JISEC

Issued on March 21, 2012

Security Architecture Guide for Developers

Even if security functions are properly implemented in an IT product, the security of the product as a whole cannot be protected once the security functions are disabled or bypassed by attackers. While developers view their products with all things considered, they have to implement the security functions as well as to design the products by taking measures to protect the security functions themselves. That is referred to as "security architecture."

Information Security Certification Office
Security Center, Technology Headquarters, IPA
www.ipa.go.jp/security/jisec/

- Table of Contents -

- Table of Contents -

End of this guide: Addendum A "Essential points for security architecture"

1 Introduction

The design concept for ensuring that security functions directly countering threats themselves can work properly without being hindered is referred to as security architecture.

A worldwide standard for IT security evaluation, Common Criteria (hereinafter referred to as "CC"), verifies the validity of the security architecture to be evaluated. This guide explains the concept of the security architecture and how to prepare documents in which the design contents are described (hereinafter referred to as "security architecture description").

The assumed readers of this guide include developers who design and implement security functions of IT products and developers who need to prepare security architecture descriptions for undergoing CC-based evaluations of IT products.

1.1 Objectives of this guide

Security architecture is an important concept for CC-based security evaluations. However, CC-related standards documents have generic and abstract expressions for describing security architecture, which apparently makes it difficult for developers who have only little experience especially with basic software to understand the contents.

Therefore, this guide is intended to support the understanding of the concept of security architecture, the design and implementation in accordance with the concept, and the preparation of security architecture descriptions, through explaining the concept of security architecture and the contents that should be included in security architecture descriptions required for the CC evaluations with concrete examples.

Note that this guide uses plain expressions in the explanations in order to facilitate the readers' understanding, part of which may not accurately correspond to the CC standards. If you have a plan to obtain a certification for the CC evaluation, it is advised to read the CC-related standards documents listed in "1.3 Common Criteria standards documents" in this guide as well.

1.2 Organization of this guide

This section explains the organization of this guide and the contents of each chapter.

- Chapter 1 Introduction
  This chapter includes the objectives of this guide, the target readers, and the scope of application.

- Chapter 2 Basic knowledge of security architecture
  This chapter explains the concept and necessity of security architecture, as well as the concepts of "security domain," "TSF self-protection," "TSF non-bypassability," and "TSF secure initialization," which are important in understanding security architecture.

- Chapter 3 Security architecture description
  This chapter provides the concrete explanations on how to describe security architecture descriptions required for the CC evaluations.

- Chapter 4 Conclusion
  This chapter explains the points that should be taken care of in designing security architecture on the basis of the contents explained in this guide.

1.3 Common Criteria standards documents

The evaluation criteria and evaluation methodology of this guide are based on the standards documents listed in Table 1-1 and Table 1-2 below. The evaluation criteria and evaluation methodology are referred to as "CC" and "CEM," respectively, in their abbreviations.

Table 1-1: CC/CEM standards documents (Japanese translation versions)

CC/CEM version 3.1 Release 3 (CC/CEM v3.1 Release 3)
Evaluation criteria: Common Criteria for Information Technology Security Evaluation (CC version 3.1 Release 3)
Part 1: Introduction and general model Version 3.1 Revision 3 [Japanese version 1.0]
Part 2: Security functional components Version 3.1 Revision 3 [Japanese version 1.0]
Part 3: Security assurance components Version 3.1 Revision 3 [Japanese version 1.0]
Evaluation methodology: Common Methodology for Information Technology Security Evaluation (CEM version 3.1 Release 3)
Evaluation methodology Version 3.1 Revision 3 [Japanese version 1.0]

Table 1-2: CC/CEM standards documents (Original versions)

CC/CEM v3.1 Release 3		
Evaluation criteria: Common Criteria for Information Technology Security Evaluation (CC v3.1 Release 3)		
Part 1: Introduction and general model	Version 3.1	Revision 3
Part 2: Security functional components	Version 3.1	Revision 3
Part 3: Security assurance components	Version 3.1	Revision 3
Evaluation methodology: Common Methodology for Information Technology Security Evaluation (CEM v3.1 Release 3)		
Evaluation methodology	Version 3.1	Revision 3

Reference: Evaluation criteria Common Criteria, IPA
http://www.ipa.go.jp/security/jisec/cc/index.html

This guide is based on the contents stated in the sections below, which are taken from the CC and CEM standards documents, "CC Version 3.1 Release 3 Part 3" and "CEM Version 3.1 Release 3."

- CC Part 3, "12 Class ADV: Development"
  page 76, paragraphs 208-209
- CC Part 3, "12.1 Security architecture (ADV_ARC)"
  pages 78-79
- CC Part 3, "Annex A Development (ADV)"
  pages 173-177, "A.1 ADV_ARC: Supplementary material on security architectures"
- CEM, "11.3 Security architecture (ADV_ARC)"
  pages 91-96

1.4 Terms and definitions

Table 1-3 shows the terms related to the CC evaluation criteria and evaluation methodology used in this guide.

Table 1-3: CC/CEM terms and definitions

Terms	Explanation
CC (Common Criteria)	The international standard ISO/IEC 15408 for evaluating whether a product or system has been properly designed and the design has been accurately implemented from the viewpoint of information security. This is referred to as **CC**, which is the abbreviation of "**C**ommon **C**riteria."
CEM (Common Evaluation Methodology)	A common evaluation methodology that defines the approach of the CC-based evaluation (e.g., what target to evaluate, what judgment is required). This is referred to as **CEM**, which is the abbreviation of "**C**ommon **E**valuation **M**ethodology."
TOE (Target of Evaluation)	A software product or hardware product, etc., that is the target of the CC-based evaluation. This is referred to as **TOE**, which is the abbreviation of "**T**arget **o**f **E**valuation."
TSF (TOE Security Functionality)	Functions required for properly implementing the security requirements that should be satisfied by the TOE, including security functions and supporting functions for ensuring proper behavior of security functions. It consists of hardware and software in the TOE. Note that, to make it a plain expression, it may be simply expressed as "security function" in this guide. This is referred to as **TSF**, which is the abbreviation of "**T**OE **S**ecurity **F**unctionality."
Security Target	A document describing the security requirements and security-related specifications that serves as a basis for the evaluation of the TOE. This is referred to as **ST**, which is the abbreviation of "**S**ecurity **T**arget."

2 Basic knowledge of security architecture

This chapter explains the knowledge, terms, and ideas necessary for understanding the requirements for preparing the security architecture description, including the details of security architecture and its realization procedure.

2.1 Security architecture

To put it briefly, security architecture is a mechanism for protecting the security functions in a product and ensures their proper behavior even when the security functions themselves become the target of an attack.

The following explains the relation between "security functions" and "security architecture."

■ Security functions

Security functions are comprised of functions for protecting assets to be protected (hereinafter "protected assets"), including the users' important data, from unauthorized access and functions for realizing organizational security policies (security-related requirements needed as the policies of an organization, such as "audit logs shall be obtained," which may not necessarily be associated with the protection of unauthorized access directly).

For example, the identification and authentication function and access control function for preventing unauthorized access to protected assets, encryption function for protecting confidential information in protected assets, and audit logging function for monitoring the operating status of the security functions fall under the category of "security functions."

■ Security architecture

Security architecture is a mechanism for protecting security functions from attacks to ensure their proper behavior.

Even when security functions are implemented in a product, it cannot possibly be claimed that protected assets are protected and the organizational security policies have been achieved if the security functions themselves become the target of attacks and fail to behave properly. Therefore, when security functions such as identification/authentication and access control are implemented, "security architecture," which is a mechanism that can protect the security functions

themselves from attacks, is absolutely essential.

For instance, a mechanism that checks unauthorized inputs from users for eliminating them and a mechanism where separate memory spaces are allocated for each process in response to user input processing for restricting such processes from affecting memory spaces used by another process fall under the category of "security architecture."

As described above, security architecture has to be taken into account for all security functions as a countermeasure against threats and for realizing the organizational security policies. The following provides an explanation using an attacker's attempt to make unauthorized access to protected assets as an example.

2.2 Attack against security functions

In general, products have interfaces to use protected assets, and the users access the protected assets via such interfaces. Normally, as described in Figure 2-1, security functions of some sort, such as identification/authentication and access control, prevent unauthorized access and protect protected assets.

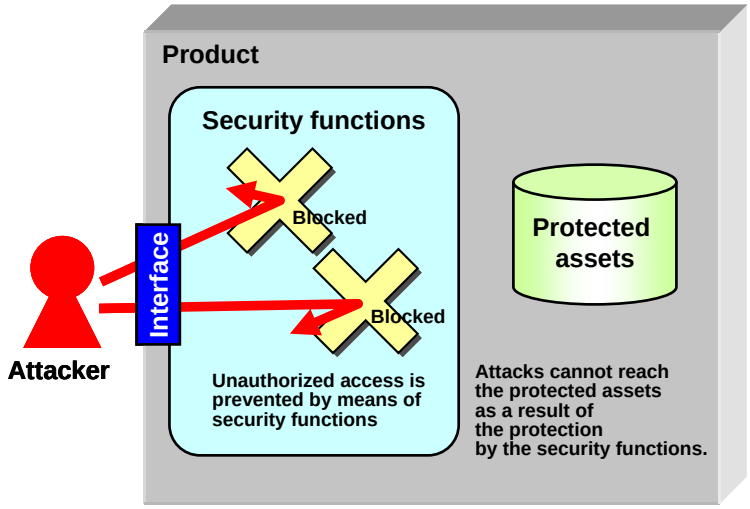


Figure 2-1: Image of a security function protecting protected assets

In case the security functions themselves in the figure above are attacked to be bypassed (avoided) or tampered with, the security functions will not be able to behave properly and might let the attacker conduct unauthorized access.

The next section explains what the "bypassing (evasion)" and "tampering" attacks against security functions are.

■ **Bypassing**

"Bypassing" of security functions denotes a situation where a security function that ought to be applied when the product is used under normal circumstances is avoided without being applied.

Figure 2-2 shows an image of bypassing.

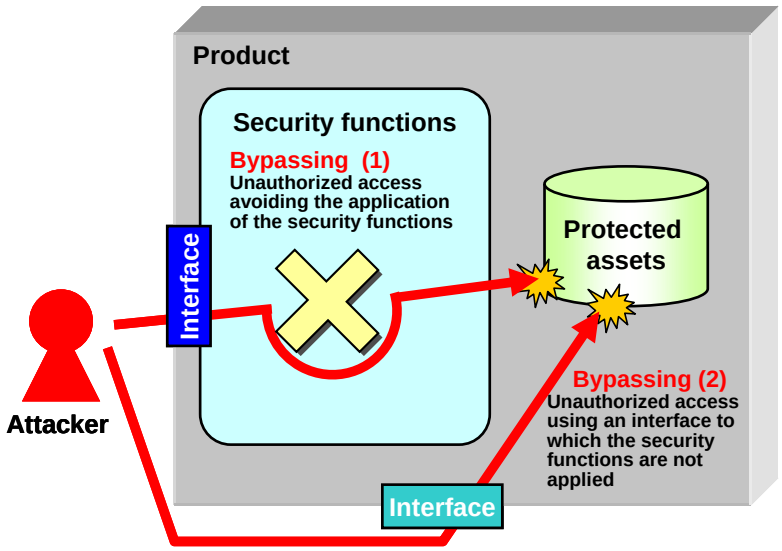


Figure 2-2: Image of a security function being bypassed (avoided)

For instance, the following cases are included in "bypassing."

