JISEC

2018 年 4 月 18 日発行

開発者のためのセキュリティ解説書（セキュリティアーキテクチャ編）

IT製品がセキュリティ機能を備えていたとしても、攻撃者がそのセキュリティ機能を迂回したりセキュリティ機能自体を無効化してしまえば、製品全体としてセキュリティを守ることはできません。

開発者は、製品全体を俯瞰し、セキュリティ機能とともに、その機能自体を保護するための手段を講じた設計を行う必要があります。
それが「セキュリティアーキテクチャ」です。

独立行政法人
情報処理推進機構

技術本部セキュリティセンター
情報セキュリティ認証室
www.ipa.go.jp/security/jisec/

― 目次 ―

― 目次 ―

1 はじめに

本書で解説する「セキュリティアーキテクチャ」とは、IT 製品において、脅威に対抗するセキュリティ機能そのものを安全に実装するための、設計方針・プログラムの構造・しくみのことです。

IT セキュリティ評価の国際的な規格であるコモンクライテリア（Common Criteria, 以下「CC」といいます。）では、評価対象の IT 製品について、そのセキュリティアーキテクチャの妥当性を検査し、セキュリティ機能の安全性を保証するための評価方法が定められています。本書は、CC で要求されているセキュリティアーキテクチャの概念と、その設計内容を記述した文書（以下「セキュリティアーキテクチャ記述」といいます。）に記載すべき内容について、具体的な例をあげてわかりやすく解説します。

CC のセキュリティアーキテクチャの考え方は、堅牢で安全なセキュリティ機能を設計・実装するために不可欠なものです。本書が、CC の理解とともに、IT 製品のセキュリティ品質の向上に、参考になれば幸いです。

1.1 本書の対象読者

本書の対象読者は、IT 製品のセキュリティ機能を設計・実装する開発者や、CC に基づく IT 製品の評価を受けるためにセキュリティアーキテクチャ記述を作成する必要のある開発者を想定しています。

本書では、CC の内容を読者が理解し易いように平易な表現を用いて解説しており、CC 規格に対して正確ではない表現も含まれています。CC 評価認証の取得を予定している場合には、本書の「1.3 コモンクライテリア規格文書」に示した CC 関連の規格書を併読されることをおすすめします。

なお、本書では、IT 製品のセキュリティ機能を保護するための考え方を解説しています。IT 製品のセキュリティ機能自体や具体的な保護対策については、他の文献を参照してください。

1.2 本書の構成

本書の構成と各章の内容を説明します。

- 1 章　はじめに
  本書の目的、対象とする読者と適用範囲などについて述べています。

■ 2章　セキュリティアーキテクチャの基礎知識
セキュリティアーキテクチャの概念と必要性、理解する上で重要な概念「セキュリティドメイン」、「自己保護」、「非バイパス性」、「セキュアな初期化」について解説しています。

■ 3章　セキュリティアーキテクチャ記述
CC 評価において求められるセキュリティアーキテクチャ記述について具体的な記述方法を解説しています。

■ 4章　おわりに
本書で解説している内容について、全体をとおしての注意点を述べています。

1.3 コモンクライテリア規格文書

本書の評価基準及び評価方法は、以下の表 1-1 及び表 1-2 の規格文書に基づいています。評価基準は「CC」、評価方法は「CEM」という略称で呼ばれています。

表 1-1 CC/CEM の規格文書（日本語翻訳版）

CC / CEM バージョン 3.1 リリース 5	
評価基準 情報技術セキュリティ評価のためのコモンクライテリア（CC バージョン 3.1 リリース 5）	
	パート 1: 概説と一般モデル　バージョン 3.1 改訂第 5 版［翻訳第 1.0 版］
	パート 2: セキュリティ機能コンポーネント バージョン 3.1 改訂第 5 版［翻訳第 1.0 版］
	パート 3: セキュリティ保証コンポーネント バージョン 3.1 改訂第 5 版［翻訳第 1.0 版］
評価方法 情報技術セキュリティ評価のための共通方法（CEM バージョン 3.1 リリース 5）	
	評価方法　バージョン 3.1 改訂第 5 版［翻訳第 1.0 版］

表 1-2 CC/CEM の規格文書（原文）

CC / CEM v3.1 Release5			
評価基準 Common Criteria for Information Technology Security Evaluation（CC v3.1 Release5）			
	Part 1: Introduction and general model	Version 3.1	Revision 5
	Part 2: Security functional components	Version 3.1	Revision 5
	Part 3: Security assurance components	Version 3.1	Revision 5
評価方法 Common Methodology for Information Technology Security Evaluation（CEM v3.1 Release5）			
	Evaluation methodology	Version 3.1	Revision 5

参照：評価基準 Common Criteria, IPA
https://www.ipa.go.jp/security/jisec/cc/index.html

本書は、CC/CEM の規格文書「CC バージョン 3.1 リリース 5 パート 3」、「CEM バージョン 3.1 リリース 5」のうち、以下に記載された内容に基づいています。

