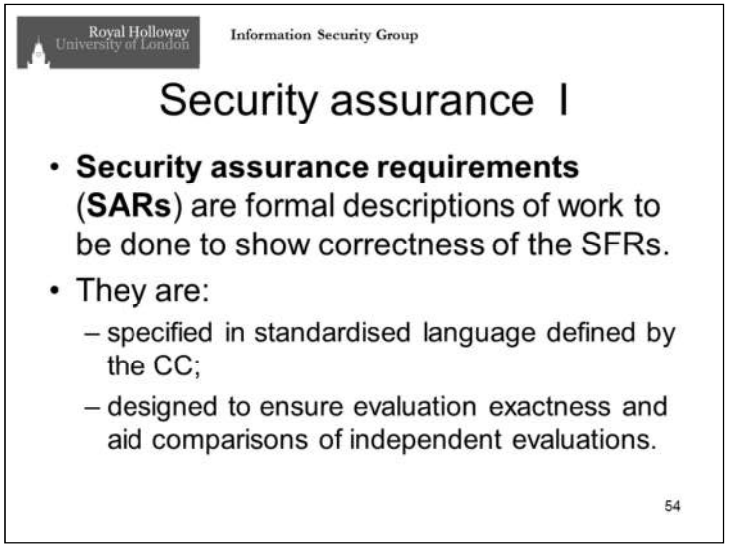
A ToE may be incorrectly designed and/or implemented. As a result, there are two main objectives in evaluating a ToE with respect to the CC (using the relevant protection profile):

- to make a judgement about the correctness of the ToE;
- to determine a degree of confidence in this judgement.

Making a judgement about correctness may involve:

- testing the ToE;
- examining the design of the ToE;
- examining the security of the development environment of the ToE, including looking at the vendor's software development procedures (e.g. do they use the SDL?).

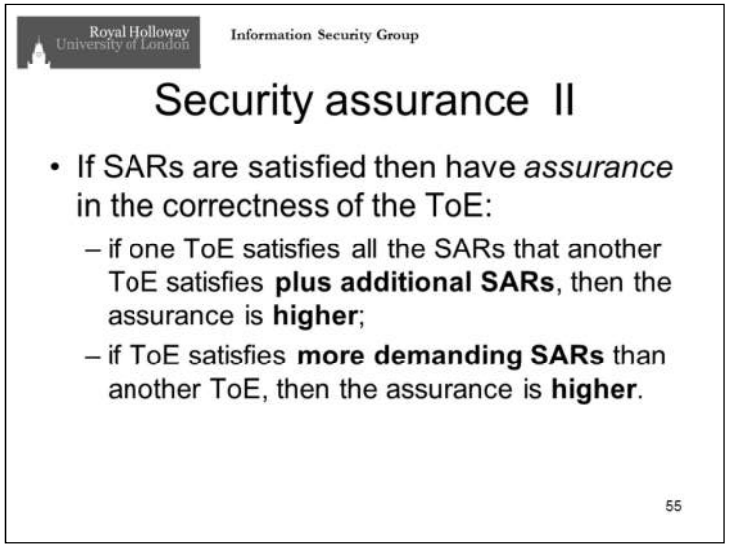
IY5512: Part 2



The **security assurance requirements** (**SARs**) are a formal description of the activities to be undertaken to establish the correctness of the SFRs.

They must be specified in a standardised language defined by the CC.

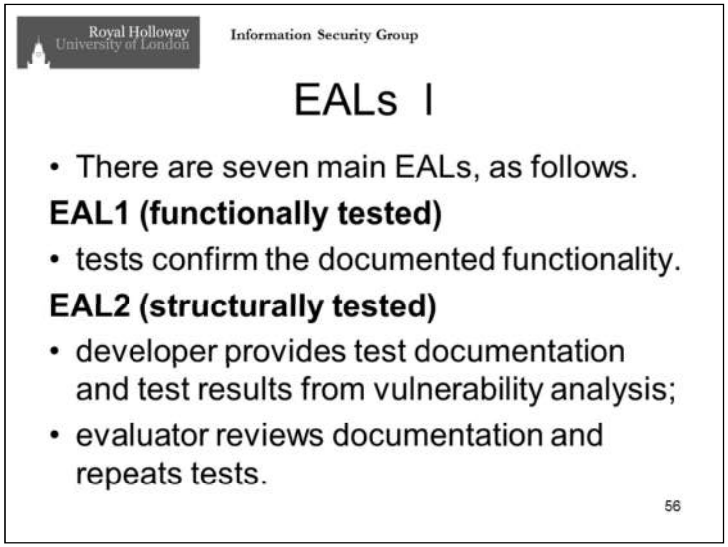
The rationale for the SARs is to ensure exactness of evaluation, and to facilitate comparisons of independent evaluations of competitor products.