- Cases where an insufficient design or implementation of an interface of a product prevents the application of the security functions when the product is used
- Cases where a product contains processing in which the influence prevents the application of the security functions when the product is used in a way exceeding assumptions, for example, by changing the assumed order of use or by modifying the value of a parameter out of the scope of the assumption
- Cases where a leakage or presumption of confidential data on which security functions depend, such as a cryptographic key, allows for the decryption without using the security functions of the product

■ **Tampering**

"Tampering" of security functions denotes not only physical modification of the security functions but also attacks as a whole, including unauthorized inputs having adverse effects on the behavior of the security functions.

Figure 2-3 shows an image of "tampering."

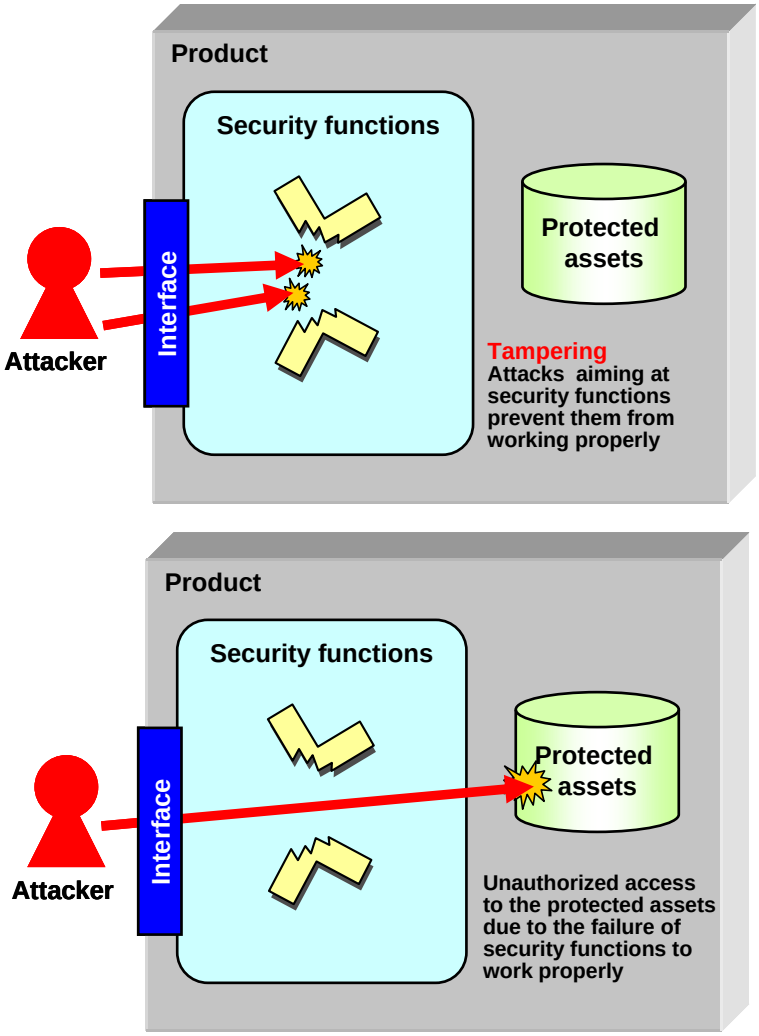


Figure 2-3: Image of a tampered security function

For instance, the following cases are included in "tampering."

- Modification of program codes that realize security functions or data on which the behavior of security functions depend
- Unauthorized inputs that cause unexpected behavior of the product, such as buffer overflow and SQL injection
- Attacks that prevent the behavior of security functions, for example, by suspending a process that records audit logs

Note that tampering of security functions can result in the suspension of the security functions, and the execution of a command infiltrated from outside can result in bypassing of security functions. Therefore, such bypassing resulting from tampering is treated as "tampering" in the explanations of this guide.

With the understanding to the types of those attacks of "bypassing" and "tampering" as explained above, what is important for security architecture is to exhaustively consider the possibility of intrusion to security functions.

2.3 Conditions for protecting security functions

CC evaluations require developers to design and implement their products in a manner in which security functions cannot be bypassed or tampered with, and at the same time, to describe the concrete mechanisms designed and implemented in the products as the security architecture description.

In security architecture descriptions, it is important to consider the protection of the security functions from the four perspectives as follows: "security domain," "TSF self-protection," "TSF non-bypassability," and "TSF secure initialization."

* TSF (TSF: TOE Security Functionality, TOE: Target of Evaluation) means all the portions in a product that are required for proper behaviors of the security functions, including both of the mechanisms for realizing security functions and for realizing the security architecture.

(1) Security domain

Security domain is an idea that serves as a basis for the design and implementation intended to prevent the security functions from being bypassed or tampered with. Realizing the following "TSF self-protection" and "TSF non-bypassability" on the basis of security domain will lead to an efficient and robust implementation to prevent bypassing and tampering of security functions.

(2) TSF self-protection

TSF self-protection refers to a mechanism in which security functions protect themselves for preventing the security functions from being tampered with.

(3) TSF non-bypassability

TSF non-bypassability refers to a mechanism of security functions that ensures the application of necessary security functions at a proper timing for the prevention of bypassing.

(4) TSF secure initialization

TSF secure initialization refers to a mechanism that prevents intrusions to the security during the initialization process of a product from startup to the beginning

of the operation mode for ensuring that the security functions are initialized in perfect conditions and enter the operation mode.

The following sections explain those four perspectives.

2.3.1 Security domain

(1) Concept of security domain

Security domain is, in short, the concept of confining the behavior of programs having a risk of adversely affecting security functions or protected assets within a certain limited scope.

The behavior of programs having a risk of adverse effects refers to the "behavior" of programs existing in an environment from which security functions or protected assets can be accessed.

Examples of applicable behaviors are attacks by unauthorized programs such as computer viruses, as well as various behaviors even of authorized programs, including abnormal behaviors caused by the attacker's unauthorized manipulations, malfunctions caused by the user's operational errors, and faulty behaviors caused by implementation errors. All such behaviors having adverse effects can be regarded as "attacks."

Figure 2-4 shows an image of the attacks by the behavior of such programs.

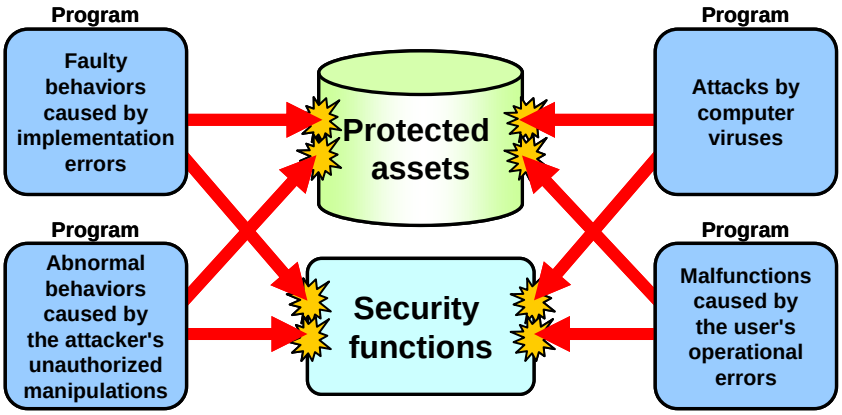
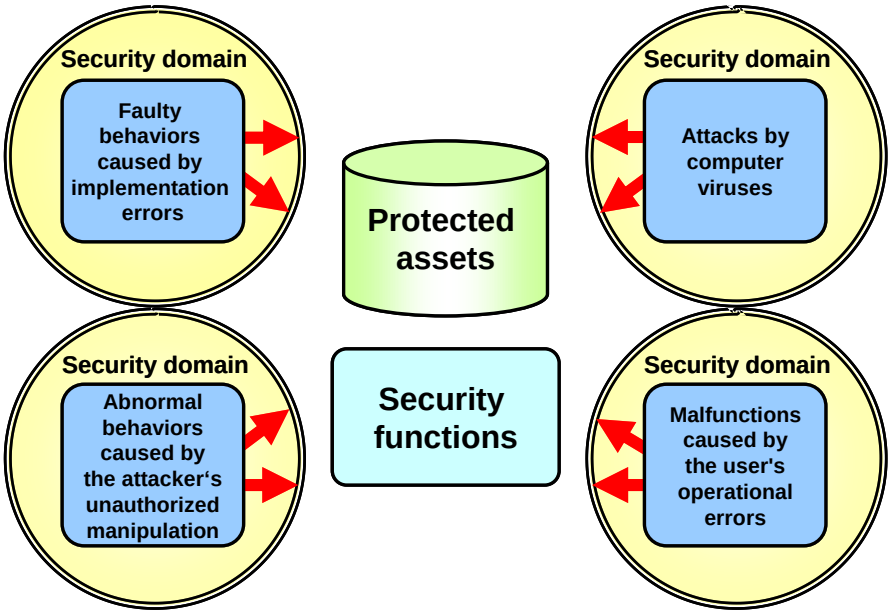


Figure 2-4: Image of behaviors adversely affecting security functions or protected assets

"Security domain" refers to a scope or environment in which such a program is isolated for preventing attacks by its behavior.

Surrounding a program with security domain will restrict the scope that the program can access inside the security domain, thus resulting in restraining the attacks by the behavior of the program in a collective manner.

Figure 2-5 shows an image of the security domain.



Against security functions and protected assets, the behaviors of the programs in an environment from which the security functions or protected assets can be accessed, can be protected from adverse effects by surrounding with and confining to security domains.

Figure 2-5: Image of security domain

As a concrete example, general operating systems have a mechanism, in which the execution of applications running under user operations is controlled through the process management and memory management of the operating system, restricting the scope that the application can freely read and write to the memory in the same process. The scope of the processes that the application can execute and the memory that it can use, which is controlled through the mechanisms of process management and memory management, can be considered to be security domain.