- CC Part 3,「13 ADV クラス:開発」, 226-227 段落
- CC Part 3,「13.1 セキュリティアーキテクチャ(ADV_ARC)」
- CC Part 3, 附属書 A 開発(ADV),
  「A.1 ADV_ARC: セキュリティアーキテクチャに関する補足資料」
- CEM,「12.3 セキュリティアーキテクチャ(ADV_ARC)」

1.4 用語集

本書で使用する CC/CEM に関係する用語を表 1-3 に示します。

表 1-3 CC/CEM 用語

用語	説明
CC (Common Criteria：コモンクライテリア)	情報セキュリティの観点から、IT 製品が適切に設計され、その設計が正しく実装されていることを評価するための国際標準規格「ISO/IEC 15408」のことです。
CEM (Common Evaluation Methodology：共通評価方法)	CC に基づいたセキュリティ評価を均質に行うために定められた評価の方法のことです。CC 規格を満たすための、評価すべき項目や評価の観点が定められています。
EAL (Evaluation Assurance Level：評価保証レベル)	CC に基づいたセキュリティ評価の保証の度合いを表します。EAL1 から EAL7 の 7 段階が定義されています。EAL が高くなるほど、製品の広範囲の設計情報等が厳格に評価されます。
セキュリティターゲット (Security Target)	CC に基づいたセキュリティ評価のために、評価対象の IT 製品について記述した文書のことです。IT 製品の評価範囲、前提条件、満足しなければならないセキュリティ機能要件、評価保証レベルなどが記述されています。セキュリティターゲットは、IT 製品の調達者の要求に基づいて、IT 製品の開発者が作成します。

2 セキュリティアーキテクチャの基礎知識

本章では、「セキュリティアーキテクチャ」を理解するために必要な基礎知識として、セキュリティ機能とそれを保護するセキュリティアーキテクチャの関係、及び、セキュリティアーキテクチャに求められる特性を説明します。

2.1 セキュリティ機能とは

セキュリティ機能とは、利用者の重要データ等の保護すべき資産（以下「保護資産」といいます。）を不正なアクセスから守るための機能や、組織のセキュリティ方針（たとえば、「監査ログを取得しなければならない」など、必ずしも不正アクセス防止とは直接結びつきませんが、組織の方針として求められるセキュリティに関する要件のこと）を実現するための機能です。

たとえば、保護資産への不正なアクセスを防止するための識別認証機能やアクセス制御機能、保護資産の秘密情報を守るための暗号化機能、セキュリティ機能の動作状況を監視するための監査ログ機能などが、セキュリティ機能に該当します。

保護資産が、攻撃者による不正アクセスからセキュリティ機能によって守られている概念を、図 2-1 に示します。

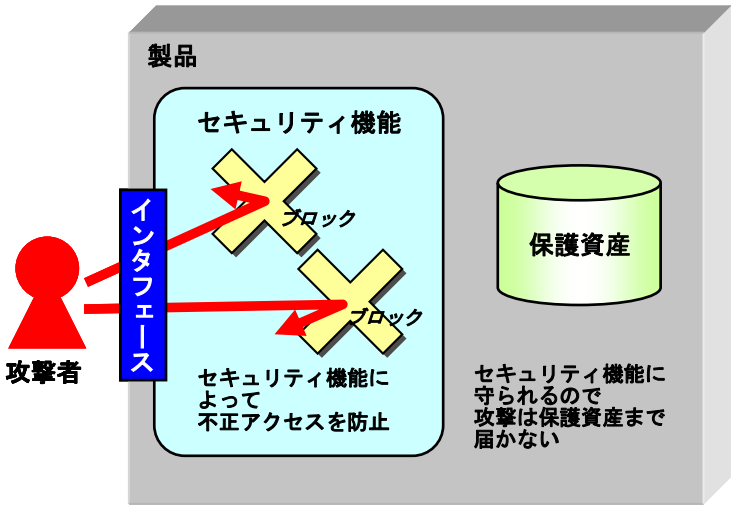


図 2-1 セキュリティ機能による保護

図 2-1 において、「インタフェース」は、保護資産を利用するために製品が想定している正当なインタフェースを表しています。

2.2 セキュリティ機能に対する攻撃とは

セキュリティ機能に対する攻撃とは、攻撃者がセキュリティ機能の欠陥を悪用して、製品の設計上は本来意図されていない方法で、保護資産に不正にアクセスすることです。セキュリティ機能に対する攻撃には、「バイパス（回避）」と「改ざん」の二種類があります。