The main idea is that, if the SARs are satisfied, the user of the evaluation has assurance (i.e. confidence) in the correctness of the ToE.

In general, if one ToE satisfies all the SARs satisfied by another ToE, together with some additional SARs, then the level of assurance is higher.

If a ToE satisfies more demanding SARs than another ToE, then the level of assurance is higher.



There are seven main Evaluation Assurance Levels (EALs). [Actually there are eight including EAL0, which corresponds to a failed evaluation.]

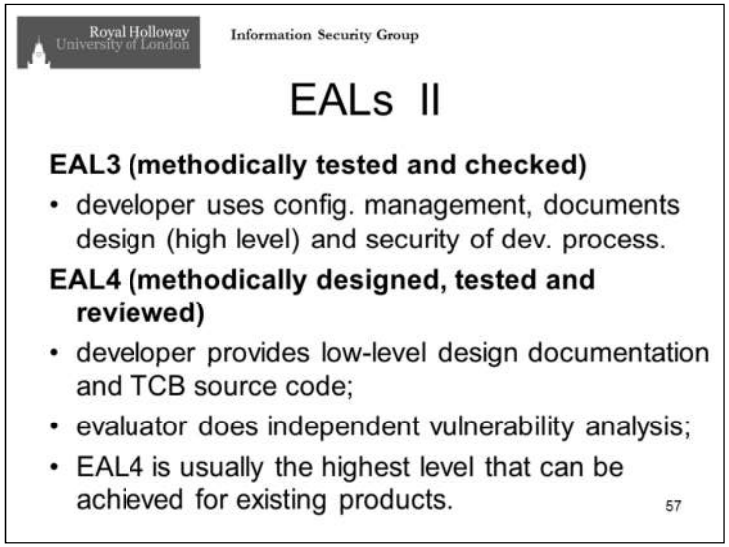
EAL1 (functionally tested)

• tests confirm the documented functionality. [This is the only level that does not have a matching level under Orange Book or ITSEC].

EAL2 (structurally tested)

• the developer provides test documentation and the test results from a vulnerability analysis;

• the evaluator reviews documentation and repeats tests.



EAL3 (methodically tested and checked)

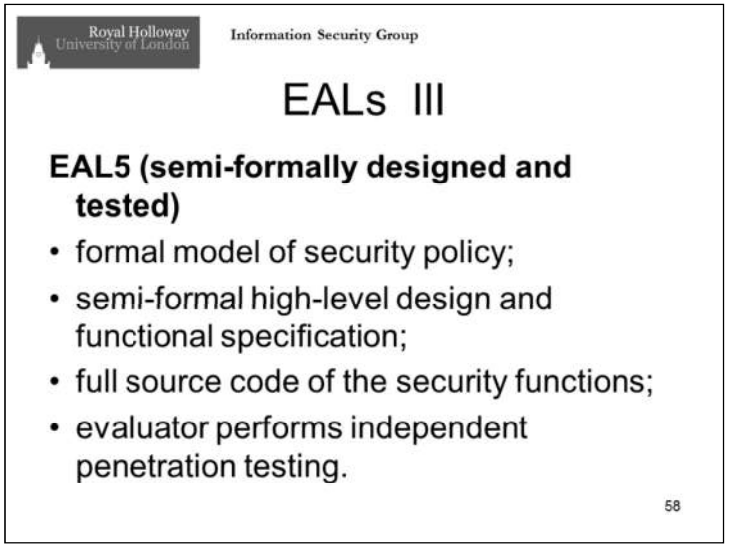
• the developer uses configuration management, and provides documentation of the design (at a high level) and of the security of development process.

EAL4 (methodically designed, tested and reviewed)

• the developer provides low-level design documentation and source code of the TCB;