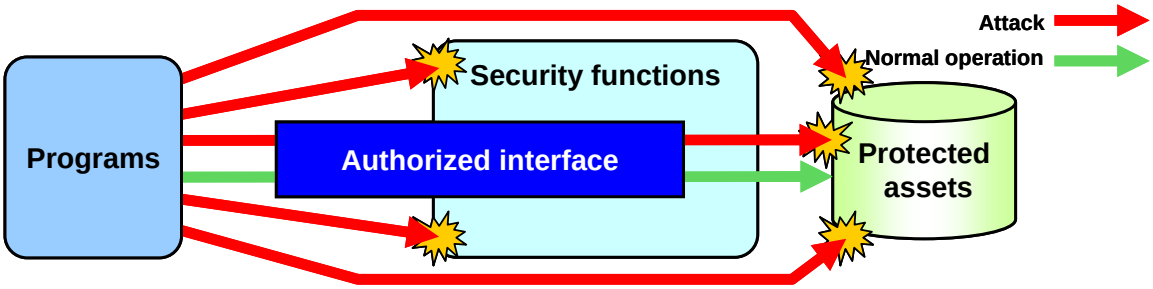
(2) Mechanism of domain separation

A mechanism called "domain separation" is required for restricting security domain, i.e., the scope of the memory area where a program having a risk of adverse effects can freely read and write.

For instance, hardware systems and address administration methods that distinctly separate the address space used by the users and that used by the system are often adopted as the mechanism of domain separation. Some application software products utilize a mechanism offered by the operating system, which is out of the scope of the product, for realizing domain separation.

(3) Effectiveness of domain separation

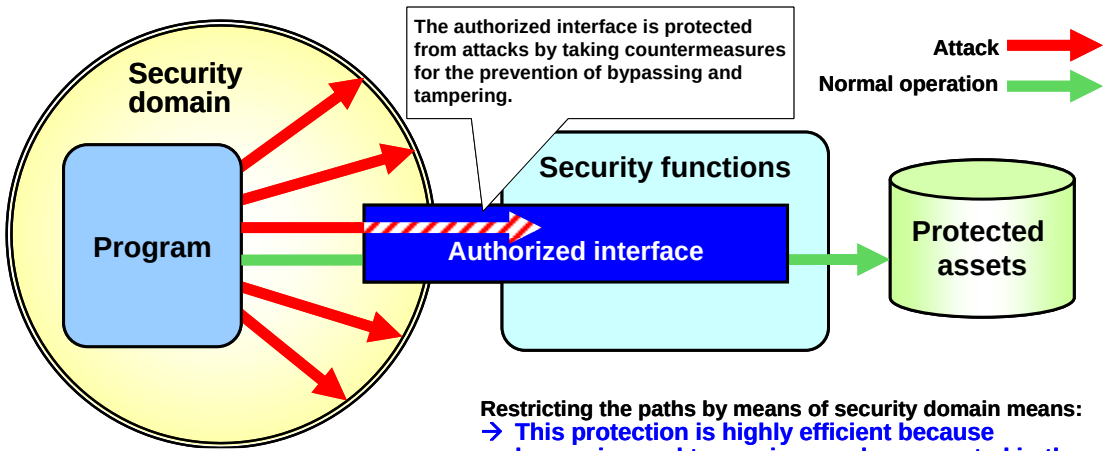
Figures 2-6 and 2-7 show the cases where domain separation is not implemented and where domain separation is implemented, respectively, as the preventive measures against the bypassing and tampering of security functions.



If the scope that the program can access is not restricted, there are countless attacking paths in addition to the path to the authorized interface, such as access to memory out of the area.

The existence of countless paths means:
→ Countermeasures for the prevention of bypassing and tampering are required to all the paths individually, thereby resulting in an increased quantity of work, posing additional risk of overlooked attacking paths or insufficient consideration to measures.

Figure 2-6: When domain separation is not implemented



The scope that the program can access is restricted to the inside of the security domain with the exception of the path to the authorized interface.

Restricting the paths by means of security domain means:
→ This protection is highly efficient because bypassing and tampering can be prevented in the majority of the paths, eliminating the need for individual measures.

Figure 2-7: When domain separation is implemented

When domain separation has not been implemented, there are countless paths of attacks by the behavior of the program. This calls for taking countermeasures against individual attacks, posing additional risk of overlooked attacking paths or insufficient consideration to measures.

Use of domain separation, however, makes it possible to counter attacks by the behavior of programs that were formerly able to access protected assets or security functions in a collective manner, eliminating the need to take measures individually. Note that it is still needed to take preventive measures against the bypassing and tampering of the security functions on authorized interfaces. In this way, domain separation is a method that can realize the prevention of the bypassing and tampering of security functions robustly and efficiently. For the protection of security functions, it would be advisable to adopt the concept of security domain and implement the mechanism of domain separation.

2.3.2 TSF self-protection

(1) The concept of TSF self-protection

TSF self-protection refers to a mechanism in which security functions (TSF) of a product protect themselves from tampering attacks against security functions.

(2) How to realize TSF self-protection

The methods for realizing TSF self-protection can be divided into two; using domain separation described above and using a mechanism other than domain separation. It is common to use both in combination for countermeasures.

(a) Using domain separation

As described above, programs having a risk of tampering attacks can be efficiently countered by surrounding them with security domain. However, attacks to interfaces connected to security functions cannot be covered by this method.

(b) Using a mechanism other than domain separation

Countering tampering attacks using interfaces requires a mechanism other than domain separation. Examples of such attacks are buffer overflow, SQL injection, and other unauthorized inputs to interfaces. Individual countermeasures suitable for the nature of those attacks are required of the interfaces.

Recognizing all paths having the risk of attacks to the security functions and their ways of tampering, developers are required to design and implement the countermeasures for preventing tampering without omission by using domain separation or other mechanisms.

(3) TSF self-protection using a mechanism other than the product

A product can realize TSF self-protection by using a mechanism offered by a function outside the product.

When the product is an application program, for instance, it can realize domain separation by using the process management and memory management that are provided by the operating system, which is the execution environment of the program. Although those functions provided by the operating system are not functions of the product itself, the parts where the product protects itself by using external functions can fall under the category of "TSF self-protection."

Going into further detail about the example above, suppose that the product is an application program, the processing of the application program that uses system calls provided by the operating system for generating a process or executing a program is processing inside the application program necessary for using an operating system-based domain separation mechanism, thus falling under the category of "TSF self-protection."

2.3.3 TSF non-bypassability

(1) Overview of TSF non-bypassability

TSF non-bypassability refers to a mechanism of security functions (TSF) that ensures the application of security functions at a proper timing when the program is used for preventing the security functions from being bypassed.

Bypassing paths can be categorized into two types as follows: when the product has an interface to which security functions are not applicable, and when the security functions of the product internally have a usage that can hinder the application of the security functions. Developers are required to take countermeasures against both cases.

(2) How to realize TSF non-bypassability

When realizing TSF non-bypassability, developers have to pay attention to the following points.–

- Prevention of omissions in the application of security functions

  Developers have to clarify all interfaces that can access the protected assets in the product and make sure that there are no omissions in the application of the security functions. This is because all interfaces that can access the protected assets are supposed to be designed so that the security functions are applied to them without fail.

  Note that the more access paths to the protected assets there are, the more difficult it becomes to ensure that there is no omission in the application of the security functions in every path. In such cases, using domain separation for limiting the paths that can access the protected assets will facilitate the prevention of omissions in the application of security functions.

- Prevention of bypassing inside security functions

  Developers should design the interfaces of a product with care so that the security functions internally have no processing, in which the influence from the unexpected order of use or the entry of an unexpected parameter may hinder the application of the security functions.

  For instance, Web applications may require the implementation of a mechanism that can control the session management and the order of screen transition so that the security functions such as identification/authentication and access control are applied without fail.

2.3.4 TSF secure initialization

TSF secure initialization is a mechanism that ensures the security of the product in the middle of startup.

Since the security functions of a product in the middle of startup have not yet been working properly, the security functions may fail to counter attacks, or a special function having an influence to the security can be used for a short time. This state in the middle of startup, where security measures tend to be overlooked in general, can be a good opportunity for attackers to attack.

For instance, a product in the middle of startup may enter dangerous states as follows:

- If the network communication function is activated earlier than the activation of the access control function (filtering) in the middle of startup of a firewall, the

filtering will not function, posing a risk of communications aimed at attacks passing through the firewall.

- Taking a server operable only by the administrators as an example, the server may enter a state in which general users other than the administrators can use special maintenance functions (e.g., the single user mode) if a special operation is carried out or a failure occurs in the middle of startup of the operating system, posing the risk of, for example, the change of the administrator password and the copy of confidential data to external storage media.

Because of the risks described above, developers shall design and implement a mechanism in which the product can counter attacks to ensure that the initialization process of the security functions is properly conducted even in the period from the startup of the product to the beginning of the operation mode.

3 Security architecture description

This chapter explains the contents to be described in the security architecture description, a document that the certification for the CC evaluation requires developers to submit, with concrete methods of description and examples.

3.1 Content of security architecture descriptions

The security architecture description should contain descriptions about how the product is designed and implemented for preventing the security functions of the product from being bypassed or tampered with, which are sorted out into the four perspectives explained in Chapter 2.

- Security domain
- TSF self-protection
- TSF non-bypassability
- TSF secure initialization

Among the four perspectives above, "TSF self-protection" and "TSF non-bypassability" are mechanisms that protect security functions of a product in the operation mode from attacks of bypassing and tampering, whereas "security domain" is an idea that serves as a basis for the implementation of those two mechanisms. "TSF secure initialization" refers to a mechanism that protects a product during the initialization process from the startup of the product to the beginning of the operation mode. The next section and subsequent sections explain how to describe each of those perspectives.

Note that the contents described in the security architecture description are the mechanisms that have actually been implemented in products in terms of the perspectives above. Therefore, the security architecture description and design documents actually prepared in the product development must maintain consistency without fail.

3.2 Security domain

Security domain is defined as a "collection of resources to which an active entity has access privileges" in the CC standards. An active entity typically refers to a program running in the product in accordance with the operations (hereinafter referred to as "program acting on the user's behalf") when a user or a terminal used by a user (hereinafter collectively referred to as "user") operates a product.

In the security architecture description, the item of security domain has to contain the descriptions about the definition of the security domain implemented in the product and the mechanism of the domain separation. In particular, it is important to describe the following information clearly.

- Definition of security domain

  The resources that programs acting on the user's behalf can access without restrictions must be defined. Typically, address spaces assigned on a process-by-process basis fall under this category. In addition, address spaces of a virtual machine, such as JavaVM, may also fall under this category when programs running on the virtual machine are executed in a restricted environment for preventing unauthorized system manipulations (generally, referred to as "sandbox"). When defining security domain, developers should be careful that security functions to be protected or protected assets are not included in the scope.

- Domain separation mechanism

  On the basis of the definition of the security domain above, the restriction mechanism must be clarified to prevent the programs acting on the user's behalf from accessing outside the scope of the security domain.