■ バイパス（回避）

セキュリティ機能の「バイパス」とは、本来であれば製品の使用時に適用されるはずのセキュリティ機能が適用されないで回避されてしまうことを指します。

攻撃者がセキュリティ機能をバイパスして保護資産に不正アクセスする概念を、図 2-2 に示します。

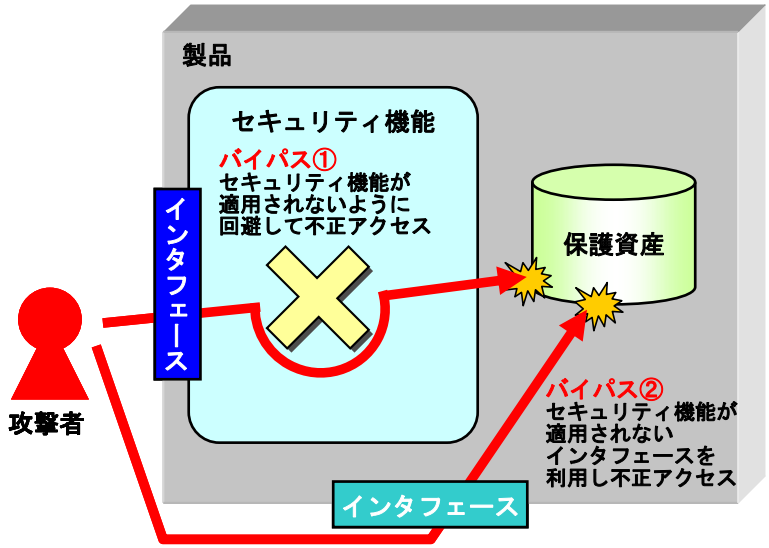


図 2-2 セキュリティ機能のバイパスによる不正アクセス

図 2-2 で、バイパス①は、セキュリティ機能が適用されるべき正当なインタフェースに、セキュリティ機能の適用を回避して保護資産にアクセス可能な経路が存在する場合を示しています。また、バイパス②は、正当なインタフェースの他に、セキュリティ機能が適用されない別のインタフェースが存在する場合を示しています。

具体的には、たとえば以下のような場合がバイパスに該当します。

- 攻撃者が、Web 画面でアクセス制御に用いられているパラメタを他人の値に書き換えることによって、他人になりすまして保護資産にアクセスできてしまう場合

・ 攻撃者が、Web ブラウザでログイン画面を経由せずに保護資産の URL を直接指定することによって、アクセスが許可されていない保護資産にアクセスできてしまう場合

・ 攻撃者が、秘密の暗号鍵を推測したり暴露したりすることによって、暗号化された保護資産を復号できてしまう場合

■ 改ざん

セキュリティ機能の「改ざん」（注：「tampering」の訳）とは、セキュリティ機能を改変したり破壊したりする攻撃を指します。

攻撃者がセキュリティ機能を改ざんして保護資産に不正アクセスする概念を、図 2-3 に示します。

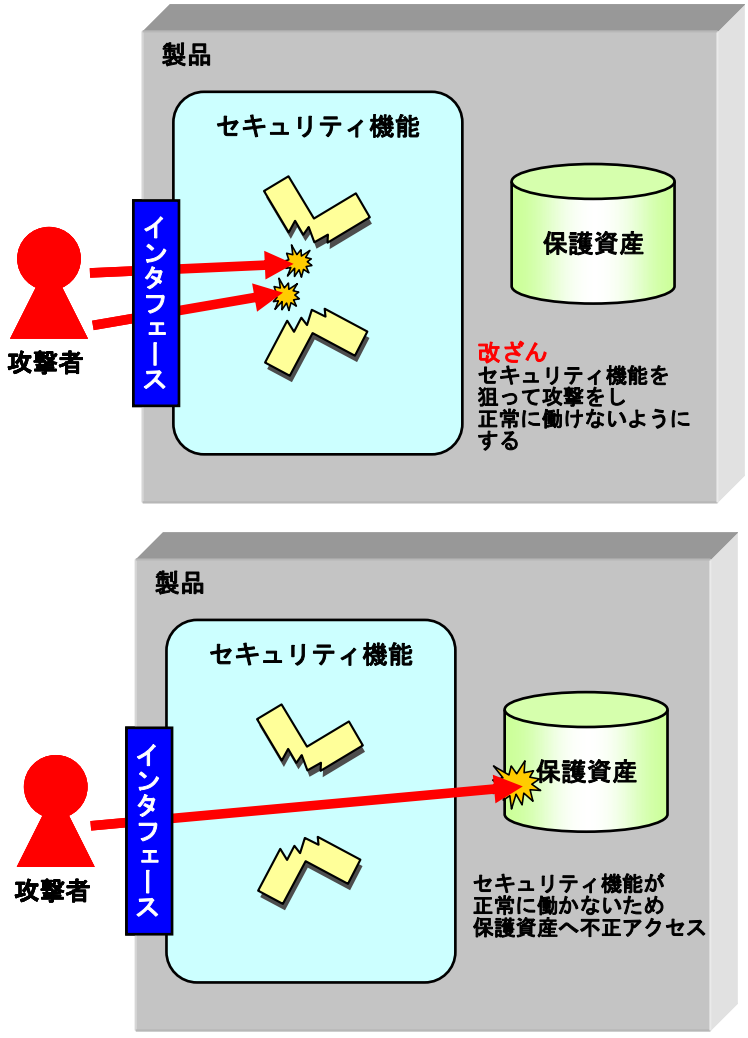


図 2-3 セキュリティ機能の改ざんによる不正アクセス

図 2-3 は、攻撃者が、まず、製品に異常なデータを入力することによってセキュリティ機能を破壊し、続いて、セキュリティ機能が働かない状態で保護資産にアクセスする場合を示しています。

具体的には、たとえば以下のような場合が改ざんに該当します。

- 攻撃者が、バッファオーバフローを引き起こすデータを入力することによって、セキュリティ機能のデータを破壊し、送り込んだプログラムを実行する場合
- 攻撃者が、製品の何らかの欠陥を悪用して監査ログを記録するプロセスを停止することによって、不正なアクセスの痕跡が残らない状態で製品を使用する場合