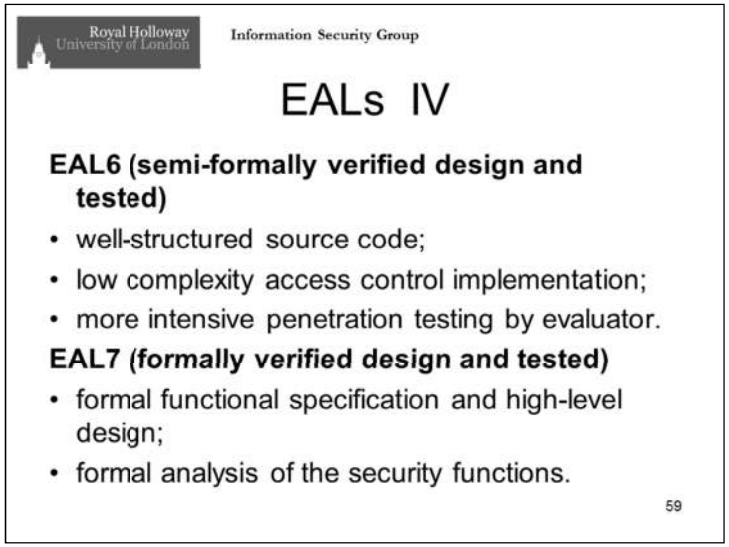
• the evaluator performs an independent vulnerability analysis;

• EAL4 is typically the highest level that can be achieved for existing products, i.e. products not developed with evaluation in mind from the outset of the development process.



EAL5 (semi-formally designed and tested)

- there must exist a formal (mathematical) model of the security policy;
- a semi-formal high-level design and functional specification must be provided;
- the full source code of the security functions must be submitted for evaluation;
- the evaluator performs independent penetration testing.

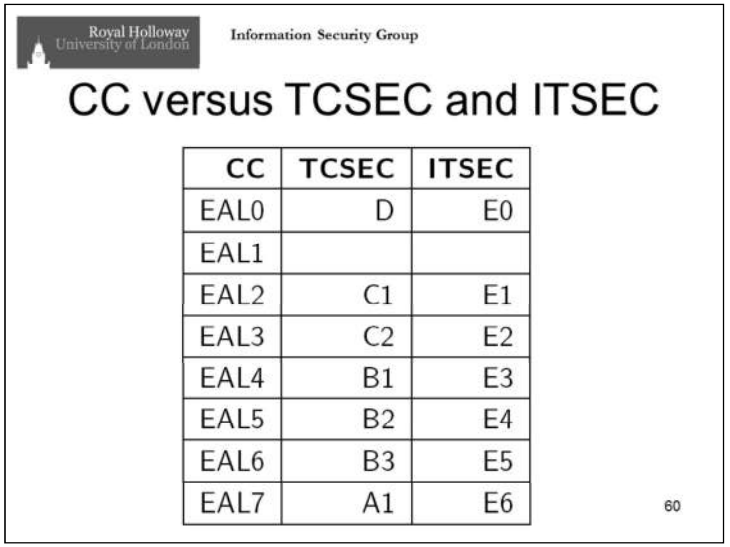


EAL6 (semi-formally verified design and tested)

- the product must possess well-structured source code;
- the access control implementation must have low complexity;
- it requires more intensive penetration testing by the evaluator.

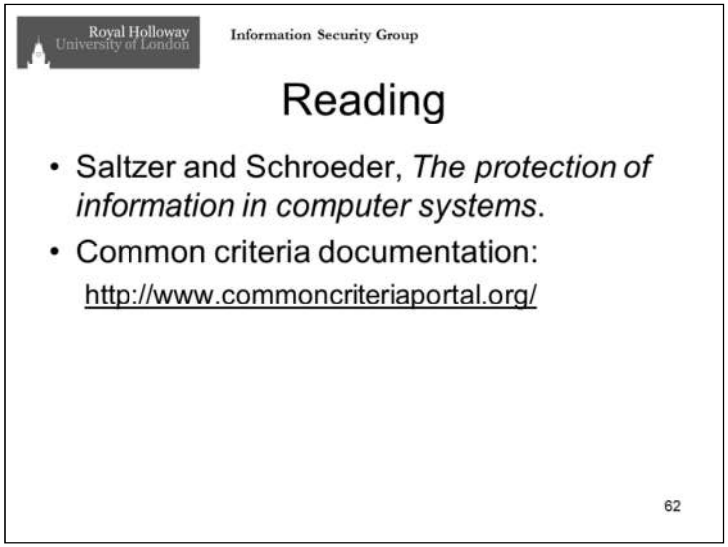
EAL7 (formally verified design and tested)

- requires a formal functional specification and high-level design;
- a formal analysis of the security functions must be provided.



The table provides a mapping between the evaluation levels of the Common Criteria and those of the TCSEC (orange book) and the European ITSEC. Note that EAL1 is a 'new' level introduced in the CC.

IY5512: Part 2



Saltzer and Schroeder, *The protection of information in computer systems.*

Common criteria documentation:

http://www.commoncriteriaportal.org/

Note that Annex A of part 1 of the common criteria provides a short history of the development of the common criteria.