There are various functions to be provided, usage, and implementation methods of programs depending on products, so that the interpretation of the security domain and the mechanism of the domain separation differ accordingly. It can be said that product developers shall design and implement proper security domain in view of the properties of those products.

In the next section, examples of how to specify the security domain of a product and to describe the items in the security architecture description are introduced for your improved understanding. Note that the following methods are just examples, and there are different methods as well.

3.2.1 Specification of security domain

One of the primary objectives of security domain is to prevent the tampering of security functions. For that objective, it is required to prevent programs acting on the user's behalf from writing without any restrictions to memory areas to which codes and data constituting the security functions of the product are located.

This section introduces how to specify the mechanisms of the security domain and domain separation, focusing attention on what mechanism can prevent programs acting on the user's behalf from accessing a memory area to which security functions are located. The following is the explanation of a concrete procedure with examples.

(1) Specification of the programs acting on the user's behalf and the security functions

First, the security functions of the product and the parts of the programs acting on the user's behalf should be specified. Products, in general, have user functions, which are the principal purpose of the product, and security functions for the purpose of restricting user access and preventing unauthorized access.

With an Internet banking system, for instance, the functions for inquiring the balance of user accounts and making transfers are user functions, while the functions to identify and authenticate users on the log-in screen and impose restrictions for preventing them from inquiring accounts of others are security functions. In the product, the parts of programs that realize the former user functions (parts of the programs acting on the user's behalf) and the parts of programs that realize the latter security functions should be separately specified.

Next, the implementation styles used for locating of the security functions of the product and the programs acting on the user's behalf should be specified. It can be considered that, in general, the implementation styles of the security functions and the programs acting on the user's behalf are categorized into one of the following four types.

(a) Implemented as a kernel (e.g., device driver) of the operating system
(b) Implemented as processes on a program-by-program basis
(c) Implemented as threads in a process on a program-by-program basis
(d) Implemented as a shared library that can be commonly used by multiple programs

There are various implementation styles depending on the product, where, for instance, (a) is used for the security functions and (c) for the programs acting on the

user's behalf, or (b) is used for both the security functions and the programs acting on the user's behalf.

(2) Judgment of domain separation

Then, after determining which implementation the programs acting on the user's behalf and the security functions falls under, respectively, among the implementation styles (a) to (d) specified in Section (1), it should be examined whether or not the memory areas to which the security functions are located are protected from the programs acting on the user's behalf.

In general, memory areas to which security functions are located are protected by one of the following mechanisms or a combination of them. Under those mechanisms, programs acting on the user's behalf cannot affect the security functions no matter what behavior they conduct. In other words, the security functions are domain-separated.

- Mechanism of domain separation based on the execution modes of the processor

  This mechanism is relevant to cases where the security functions are implemented in the kernel of the operating system and the programs acting on the user's behalf are implemented as general processes running on the operating system. Figure 3-1 shows an implementation example of domain separation.

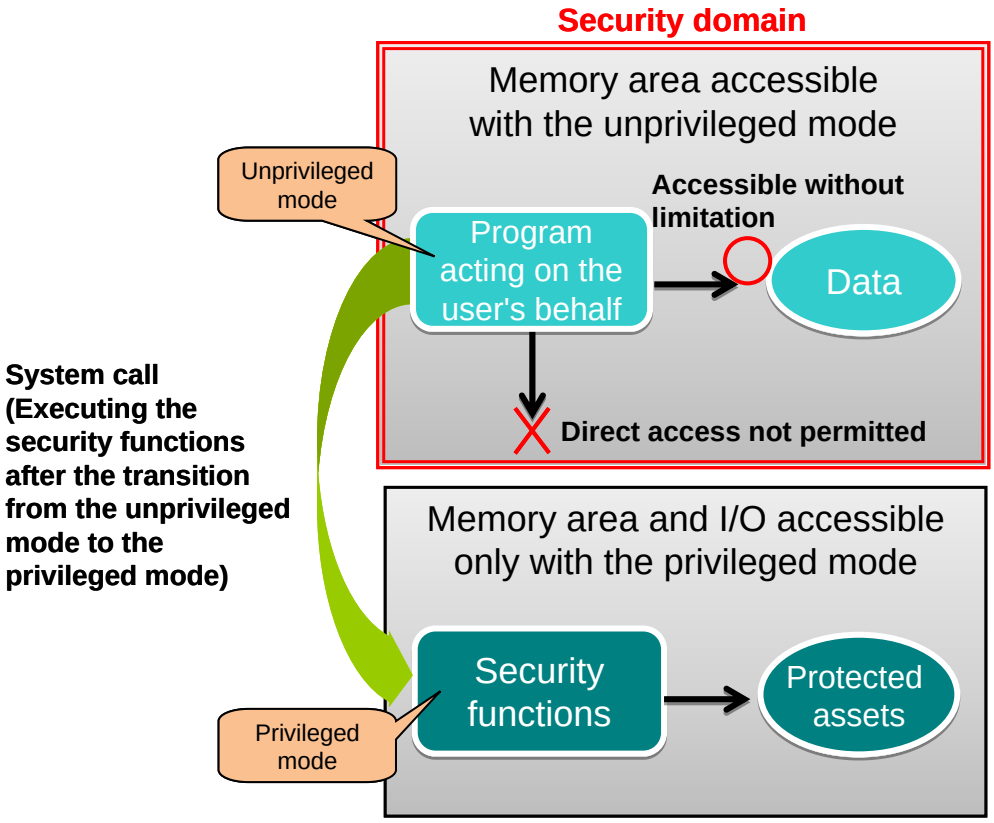


Figure 3-1: Example of domain separation (based on the execution modes of the processor)

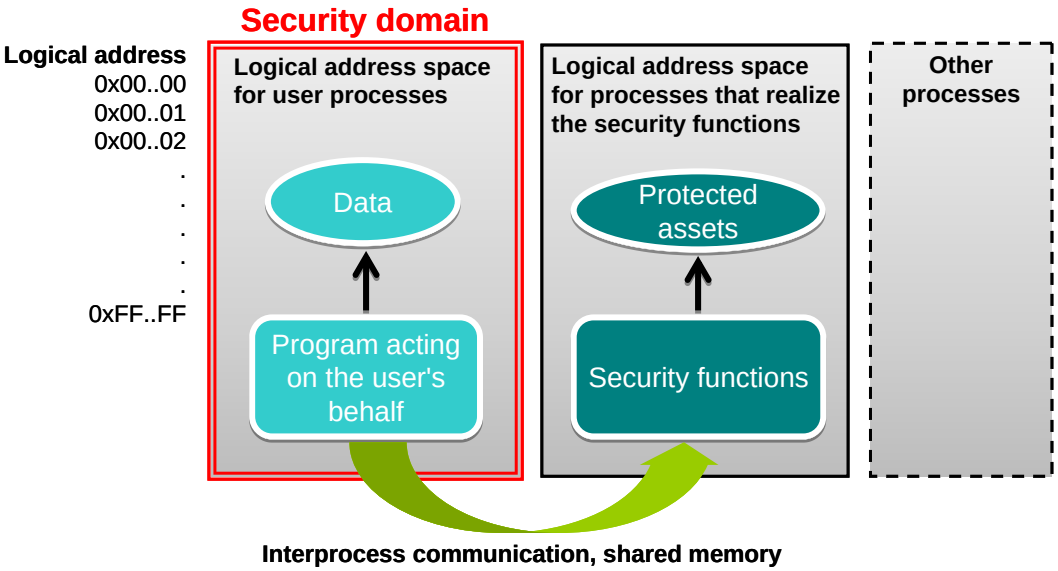
In this case, when a general process executes a program acting on the user's behalf, the operating system controls the program to be executed in the unprivileged mode of the processor by utilizing the execution modes of the processor. On the other hand, the kernel in which the security functions are located is so controlled that execution, read, and write operations are permitted only in the privileged mode of the processor.

As a result, the area where general processes can freely execute, read, and write is limited to the scope permitted in the unprivileged mode, preventing them from executing, reading, and writing in the kernel area where the privileged mode is required.

- Mechanism of domain separation based on the logical address spaces of processes

  This mechanism is relevant to cases where the security functions are implemented as processes, and the programs acting on the user's behalf are implemented as processes other than those for security functions. Figure 3-2

shows an implementation example of domain separation.



Different processes access different data even if the logical addresses are the same. Programs in each process can access only the address space assigned to its own process.

Figure 3-2: Example of domain separation (based on the logical address spaces)

In this case, the operation system executes processes by allocating different logical address spaces to each process. As a result, the area that each process can access will be limited to the logical address space assigned to its own process. Therefore, the programs acting on the user's behalf cannot access the codes and data of the processes that execute the security functions in principle because the logical address spaces are different.

- Mechanism of domain separation based on a restricted software execution environment

  Although the examples above depend on the hardware for the major part, there can be domain separation mechanism that is realized primarily by software. Such mechanism is relevant to cases, such as JavaVM which interprets and executes Java program codes at the same time, where a software execution mechanism provides the programs acting on the user's behalf with an execution environment in which the resources (e.g., programs and data) they can freely access are restricted. On the other hand, the security functions are implemented as separate programs from the programs acting on the user's behalf with another execution environment assigned. In this way, the domains can be separated.

With the following implementation style, the memory area to which the security functions are located is not protected, posing a risk of being interfered with by the programs acting on the user's behalf. In this case, the programs acting on the user's behalf are not domain separated.

- Implementation in the same logical address space in the same processor execution mode

  This category is relevant to cases where the security functions and the programs acting on the user's behalf are implemented in the same process. It includes not only cases in which both of them are not clearly separated in implementation, but also cases which have implementation styles where threads or shared libraries run in a process sharing a logical address space.

Such implementation styles as described above require another mechanism in accordance with the individual implementation style for preventing the security functions of the product from being tampered with. For instance, it is required for threads running in a process sharing a logical address space to be implemented on the basis of an additional mechanism of some kind or implementation rules for preventing interference between threads.

3.2.2 Description of security domain

On the basis of the analysis in Section 3.2.1, the specified security domain should be described in the security architecture description. As explained above, the domains for the programs acting on the user's behalf should be separated so that they cannot interfere with the security functions. There are exceptional cases, however, where such domains are not separated. The following explains the contents of the description for both cases.

(1) When domain separation is implemented

In this case, the definition of the security domain and the mechanism of the domain separation are to be described. The mechanism of the domain separation includes both cases where the product realizes the entire domain separation by itself and where the product realizes the domain separation utilizing external functions.

As an example, the following explains the case where the programs acting on the user's behalf are realized as processes, and the security function are also realized as processes.

In this case, the security domain is formed on a process-by-process basis, so that the environments assigned to each of the processes (address spaces, in this case) should be described as the definition of the security domain. In addition, in order to explain the mechanism of the domain separation, it should be described that the security functions are also realized as processes. The mechanism of the domain separation can be explained as follows. Different logical address spaces are allocated to processes on a process-by-process basis. Therefore, the scope that each process can access is limited to the logical address space for its own process, thus preventing it from accessing a logical address space for other process.