2.3 セキュリティアーキテクチャによるセキュリティ機能の保護

製品にセキュリティ機能が実装されていても、保護資産を守るはずのセキュリティ機能自体が攻撃されて役に立たなくなってしまうのであれば、セキュリティ機能の意味がありません。製品のセキュリティ機能を設計・実装するにあたっては、識別認証やアクセス制御といったセキュリティ機能だけでなく、それらのセキュリティ機能自体を攻撃から守るためのしくみ、つまり、「セキュリティアーキテクチャ」が不可欠です。

CC では、セキュリティ機能のバイパスや改ざんを防止するために、セキュリティアーキテクチャが実現すべき特性として、次の 4 つの観点が定められています。

・セキュリティドメイン
・自己保護
・非バイパス性
・セキュアな初期化

以下に、これらの 4 つの観点について、説明します。

2.3.1 セキュリティドメイン

(1) セキュリティドメインの概念

セキュリティドメインは、セキュリティ機能や保護資産に対して悪影響を及ぼす可能性のあるプログラムの動作を、ある限られた範囲内に封じ込めてしまう概念です。

悪影響を及ぼす可能性のあるプログラムには、外部からのダウンロード等で製品の内部に取り込まれたプログラムや、製品の利用者が新規に作成したプログラムだけでなく、製品の中に組み込まれているプログラムで利用者の操作に応じて動作する部分も該当します。

それらは、正当なプログラムであっても、プログラムの実装ミス、攻撃者の不正な操作、利用者の操作ミスなどにより、セキュリティ機能や保護資産に悪影響を及ぼしたり、コンピュータウイルスなどの不正なプログラムによる攻撃に悪用されたりする可能性があります。

悪影響を及ぼすプログラムの動作の概念を図 2-4 に示します。

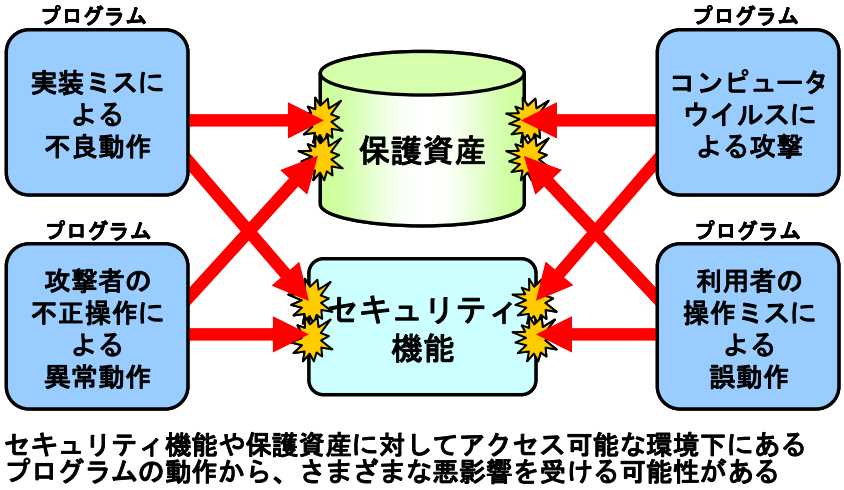


図 2-4 セキュリティ機能や保護資産に悪影響を与えるプログラムの動作

図 2-4 では、各プログラムは何の制限もなくセキュリティ機能や保護資産にアクセスすることが可能であり、それが悪影響を及ぼす原因になっています。

それに対して、悪影響を及ぼす可能性のあるプログラムが自由にアクセスできる範囲を制限した場合、つまり、セキュリティドメインの概念を図 2-5 に示します。

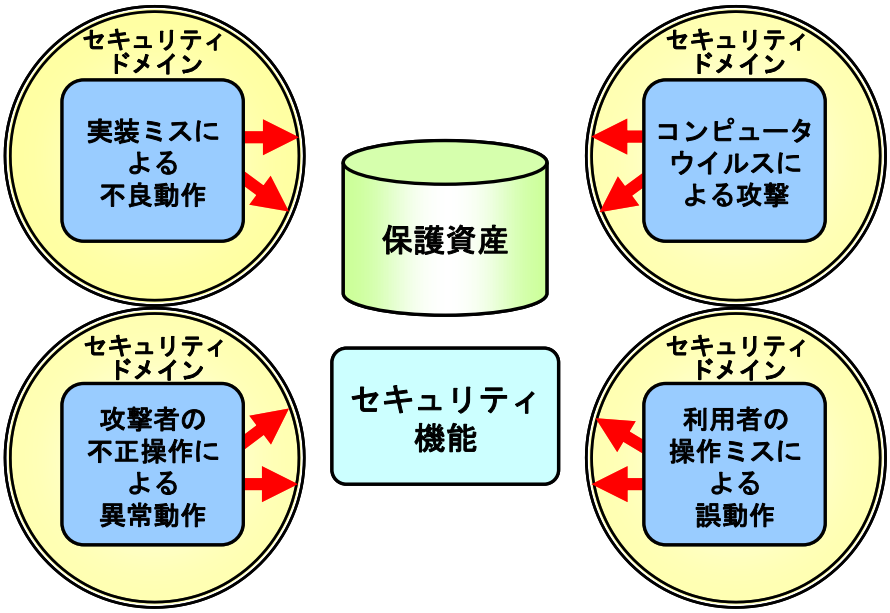


図 2-5 セキュリティドメインの概念

図 2-5 では、各プログラムが自由にアクセスできる範囲は、各プログラムのセキュリティドメイン内に制限されており、セキュリティドメインの外部に悪影響を及ぼすことはできません。

(2) セキュリティドメイン及びドメイン分離の実現方法

プログラムが自由に読み書きできるメモリ領域等の環境について、それが制限された範囲が「セキュリティドメイン」であり、その範囲を制限し他プログラムのセキュリティドメインにアクセスできないようにすることを「ドメイン分離」といいます。

たとえば、一般的なオペレーティングシステムでは、利用者プログラムはプロセスとして実行が制御され、プロセスごとに別々のアドレス空間が提供されます。利用者プログラムが自由に読み書きできるメモリ範囲は、自身のプロセスのアドレス空間だけであり、他のプロセスのアドレス空間にあるメモリは参照できません。その場合、プロセスごとのアドレス空間が、セキュリティドメインに該当します。

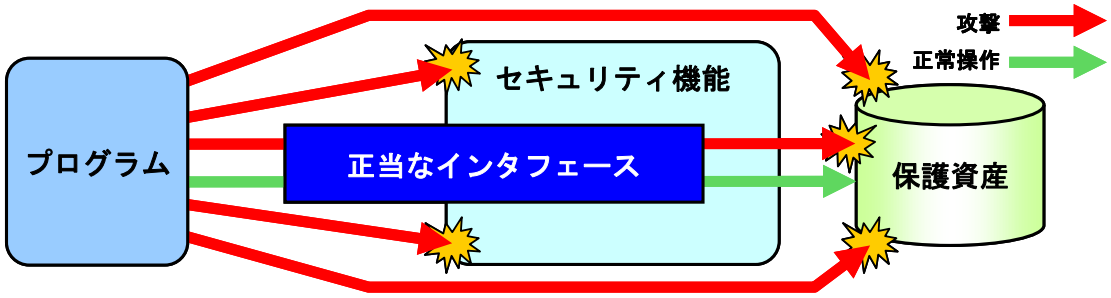
セキュリティドメインを分離する方法、すなわち、ドメイン分離の実現方法としては、たとえば、利用者が使用するアドレス空間とシステムが使用するアドレス空間を明確に分離するハードウェア機構やアドレス管理方式などがあります。また、製品が、ハードウェアやオペレーティングシステムを含まないアプリケーションプログラムである場合には、製品の範囲に含まれていないオペレーティングシステムの提供する

しくみを利用して、ドメイン分離を実現している場合もあります。

(3) ドメイン分離の効果

セキュリティドメインは、セキュリティ機能がバイパス・改ざんされないように設計・実装する上で、基盤となる考え方です。

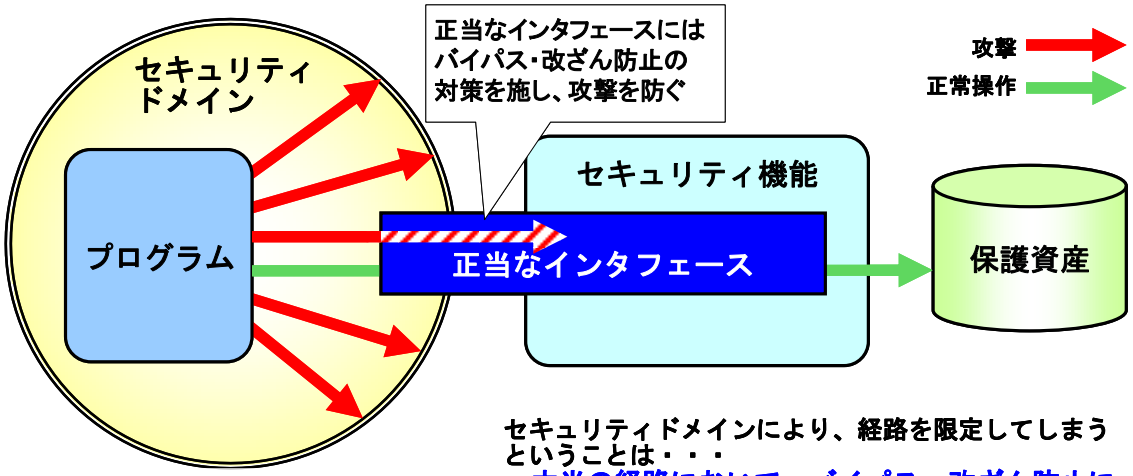
セキュリティ機能のバイパス・改ざん防止の対策について、図 2-6 にドメイン分離されていない場合を、図 2-7 にドメイン分離されている場合を示します。