(2) When domain separation is not implemented

First of all, whether domain separation is not required in a product should be examined.

In a processing of complicated data entered by a user (e.g., display processing of PDF format data), it would be more secure if the interpretation process of the data is executed in security domain that is restricted in order not to interfere with the security functions.

If domain separation is not implemented, there are substantial risks of the interference to the behavior of the security functions caused by implementation errors of the program, malfunctions, or attacks exploiting flaws. If the preventive measures against them have not been taken into account, the implementation architecture of the product has to be reconsidered.

If it is appropriate for the product that domain separation is not implemented, the rationale of its validity has to be described.

The following can be an example for such rationale. The only interface of the product is physical buttons for menu selection. User input is strictly restricted, eliminating the possibility of unpredictable behavior. Therefore, even though domain separation is not implemented, adverse effects to the security functions can be completely eliminated only by validating input values from the interface without omission.

3.2.3 Confirmation of the description contents

In the certification for the CC evaluation, the security functions are evaluated according to the security functional requirements described in the Security Target. Therefore, it is required that the description of the security domain is consistent with the security functional requirements. The following should be conducted for confirmation.

(1) Viewpoint of the security functions

It should be confirmed whether or not all parts of the security functions for realizing the security functional requirements have been taken into account in the domain separation mechanism. For instance, the security functions can be realized both in drivers in the kernel of the operating system and in processes running on the operating system. In this case, the programs acting on the user's behalf have to be domain separated from both viewpoints of drivers and processes in order not to interfere with the security functions.

(2) Viewpoint of the programs acting on the user's behalf

With the parts of the programs acting on the user's behalf, it should be confirmed whether or not the subjects of the security functional requirements stated in the Security Target have been taken into account. The target programs that examine domain separation may vary, depending on the difference in users, such as between general users and administrators, or the difference in the user's access style, such as on the console device of the product and via a network.

3.2.4 Important notes for domain separation

The objective of domain separation is to prevent the security functions from being bypassed or tampered with. For this reason, as previously introduced, domain separation is said to be realized in general by a mechanism other than the functions shown in the security functional requirements, such as the execution mode of the processor, memory management, and software execution environment.

Therefore, it should be noted that such assertion that unauthorized users can be domain-separated by means of, for instance, the identification and authentication function (e.g., ID and password) or the access control function, which are part of the security functions, is not expected as a description for security architecture.

3.3 TSF self-protection

In the item of TSF self-protection in the security architecture description, it is required to describe the mechanisms of the security functions (TSF) for preventing the security functions of the product from being tampered with. TSF self-protection mechanism can be broadly divided into the following two categories:

- Self-protection mechanism by means of domain separation
- Self-protection mechanism by means of ways other than domain separation

(Countermeasures against tampering that cannot be countered only by domain separation, such as user input)

Note that tampering prevention of TSF may be realized by utilizing not only the security functions of the product but also functions outside the product. In the security architecture description, it is required to clarify the division of roles between the mechanisms realized by the security functions themselves and those offered by functions outside the product.

The reason is that security functions of the product and functions outside the product are handled differently in the CC evaluation. Security functions of the product are the subjects of the CC evaluation, such as the design, test, and vulnerability analysis. They also include mechanisms in the product that utilizes functions outside the product. On the other hand, functions outside the product themselves are not considered the subjects of the CC evaluation.

In the following sections, example methods of stating TSF self-protection as the security architecture description are introduced.

3.3.1 Self-protection mechanism by means of domain separation

As already explained in Section 3.2, although it is preferable that a domain separation mechanism is implemented in products, there are exceptional cases where domain separation is not implemented. This section is not applicable when domain separation is not implemented.

As for the domain separation explained in Section 3.2, additionally, the parts conducted in the evaluated product and the parts conducted outside the scope of the evaluation, such as functions outside the product, should be distinguished. Then, the parts conducted in the evaluated product as the TSF self-protection mechanism should be described in the security architecture description because such parts can be regarded as part of the security functions.

For instance, in the cases of the domain separation explained in Section 3.2 that uses the execution mode of the processor or the logical address spaces of processes, the contents can be described as follows:

- When domain separation is realized in the product
  In this case, all concrete mechanisms contributing to the separation of security domain will be described, such as the details of the processing that realizes the logical address spaces on a process-by-process basis and the details of the processing that realizes the execution mode management of the processor.

- When domain separation is realized depending on functions outside the product
  In this case, the usage in the evaluated product and its timing will be described with regard to the domain separation mechanism offered by functions outside the product.
  For instance, the following can be an example description. The evaluated product (security function) is executed at startup as a resident process with administrator privilege. When receiving an input from a user, the security function identifies and authenticates the user. After that, it generates a process for each user using system calls offered by the operating system, which are functions outside the product, and executes the programs acting on the user's behalf with general-user privilege.

3.3.2 Self-protection mechanism by means of ways other than domain separation

As already explained in Section 3.2, there are products with and without domain separation. This section is applicable to both cases.

With regard to input from users and programs acting on the user's behalf (hereinafter collectively referred to as "user side"), the mechanisms that protect the security functions from being tampered with should be described in the security architecture description. The following shows an example of the procedure.

(1) Specification of target interfaces for self-protection

First of all, all the interfaces through which the user side uses the protected assets will be identified without omission. Such interfaces include screen inputs for Web and other applications, network interfaces, and interfaces between programs in the product. Figure 3-3 shows the target interfaces for self-protection.

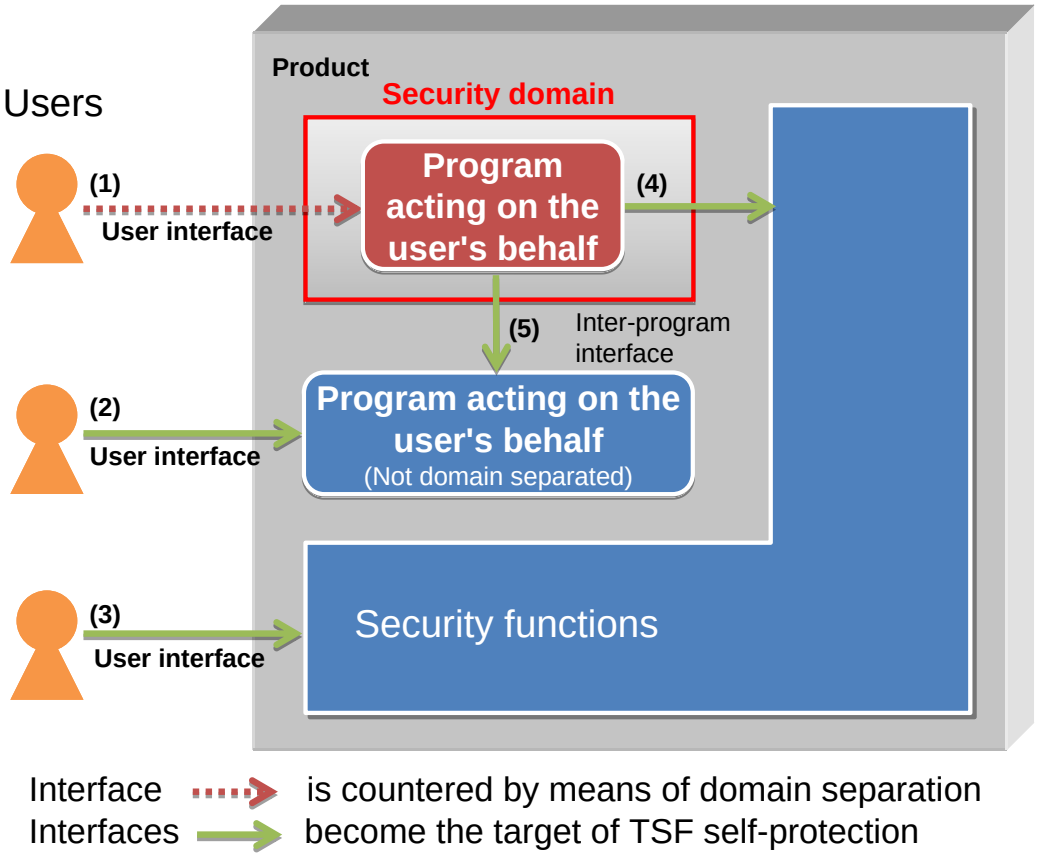


Figure 3-3: Target interfaces for self-protection

The interface (1) has no impact on the security functions because the connected program acting on the user's behalf is domain-separated. Therefore, this interface can be excluded from the target of self-protection.

All the other interfaces are connected to programs located outside the security domain; (2) and (5) are connected to a program acting on the user's behalf that is not connected to the security functions, and (3) and (4) are connected to programs of the security functions themselves, respectively. Those programs can interfere with each other because they are not domain-separated. In other words, regardless of whether the security functions are implemented to the connected program or not, all the interfaces (2) to (5) connected to programs that are not domain-separated become the target of self-protection.

When the product does not have a domain separation mechanism, self-protection mechanism has to be examined for all the interfaces.

(2) Description of the self-protection mechanism on an interface-by-interface basis

Next, the mechanism that protects the security functions from being tampered with for each of the specified interfaces will be specified, and then the contents will be described.

Interfaces must be equipped with a mechanism that can properly handle any input with no exceptions, even if they have out-of-specification inputs beyond the scope, size, or pattern defined in the design specification. Such mechanisms include the following:

- Buffer overflow measures

  Buffer overflow is a problem that occurs when a character input processing, etc., reads in a character string longer than the size of the buffer area prepared for reading input characters. The occurrence of buffer overflow poses the risks of an abnormal end of the program and the execution of an unexpected code infiltrated from the outside. As a countermeasure, the input processing must have such a restriction that prevents reading-in of character strings longer than the specified size into the buffer area. Those contents will be described in the security architecture description. Note that the countermeasure of checking the length of input characters alone will leave the possibility that buffer overflow may occur at the moment when the input characters are read-in for checking purposes.

- Prevention of the injection of scripts and commands

  When processing character strings entered from the outside, there is a risk of the execution of an unintended script or command contained in the character strings. Familiar examples include the injection of SQLs, operating system commands, and JavaScripts. Required processing as a countermeasure includes checking the entry of special characters for eliminating them or replacing them with other characters to avoid adverse impacts. An implementation without the need for error-susceptible concatenating processing of character strings (e.g., use of the bind mechanism of SQL) can also be presumed. Those contents will be described in the security architecture description.

- Countermeasures against an unauthorized memory area or file name specified by the user side

  The memory area or file name, in which input or output data is stored, can be

specified as a parameter for the use of an interface of the product. On this occasion, an unauthorized memory area or file name that is not supposed to be accessed by the user side may be specified, posing a risk that the area or file is read and written by the input-output processing in the product. As a countermeasure, checking validity such as the access privilege is required for the memory areas and file names specified by the user side. Those contents will be described in the security architecture description.