プログラムのアクセス可能な範囲が自由だと
正当なインタフェースへの経路の他にも
領域外のメモリアクセスなど攻撃が通る
経路が無数にある状態

経路が無数に存在してしまうということは・・・
→すべての経路に対し、バイパス・改ざん防止の
対策が個別に必要となるため、作業量も多く、
攻撃経路の見落としや対策方法の考慮漏れなど
発生する可能性も高くなる

図 2-6 ドメイン分離されていない場合



正当なインタフェースへの経路を残して
プログラムのアクセス可能な範囲を
セキュリティドメイン内に限定させる

セキュリティドメインにより、経路を限定してしまう
ということは・・・
→大半の経路において、バイパス・改ざん防止に
つながるので、結果的に個々の対策が不要となり、
効率が良いと言える

図 2-7 ドメイン分離されている場合

図 2-6 のドメイン分離されていない場合は、プログラムによる攻撃の経路が無数に

存在します。そのため、それらの攻撃への対策を個別に行なう必要があり、攻撃経路の見落としや対策の考慮漏れなどが発生する可能性も高くなります。

それに対して、図 2-7 のドメイン分離されている場合は、プログラムによる攻撃の経路が、正当なインタフェースに限定されています。そのため、正当なインタフェースにおいてセキュリティ機能のバイパス・改ざんを防止する対策を実施することで、プログラムによる攻撃を一括して対処することができ、図 2-6 のような個別の対策が不要になります。

このように、ドメイン分離を用いることで、セキュリティ機能のバイパス・改ざんの防止を、堅牢かつ効率的に実現することができます。セキュリティ機能を保護するためには、セキュリティドメインの考え方を採用し、ドメイン分離のしくみを実装することが推奨されます。

2.3.2 自己保護

(1) 自己保護の概念

自己保護とは、セキュリティ機能以外のプログラムや製品の利用者によってセキュリティ機能が改ざんされないように、製品のセキュリティ機能が自分自身を保護するしくみのことです。

製品のセキュリティ機能を改ざんする可能性としては、製品外部の利用者やプログラムによる場合と、製品内部のセキュリティ機能以外のプログラムによる場合の 2 通りがあります。開発者は、両方の場合を考慮することが必要です。

(2) 自己保護の実現方法

自己保護は、「2.3.1 (3)ドメイン分離の効果」で説明したように、通常、ドメイン分離、及び、ドメイン分離では対処できないインタフェース等の保護対策で実現します。

インタフェースの保護対策は、たとえばバッファオーバフローや SQL インジェクションなどの、インタフェースに不正な入力を行う攻撃に対する対策が該当します。それらの攻撃内容に応じて、個別の保護対策が必要となります。

開発者は、セキュリティ機能への攻撃の可能性のあるすべての経路と改ざん方法を認識し、ドメイン分離やそれ以外のしくみを利用して、改ざんを防止する対策を漏れなく設計・実装する必要があります。

2.3.3 非バイパス性

(1) 非バイパス性の概要

非バイパス性とは、製品を利用する際に適切なタイミングで必ずセキュリティ機能が適用されて、そのセキュリティ機能がバイパスされないという、セキュリティ機能が備えるべき特性のことです。

バイパスの経路には、製品にセキュリティ機能が適用されないインタフェースが存在する場合と、セキュリティ機能内部にそれが適用されない利用方法が存在する場合の 2 通りがあります。開発者は、両方の場合を考慮することが必要です。

(2) 非バイパス性の実現方法

非バイパス性は、開発者がセキュリティ機能を設計・実装する際に、バイパス経路が生じないように注意して設計・実装することで実現されます。

たとえば、開発者は次のような点に注意が必要です。

■ セキュリティ機能の適用漏れの防止

開発者は、セキュリティ機能以外のプログラムや製品の利用者が、セキュリティ機能の保護すべき資産にアクセス可能なすべての経路について、セキュリティ機能の適用漏れがないように設計・実装する必要があります。

なお、保護資産に至るアクセス経路が多く存在すればするほど、すべての経路においてセキュリティ機能の適用漏れがないことを保証することはむずかしくなります。そのような場合には、「2.3.1 (3) ドメイン分離の効果」で説明したようにドメイン分離を利用すると、保護資産にアクセス可能な経路は限定され、セキュリティ機能の適用漏れを防止することは容易になります。

■ セキュリティ機能内部におけるバイパス防止

開発者は、セキュリティ機能が適用されるインタフェースにおいて、想定外の利用順序の変更や想定外のパラメタ入力などにより、セキュリティ機能が適用されなくなってしまう処理がセキュリティ機能内部に存在しないように設計・実装する必要があります。

たとえば、Web アプリケーションでは、正当な画面遷移とは異なる画面にアクセスされたり、セッション管理に使用されるパラメタを他人の値に変更されたりしても、識別認証やアクセス制御などのセキュリティ機能が間違いなく適用されるように、特別な対策が必要になります。

2.3.4 セキュアな初期化

(1) セキュアな初期化の概念

セキュアな初期化とは、起動途中の製品のセキュリティを確保するしくみのことです。

製品が起動途中の状態の場合、まだセキュリティ機能が正常に動作しておらず、攻撃に対抗できなかったり、セキュリティに影響がある特別な機能が一時的に使えたりすることがあります。この起動途中の状態は、一般に安全対策が見落とされがちであり、攻撃者にとって絶好の攻撃の機会となってしまいます。