- Countermeasures against rewriting of memory areas specified by the user side
  When the product performs processing by referring to the content of the memory area specified as an input parameter several times, the user side may alter the content of the memory area during the processing. For instance, if the user side rewrites an input value to an authorized value after the product validates the input value, there is a risk that the rewritten unauthorized value may be used in subsequent processing in the product. A possible countermeasure can be to copy the content of the memory area specified as a parameter to a secure area that cannot be accessed by the user side when interface processing is called, and from this point onward, to perform various processing referring to the copied content. Those contents will be described in the security architecture description.

- Countermeasures against data formats or network protocols to be configured without authorization
  Even when a data format or network protocol is specified, the input data to the product is not necessarily valid. For instance, when a data format or network protocol handles variable-length data, an offset value from the top or a data length may be specified for indicating the ending position of the data. There is a risk that unauthorized values could be specified in the input data. If the product trusts the input data and performs the processing under such a circumstance, the product may access data out of the area, posing a risk of unexpected behavior. As a countermeasure, validity checking of the data content is required. Those contents will be described in the security architecture description.

- Others
  If the product has other mechanisms than those mentioned above for the prevention of adverse effects to the security functions (TSF) associated with the use of an interface from the user side, the contents should be described.

Examples include the confliction measures for interfaces using shared memory and the confliction measures for in-process data shared between threads. Those contents will be described in the security architecture description.

3.3.3 Confirmation of the descriptions

By comparing the mechanisms stated in the security architecture description with the specifications used in the actual product development, it should be confirmed that the contents are consistent and there are no omissions in the descriptions for each other.

All the mechanisms that contribute to the prevention of the tampering of the security functions, such as the prevention of the modification or suspension of the security functions and unexpected command execution, should be described.

3.4 TSF non-bypassability

In the item of TSF non-bypassability in the security architecture description, it is required to describe the mechanisms that ensure the application of the security functions for protecting the protected assets whenever the user accesses the protected assets.

To that end, it is required to demonstrate that all the interfaces of the product fall under either of the following, and that the product has no interface that is capable of bypassing the security functions.

- The security functions are applied to interfaces that are capable of accessing the protected assets without fail.
- Interfaces other than the above are not capable of accessing the protected assets.

In the security architecture description, the respective descriptions should be stated as follows, regarding the interfaces to which the security functions are applied when the protected assets are accessed (hereinafter referred to as "interface to the security functions") and other interfaces (hereinafter referred to as "interface irrelevant to the security functions").

- Interface to the security functions

  The mechanism should be described, which ensures the application of all security functions required for protecting the protected assets at a proper timing whenever the protected assets are accessed through the interface. This

description should also include the absence of the mode and setting in the interface for bypassing the required security functions.

- Interface irrelevant to the security functions
  The reason why the interfaces other than the above are irrelevant to the protected assets and security functions should be described. It would be insufficient to merely assert that there is no relevance. The irrelevance to the security functions should be clearly described to indicate a concrete processing mechanism.

In the following sections, example methods of stating TSF non-bypassability as the security architecture description are introduced.

3.4.1 Measures on the interfaces to the security functions

(1) Specification of the interfaces to the security functions

First, the interfaces that the user and programs acting on the user's behalf can use for accessing the protected assets should be specified. Such interfaces should have been implemented with security functions for protecting the protected assets, such as identification/authentication and access control. If there is an interface that can be used for accessing the protected assets without the intervention of security functions, it must be reviewed whether the interface and the security functions are appropriate or not.

Note that there are exceptional cases where the security is assured by a function outside the product, such as physical access control to the installation location, thereby allowing those interfaces which do not require the security functions by the product. Such cases will be discussed in Section "3.4.2 Interface irrelevant to the security functions."

(2) Description of the mechanisms that ensure the application of the security functions

Next, the mechanisms that ensure the application of the security functions to the interface should be described. The following are examples of descriptions.

- When the security functions are simple
  When the relationship between interfaces and security functions is simple, it would be sufficient to describe the processing in the security architecture description at a level commensurate with the simplicity. For instance, it will be

described that the processing is sequentially performed in response to a user input, in the order from the input parameter processing, to the security functions, the access to the intended protected assets, and the response of the processing results. Using the description, it should be made clear that there is no conditional branch that can bypass the security functions depending on the input parameter from the user and that the interface has been designed and implemented so that the user cannot change the order of processing.

If there is processing that disables the security functions referring a setting value inside the interface, a mechanism or operational measure is required for ensuring that the security functions are always enabled during operation. For example, it will be described that the setting of enabling/disabling can be accessed only by the administrators and not by general users, and also that the administrator's guidance manual has a description reminding that the product should be operated with the security functions enabled.

- When there are many paths that can be used for accessing the protected assets

When there are many paths that can be used for accessing the protected assets, a special mechanism may be required for ensuring that the security functions are applied to all the paths without omission.

As an example, security requirements of encrypting data are considered when the data is stored to a hard disk drive. A user can use various APIs and commands of the operating system for writing data to the hard disk drive. A mechanism will be required that can ensure the application of the security functions to all of them. In such cases, a commonly used method is to apply encryption to the input-output data of the hard disk drive at the device driver level, utilizing the fact that all the hard disk drive input-output offered by operating systems are performed through a device driver. By using this method, the security functions are applied without fail no matter what API or command the user uses. Those contents will be described in the security architecture description.

- When various operations are available for users

When a user can change the order of operations to the interface or parameters to input, a special mechanism may be required for ensuring that the security functions are applied without omission no matter what operation the user performs.

As an example, a case referring to the data to be protected in a Web system is considered. With a correct screen transition, the log-in screen appears first, and the intended data referencing screen can be accessed only when the identification

and authentication are successfully completed with the ID and password entered to the log-in screen. With Web browsers, however, the user can try to access any screen, in addition to the first log-in screen, by specifying the URL directly. By using the function, there is a risk of displaying data referencing screens that cannot be accessed without logging-in under normal conditions. A mechanism is required that can ensure the application of the security functions, such as identification/authentication and access control, no matter what URL is specified. As a countermeasure, it is required to have controls that maintain the state (session) in which identification and authentication were successfully completed and permit only the access of authorized sessions. Those contents will be described in the security architecture description.

Note that the session management of Web systems attracts many attackers' interest, so various attacks are known. Developers have to make consideration for countermeasures against attacks to the session management mechanism.

3.4.2 Measures on the interfaces irrelevant to the security functions

The interfaces explained in this section include all other interfaces that do not fall under the interfaces to the security functions explained in the previous section. The processing mechanism of the interface should be described so that the readers can understand the irrelevance between the security functions or protected assets, and the said interface. The following are example descriptions.

- When depending on domain separation

  When the program part activated by the said interface is domain-separated from the security function part and the protected asset part, the said interface is irrelevant to the security functions and protected assets. In this case, by specifying the security domain including the program part activated by the said interface, it will be described that bypassing is prevented by means of domain separation.

- When no special mechanism exists

  There is a risk that the program part activated by the said interface may, when it is not domain-separated, access the security functions or the protected assets via various paths. It will be described that such processing does not exist in the relevant program.

  For instance, by describing the processing of the program part activated by the said interface and the scope that its effects can reach (e.g., the scope of the data to be accessed, effects to other program part), it will be explained that no functions

reachable to the security functions or protected assets have been implemented.

- When depending on measures other than the product (exceptional cases)

  The reason should be described if no security functions for protecting the protected assets are required even though the interface can be used for accessing the protected assets. As an example, when the security depends on a function outside the product or operational measures, the dependent contents should be described, as well as the fact that the administrator's guidance manual has a description reminding that the dependent contents should be conducted without fail.

3.4.3 Interfaces that developers tend to fail to notice

Up to this section, the bypassing prevention mechanisms have been considered with respect to interfaces developed by developers and provided to users. However, interfaces not intended by the developers and interfaces undisclosed to general users also involve the risk of being exploited for bypassing the security functions. Developers are required to employ some measures for preventing the security functions from being bypassed through such interfaces. The measures will be described in the security architecture description. The following sections show examples of interfaces that are often overlooked.

(1) Example of interfaces not intended by the developers

- Interfaces of the operating system

  Operating systems offer various interfaces, including the physical memory space, logical address spaces on a process-by-process basis, and direct access to the devices. There is a risk that the protected assets are accessed from those interfaces. Possible measures include a description on the guidance manual reminding that those interfaces can be used only by the administrators and provided with an access privilege that prohibits general users from accessing it.

- Web server

  Web servers are equipped with a function of displaying the content when a directory or file name on the Web server is directly specified from a Web browser. The function poses a risk of disclosure of undisclosed information. As a countermeasure, attention should be given to, for example, the settings of the Web server and the placement of the contents (i.e., undisclosed information should not be placed in a public directory).

- Deciphering of cryptographic keys

When the protected assets are encrypted for ensuring confidentiality, there is a risk that attackers can decrypt the encrypted protected assets without using the decryption mechanism provided by the product if only they can know the cryptographic key in some way or another. For example, when generating a cryptographic key, the use of an algorithm without security assurance, as well as the easy use of time information or the serial number of the product, may lead to the deciphering or presumption of the cryptographic key. As a countermeasure, due considerations are required to the generation mechanism of cryptographic keys and the seed information used for generating cryptographic keys so that the cryptographic keys are not easily deciphered or presumed.

- Hidden channels

IC cards and similar devices have a risk that the cryptographic key is deciphered by means of analyzing the power waveforms instead of the regular input-output interfaces. When the password verification is so implemented as to compare to a password on a character-by-character basis, the password may be analyzed in a short time if the attacker knows how many characters had been entered before the verification failed by means of, for example, measuring the time required for the verification. Possible measures include devising a method of processing so that intended computations have no correlation to the power consumption and processing time in danger of being observed.

(2) Examples of undisclosed interfaces

- Maintenance interfaces

A product may have undisclosed interfaces for the purpose of maintenance, etc. Even though they are undisclosed to the public, they still have a risk of being found out and exploited by attackers. In such a case, it should be noted that keeping the usage of the interface confidential is not necessarily a sufficient countermeasure. As a countermeasure, due considerations are required for the security measures for preventing the interface from being exploited by means of, for example, identification and authentication, as well as strengthening the mechanism of identification and authentication at the same time.

- Debugging interfaces for development

If debugging interfaces that were used for the development remain in the product, they likewise pose a risk of being exploited for unauthorized access. As a

countermeasure, it should be confirmed that no unnecessary interfaces are left in the product.