製品の起動途中では、たとえば以下のような危険な状態になる恐れがあります。

■ セキュリティ機能が適用されない場合

たとえば、ファイアウォールの起動途中において、フィルタリング機能が起動するよりも先に通信データの中継機能が起動してしまうと、不正な通信データが遮断されずファイアウォールを素通りされてしまう恐れがあります。

■ 特別な利用モードが存在する場合

たとえば、OS の起動途中において、特別な操作が行われたり障害が発生したりすると、通常の運用とは異なり保守用の特別な利用モード（セーフモードやシングルユーザモードなど）になったり、USB メモリ等から別のプログラムを起動できる場合があります。そのような利用モードが攻撃者によって悪用されると、管理者パスワードが変更されたり、機密データが外部記憶媒体へコピーされたりするなどの恐れがあります。

上記のような危険性があるため、開発者は、製品を起動してから運用状態に至るまでの間であっても、製品が攻撃に対抗し、セキュリティ機能の初期化処理が正しく行われるしくみを設計・実装しなければなりません。

(2) セキュアな初期化の実現方法

セキュアな初期化は、開発者がセキュリティ機能を設計・実装する際に、運用状態だけでなく製品の起動途中にも注意を払い、セキュリティ機能が有効になる前に、保護資産にアクセス可能な経路や機能が存在しないように、設計・実装することで実現されます。

3　セキュリティアーキテクチャ記述

本章では、CC 評価で開発者に要求される文書である「セキュリティアーキテクチャ記述」について、具体的な記載方法や事例をまじえながら、記述すべき内容を説明します。

3.1　セキュリティアーキテクチャ記述の内容

セキュリティアーキテクチャ記述には、「信頼できない能動的なエンティティ」によって、製品のセキュリティ機能がバイパスされたり、改ざんされたりすることを防止するために、製品がどのように設計及び実装されているのかを、2 章で説明した以下の 4 つの観点に整理して記述します。

・セキュリティドメイン
・自己保護
・非バイパス性
・セキュアな初期化

ここで、「信頼できない能動的なエンティティ」とは、セキュリティ機能や保護資産に対して悪影響を及ぼす可能性のある利用者やプログラムのことです。典型的には、利用者または製品とやりとりを行う外部のプログラム（以下これらを「利用者」といいます。）、及び、利用者が製品を操作する際にその操作に応じて製品内部で動作するプログラム（以下「利用者代行プログラム」といいます。）が該当します。

以下に、4 つの観点の記述方法を説明します。

3.2　セキュリティドメイン

セキュリティアーキテクチャ記述では、セキュリティドメインの項目において、製品に実装されているセキュリティドメインの定義と、ドメイン分離のメカニズムを記述する必要があります。特に、以下の内容を明確に記述することが重要です。

■ セキュリティドメインの定義

利用者代行プログラムが制限なくアクセスできる範囲に含まれている資源を定義します。典型的には、プロセスごとに与えられるアドレス空間が該当します。また、JavaVM などの virtual machine において、その上でプログラムが隔離された環境（一般に「sandbox」と呼ばれています）で実行される場合、その隔離された環境に含まれる資源も該当する可能性があります。利用者代行プログラムのセキュリティドメインを定義する際には、その範囲内に、セキュリティ機能やセキュリティ機能で保護すべき資産を含まないように注意します。

■ ドメイン分離のメカニズム

上記のセキュリティドメインの定義に基づき、利用者代行プログラムがセキュリティドメインの範囲外にアクセスできないように制限するメカニズムを明確にします。

製品によって、提供する機能や使用方法、及びプログラムの実現方法は様々であり、それらに応じて、セキュリティドメインの捉え方や、ドメイン分離のメカニズムも異なります。それらの製品の特性を考慮して、製品の開発者は適切なセキュリティドメインを設計し実装しなければなりません。

以下では、より深い理解のために、製品においてセキュリティドメインを特定し、その事項をセキュリティアーキテクチャ記述に記載する方法の例を紹介します。なお、以下の方法はあくまで一例であり、他の手法もあることに注意してください。

3.2.1 セキュリティドメインの特定

本節では、利用者代行プログラムがセキュリティ機能の配置されているメモリ領域に対してアクセスを行おうとした場合に、どのようなメカニズムで防止されるかに着目して、セキュリティドメイン及びドメイン分離のメカニズムを特定する方法を紹介します。以下、具体的な手順について、例をあげて説明します。

(1) 利用者代行プログラムとセキュリティ機能の特定

まず、製品の利用者代行プログラムとセキュリティ機能を特定します。製品には一般に、製品の主目的である利用者向け機能と、利用者からのアクセスを制限し不正なアクセスを防止することを目的としたセキュリティ機能が存在します。

たとえば、インターネットバンキングシステムであれば、利用者の口座の残高を参照したり振込などを行ったりする機能は利用者向け機能であり、ログイン画面などにより利用者を識別認証し他人の口座の参照などができないように制限する機能はセキュリティ機能です。製品の中で、前者の利用者向け機能を実現している利用者代行プログラム部分と、後者のセキュリティ機能を実現しているプログラム部分を区別して特定します。