3.4.4 Confirmation of the descriptions

As is the case with TSF self-protection, by comparing the mechanisms stated in the security architecture description with the specifications used in the actual product development, it should be confirmed that the contents are consistent. In addition, it should be confirmed that the non-bypassability has been demonstrated not only for identification/authentication and access control, but also for audit logging function and all the other security functions required for protecting the protected assets. Furthermore, it should also be confirmed that the non-bypassability has been demonstrated for all interfaces available for each of the security functions.

3.5 TSF secure initialization

In the items of TSF self-protection and TSF non-bypassability in the security architecture description previously mentioned, the states where the product is in operation are explained. On the other hand, in the item of TSF secure initialization in the security architecture description, the mechanisms that ensure the security should be described by focusing on the initialization process of the security functions in the period from the startup of the product triggered by power-on to the beginning of the operation mode.

Although the security measures during the initialization process tend to be overlooked, the security must be ensured even during the initialization process by means of some mechanisms, such as processing itself in the product and operational measures.

TSF secure initialization includes the following items.

(1) Specification of the initialization process of the security functions
By specifying the part of the product where the initialization process of the security functions is performed, the overview of the processing will be described. With respect to the initialization process specified in this stage, the mechanisms that ensure the security of the process will be described in the viewpoints explained in the next sections.

(2) Ensuring the integrity of the initialization process of the security functions to be initialized
The mechanisms that ensure the integrity of the security functions to be

initialized in the initialization process of the security functions specified above will be described.

(3) Protection of the protected assets during the initialization process

The mechanisms that prevent unauthorized access to the protected assets during the initialization process, in which the security functions have not yet been working, will be described.

(4) Prevention of exploiting the initialization process

The mechanisms that prevent exploiting the initialization process of the security functions by executing it in the operation mode will be described.

Figure 3-4 shows the overview of TSF secure initialization in accordance with the items (1), (2), (3), and (4) above.

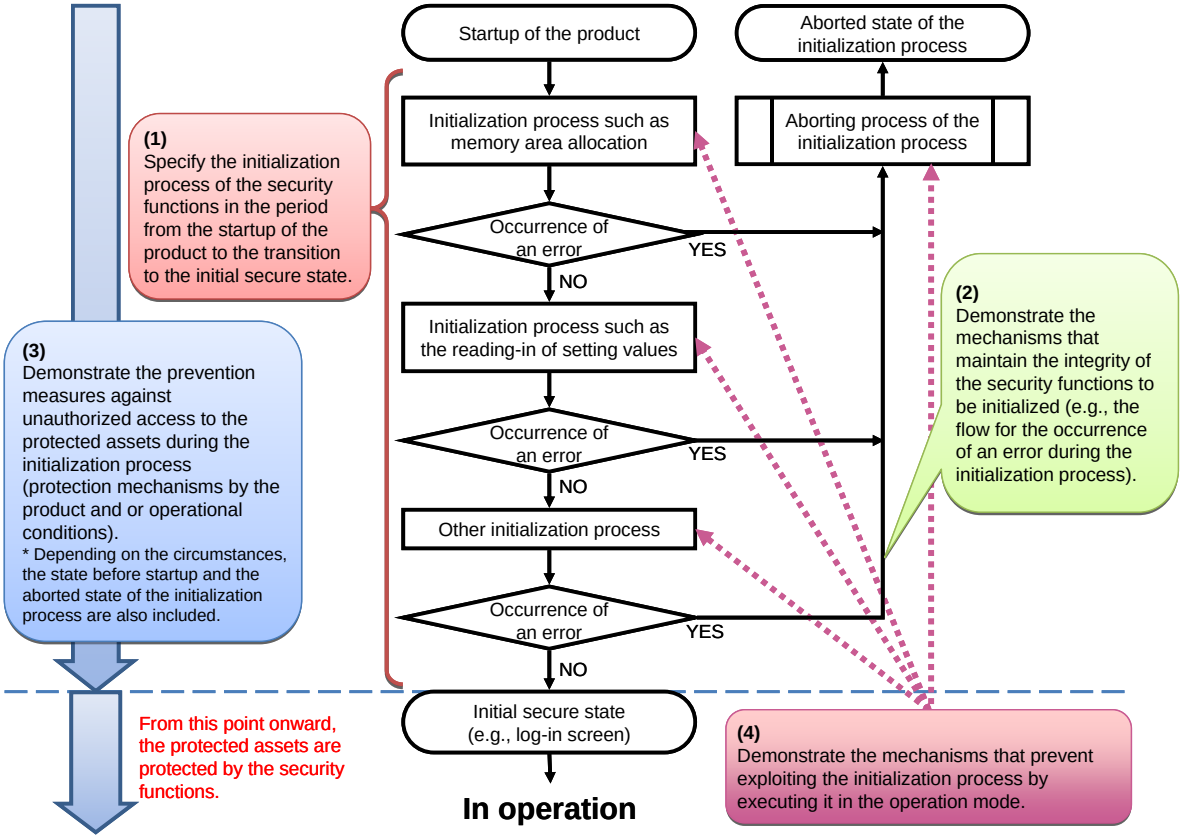


Figure 3-4: Overview of TSF secure initialization

In the following sections, example methods of stating TSF secure initialization as the security architecture description are introduced.

3.5.1 Specification of the initialization process of the security functions

The targets of the initialization process are the processing part in the product during the period from the startup of the product to the transition to the state where the entire part of the security functions (TSF) of the product is operable (hereinafter referred to as "initial secure state"). The scope covers not only the initialization process directly related to the security functions, but also the initialization process for all the mechanisms in the product for supporting the security functions, including the security architecture.

By specifying the initialization process as follows, the results should be described in the security architecture description.

(1) Specification of the startup methods of the product

First, the startup methods of the product should be specified. In the case of products containing hardware devices and software products controlling the entire hardware, the startup of the product is triggered by power-on or reset, etc. In the case of software products running on an operating system, they can be launched automatically at the start-up of the operating system or launched in accordance with an administrator's instruction after the start-up of the operating system.

In addition, when rebooting of the product or when the product is equipped with the functions of suspension and resume of the operation, the resume instruction are also included in the start-up methods of the product.

(2) Definition of the initial secure state

The state falling under the initial secure state varies depending on the product. The initial secure state after the start-up of the product (i.e., operable state) is defined in accordance with the properties of the product.

For example, it may include a state where a software product that is automatically launched at the power-on of the computer device displays a log-in prompt to a user, while it may include a state where the filtering processing enters the waiting-for-input state of network packet in a firewall product.

(3) Specification of the initialization process

Among the processing during the period from the startup of the product to the transition to the initial secure state, the process that falls under the initialization process of the security functions is specified.

It should be noted that every processing related to the security functions covered in the scope of the product can be the target when the relevant processing is specified.

For example, not only security function-specific processing, such as the reading-in of the setting values used by the security functions, but also the setup of the memory management in preparation for domain separation and other similar processing, as long as they are covered in the scope of the product, are included in the initialization process that should be stated in the security architecture description.

Depending on the product, the processing executed in the initial start-up can be different from that executed after reboot or resume. Paying attention also to those points, the initialization process part of the security functions during the period from the down state of the product to the transition to the initial secure state should be specified without omission.

(4) Description of the security architecture description

The start-up methods of the product and the definition of the initial secure state should be explicitly described in the security architecture description. In addition, the overview of the initialization process detailed enough to understand the correspondence to the design specifications should also be described.

3.5.2 Ensuring the integrity of the security functions to be initialized

In spite of the situation that the security functions were not working properly owing to a failure in the initialization process caused by tampering through unauthorized access or some factor during the period from the startup of the product to the transition to the initial secure state, it would be very dangerous if the administrator, without noticing such a situation, continued the operation of the product with the security functions working insufficiency. To prevent such a situation and to achieve a secure operating state, a mechanism for ensuring the integrity of the security functions will be required in the initialization process of the security functions.

With respect to this viewpoint, the following contents should be described in the security architecture description.

(1) Achievement of the initial secure state

The reading-in of the setting values for the security functions, the allocation of the memory area used by the security functions, and other various conditions have to be achieved for establishing the initial secure state. In the security architecture description, the mechanisms that can securely achieve the conditions required for the establishment of the initial secure state by means of the initialization process specified in the previous section will be described.

In the statement, it is insufficient to merely describe that successful completion of a series of the initialization process will result in the initial secure state. It is required to specify the factors with which the initialization process can fail and then to demonstrate that the security is not compromised under any circumstances.

In most cases, the initialization process is sequentially executed until the initial secure state is established. One thing it should be noted in this stage is the case where an error occurs in processing, such as a failure in the allocation of the memory area. In the initialization process, if the processing is continued regardless of the occurrence of an error owing to an insufficient checking of the return value from a function, the operation may be started with the security functions not working properly. In this case, therefore, the occurrence of errors in the return values of functions must be securely checked, and it is required to implement a mechanism that, for example, aborts the initialization process upon detection of an error.

(2) Protection of the aborted states during the initialization process

There is a risk that attackers may exploit the states where the initialization process has been aborted due to the occurrence of an error, etc. Depending on the operating system, for instance, the product can be switched into the command prompt mode with the administrator privilege, or the operating system can be booted in a special mode such as the Safe Boot option of Windows, which allows the attackers to use the product in a state where the security functions are not sufficiently working. To prevent such situations, it is required to implement mechanisms ensuring that the protected assets do not allow unauthorized access through the prevention of exploiting the aborted state during the initialization process. Examples of such measures include restricting operations available for the users to power-off in the aborted state during the initialization process. Note that the same countermeasures as the operational measures explained in Section 3.5.3 will be required when it is difficult to take countermeasures by means of the functions of the product.

(3) Cautions required for rebooting

The reboot and resume of the product should be handled in the same way as the case with "(1) Achievement of the initial secure state" in principle. However, it should be noted that problems specific to rebooting or resuming the product may occur. For instance, if the administrator rebooting the product omits some part of the initialization process based on the assumption that all the previous initialization process has successfully completed, the operation may be started with the security functions working insufficiently in case the previous initialization process was actually

incomplete due to the occurrence of an error.

Another example is a product that is designed to reset security-related settings at reboot, which are made by the administrators during the operating state, posing a risk that the administrators may start the operation without noticing that the settings have been reset at the reboot. Possible countermeasures in such a case include displaying a warning message so that the administrators can recognize the timing when the security-related settings are changed, such as reboot or operation resume of the product, as well as calling attention in a guidance manual.

3.5.3 Protection of the protected assets during the initialization process

In the middle of the initialization process, the protected assets may be accessed without authorization because the security functions have not entered the operation modes. With a firewall product, for instance, if the TCP/IP packet processing starts relaying TCP/IP packets before the filtering function starts the operation, there is a risk that the packets that are supposed to have been filtered are relayed without the application of the filtering function during the period from the power-on to the transition to the operation mode.

In addition, with a product originally intended that programs are started from the internal hard disk drive, there can be a risk of unauthorized access to the protected assets stored in the internal hard disk drive by starting another program from a USB memory, etc. There is a risk that attackers may launch another operating system stored in a USB memory for performing various operations with the administrator privilege; for example, by intentionally suspending the initialization process of the product for instructing that the product should be launched from the USB memory using BIOS, or by starting the product with the USB memory left inserted, depending on the BIOS settings.

The following are possible measures for preventing such unauthorized access.

(1) Measures by the product

The mechanisms should be described when measures by the product have been implemented. In most cases, the function to access the protected assets is not working in the middle of the initialization, and the access function to the protected assets starts working only after the security functions are ready for operation. As for a firewall product, for instance, the relaying of TCP/IP packets is prohibited in the initial state, and then the relaying of TCP/IP packets is permitted after the filtering becomes ready for operation.

In the security architecture description, the order of the processing steps performed in the initialization process should be concretely described in order to demonstrate that the protected assets cannot be accessed in the middle of the initialization.

(2) Operational measures

When measures by the product have not been taken, the obligations that the operations manager shall perform as the operational conditions of the product should be described.

Citing as an example of a firewall product previously mentioned, possible operational measures include the operations manager's compulsory attendance at the startup of the product for ensuring that the LAN cable is disconnected before the startup of the product and reconnected after the product has entered the operation mode. When operational measures where the operations manager attends only during the initialization process is assumed, as is the case with this example, it is also necessary to clarify the method to determine that the product has entered the operation mode, such as by checking the lamp display or the console messages of the device.

As another example, many server machines are required to be physically isolated in order to prevent anyone except for the operations manager from conducting unauthorized operations to the console of the machine.

With products running on a PC, it sometimes may be required to restrict the bootable devices in the BIOS settings during the initial settings so that the PC can be booted only from the internal hard disk drive, or to set a BIOS password for protecting the BIOS settings from being modified.

Note that those descriptions have to be stated not only in the security architecture description, but also in the guidance manual of the product for calling attention to the operations manager for certain implementation.

3.5.4 Prevention of exploiting the initialization process

To make the security functions ready for operation, the initialization process performs the setting of special data and hardware required for the operation of the security functions. There is a risk of the security functions being tampered with if those initialization processes are executed in the operation mode. To prevent such situations, the initialization process must be protected from being executed in the operation mode, and the mechanisms that realize such protection should be described in the security architecture description. It will be described that, for instance, although the initialization

process is sequentially executed at power-on, no interfaces that can be used for executing the initialization process are provided once the system has entered the operation mode.

When the initialization process is realized with programs running on the operating system, a mechanism that controls whether the initialization process has been already executed or not may be implemented to prevent the initialization process, which is supposed to be executed only once, from being executed multiple times by reexecuting such programs. Such descriptions will be described in the security architecture description.

If the system provides an interface that can be used for executing the initialization process even in the operation mode, the mechanisms for ensuring that attackers cannot tamper with the security functions by using the interface have to be described.

3.5.5 Confirmation of the descriptions

By comparing the mechanisms stated in the security architecture description with the specifications used in the actual product development, it will be confirmed that the initialization process has been described without omission, as is the case with the other security functions, and that the contents are consistent.

3.6 The level of detail in the security architecture description

As previously explained, the security architecture description is required to include the mechanisms that the product is equipped with in order to prevent bypassing and tampering of the security functions. This section explains at what level of detail the contents should be described.

In the CC evaluation, the security functions to protect the protected assets, such as identification/authentication and access control, are evaluated whether they have been accurately implemented and are capable of countering attacks on the basis of the design materials provided by the developers. The design materials required for the evaluation are determined according to the seven-stage evaluation assurance level (EAL). The higher the EAL is, the more detailed the information has to be. The same level of detailed information as the security functions to protect the protected assets is required for the mechanism of the security architecture. The following shows the level of details required for each of the EALs.

(1) EAL 1

At EAL 1, the external interface specifications of the product are evaluated.

Therefore, the security architecture description, which only includes the mechanisms inside the product, is not necessary for the CC evaluation.

(2) EAL 2 and EAL 3

At EAL 2 and EAL 3, the outline level of design information inside the product is evaluated, in addition to the external interface specifications of the product. With regard to the parts that realize the security functions for protecting protected assets, in particular, the design information must be so detailed that important data transferred between different functions and the outline of such processing can be clarified.

For instance, the design information should include the input-output parameters described in the external interface specifications, the setting parameters, the user IDs and other information retained in the product for access control after identification and authentication, and the outline of those processing.

At EAL 3, in addition to those, the design information should include the information about the important data used in the security functions. Also in the security architecture description, it is required to clarify the important data and the outline of the processing in the same level of details as those.

(3) EAL 4 or higher

At EAL 4 or higher, the design information that includes the detailed data structures and process flows at a program implementation level and the source codes are evaluated, in addition to the external interface specifications of the product and the outline level of design information inside the product. In other words, the design information must be so detailed that third parties other than the developers (evaluators) can interpret the source codes.

Also in the security architecture description, it is required to clarify the detailed data and the process flows in the same level of details as those.

Note that the security architecture description should include the coding rules because they may also contribute to the proof of the protection of the security functions. For instance, the rule for checking the length of data to be stored for preventing the data from exceeding the scope of the allocated input-output buffer falls under self-protection measures for the prevention of buffer overflow.

4 Conclusion

Security architecture is a mechanism to protect security functions themselves from unauthorized access to security functions for protecting protected assets. This guide has explained the contents of the security architecture description required for CC evaluations and how to describe them.

The contents of this guide include many reminders to the points that are often overlooked during the development and design regarding the security of products. When designing and developing a product, it would be useful for readers to realize more secure products by referring to this guide, regardless of having those products to be considered for the CC evaluation.

Note that developers should pay attention to the following.

- As the planning of the security functions for protecting protected assets is out of the scope of this guide, it does not include its explanation.
- The protection measures of the security functions may have already been realized in an execution environment including the employed operating system and existing libraries and frameworks. Possible options include domain separation by means of the execution environment and the adoption of a mechanism that can lead to countermeasures against problems, such as buffer overflow and SQL infection. After examining the mechanisms that can prevent the bypassing and tampering of the security functions and clarifying the dependence with those options in mind, the developers should state them in the security architecture description.
- Parts of the contents of the security architecture description that are realized by means of a mechanism of the product must be described in the design documents of the product in the same manner as general functions, and tested for confirming that they are working properly.
- Parts of the contents of the security architecture description that are realized by means of operational measures must be reminded in a guidance manual or other documents for ensuring the users' implementation.

Lastly, the following are references that would be helpful for examining and implementing security architecture.

- How to Secure Your Web Site
  http://www.ipa.go.jp/security/vuln/websecurity.html
- Secure Programming Course
  http://www.ipa.go.jp/security/awareness/vendor/programmingv2/index.html
- CEM "B.2.1 Generic vulnerability guidance"
  (For CEM, refer to "1.3 Common Criteria standard documents" in this guide.)

Addendum A Essential points for security architecture

Table A-1 below is a summary of the points of this guide, intended for use as a reference for examining and realizing security architecture. Note that this table does not assure the exhaustiveness of the requirements to be stated in the security architecture description.

Table A-1 Essential points for security architecture

#	Perspective	Overview	Summary	Section
1-1	Security domain (Domain separation)	- "Security domain" refers to a scope or environment, in which resources, such as memory areas that programs acting on the user's behalf can access without restrictions, are isolated. - "Domain separation" refers to confining the above mentioned resources, such as memory areas, in an isolated scope or environment. * Domain separation is realized by means of mechanisms other than the security functions of the product.	When the necessity or unnecessity of security domain is examined The necessity or unnecessity of security domain should be examined and realized for every part of the security functions to realize the security functional requirements.	3.2
1-2			When security domain is not necessary With respect to security functional requirements that do not need security domain, appropriate rationales should be given for the assertion that the absence of security domain will not adversely affect the realization of TSF self-protection and TSF non-bypassability as well as other protected assets and security functions.[*1]	
2-1	TSF self-protection	A mechanism to protect security functions of the product from being tampered with	Self-protection mechanism by means of domain separation The mechanisms or operational measures for protecting the product itself by means of domain separation should be examined and realized.	3.3 3.3.1
2-2			Self-protection mechanism by means of ways other than domain separation The mechanisms or operational measures for protecting the product itself without domain separation should be examined and realized. * There are cases such as user input where adverse effects cannot be prevented only by security domain.	3.3 3.3.2
3-1	TSF non-bypassability	A mechanism to ensure that the security functions for protecting the protected assets are applied at a proper timing without fail and cannot be bypassed	Interfaces to the security functions[*2] The mechanisms or operational measures for realizing the non-bypassability, in which all the security functions are applied at a proper timing without fail when the protected assets are accessed via the interface, should be examined and realized. The absence of the mode and setting in the interface for bypassing the security functions is also included.	3.4 3.4.1
3-2			Interfaces irrelevant to the security functions[*2] With respect to interfaces irrelevant to the security functions, what mechanisms or operational measures can prevent adverse effects to the protected assets and the security functions should be indicated without omission.	3.4 3.4.2
3-3			Interfaces that developers tend to fail to notice[*2] With respect to interfaces that can be used for bypassing the security functions of the product among the interfaces that developers tend to fail to notice, some measures should be examined and realized without omission.	3.4 3.4.3

4-1	TSF secure initialization	A mechanism to ensure that the security functions can be initialized in a perfect state and the product can enter the operation mode, preventing the compromise of the security during the initialization process in the period from the startup of the product to the transition to the operation mode of the product. (The protection in the period from the down state to the transition to the initial secure state is also included.)	Specification of the initialization process of the security functions After specifying the startup methods of the product, defining the initial secure state of the product, and specifying the part where the initialization process of the security functions is carried out toward the transition to that state, the outline of the processing should be described, including the steps in the middle of the transition as well as the occurrence of errors in the initialization process.	3.5 3.5.1
4-2			Ensuring the integrity of the security functions to be initialized The mechanisms or operational measures for ensuring the integrity of the security functions to be initialized in the initialization process of the security functions should be examined and realized.	3.5 3.5.2
4-3			Protection of the protected assets during the initialization process The mechanisms or operational measures for preventing unauthorized access to the protected assets even during the initialization process in which the security functions have not yet been working should be examined and realized.	3.5 3.5.3
4-4			Prevention of exploiting the initialization process The mechanisms or operational measures for preventing the programs for the initialization process of the security functions from being exploited after the product enters the initial secure state should be examined and realized.	3.5 3.5.4

(*1) Refer to Section 3.2.2 (2) for the examples that do not require security domain.

(*2) As is the case with #2 "TSF self-protection," there are two cases where domain separation is used and not used.

Security Architecture Guide for Developers

March 21, 2012 First edition

Author and Publisher Information-Technology Promotion Agency, Japan (IPA)

Writers Information Security Certification Office