続いて、製品の利用者代行プログラムとセキュリティ機能のプログラムが配置されている実装形態を特定します。一般に、プログラムの実装形態は、それぞれ、以下のどれかにあてはまると考えられます。

（ア）OS のカーネル（デバイスドライバなど）として実装
（イ）プログラムごとにプロセスとして実装
（ウ）プロセスの中で、プログラムごとにスレッドとして実装
（エ）複数のプログラムから共通的に利用可能な、共有ライブラリとして実装

たとえば、セキュリティ機能が(ア)で利用者代行プログラムが(ウ)であったり、セキュリティ機能・利用者代行プログラムともに(イ)であったりなど、製品によって実装形態は様々です。

(2) ドメイン分離の判断

次に、利用者代行プログラムとセキュリティ機能が、それぞれ、(1)で特定した実装形態（ア）～（エ）のどれに該当するのかを考えた上で、セキュリティ機能の配置されているメモリ領域が、利用者代行プログラムから保護されているかどうかを検討します。

一般に、セキュリティ機能の配置されているメモリ領域は、以下のようなメカニズムまたはそれらの組合せで保護されています。これらのメカニズムの下では、利用者代行プログラムは、どのような動作をしても、セキュリティ機能に影響を与えることはできません。すなわち、ドメイン分離されていることになります。

■ プロセッサの実行モードによるドメイン分離メカニズム

セキュリティ機能は OS のカーネル内に実装されており、利用者代行プログラムは OS 上で動作する一般プロセスとして実装されている場合が該当します。図 3-1 にドメイン分離の実装例を示します。

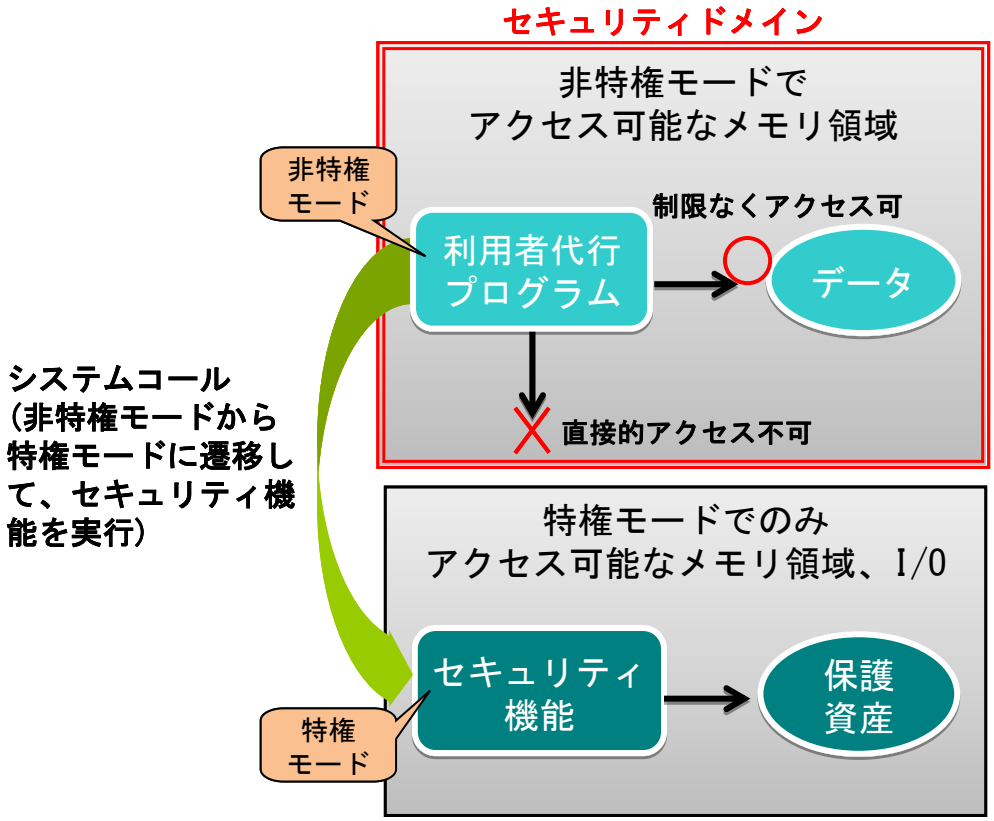


図 3-1 ドメイン分離の例（プロセッサの実行モードによる場合）

この場合、OS では、プロセッサの備える実行モードを利用して、一般プロセスが利用者代行プログラムを実行する際は、プロセッサの非特権モードで実行するように制御します。一方、セキュリティ機能が存在するカーネルは、プロセッサの特権モードでのみ実行や読み書きを許可するように制御します。それにより、一般プロセスからは、実行や読み書きが自由にできる領域は、非特権モードで許可された範囲に限定され、特権モードが必要なカーネル領域の実行や読み書きはできなくなります。

■ プロセスの論理アドレス空間によるドメイン分離メカニズム

セキュリティ機能はプロセスとして実装されており、利用者代行プログラムは、セキュリティ機能とは別のプロセスとして実装されている場合が該当します。図 3-2 にドメイン分離の実装例を示します。

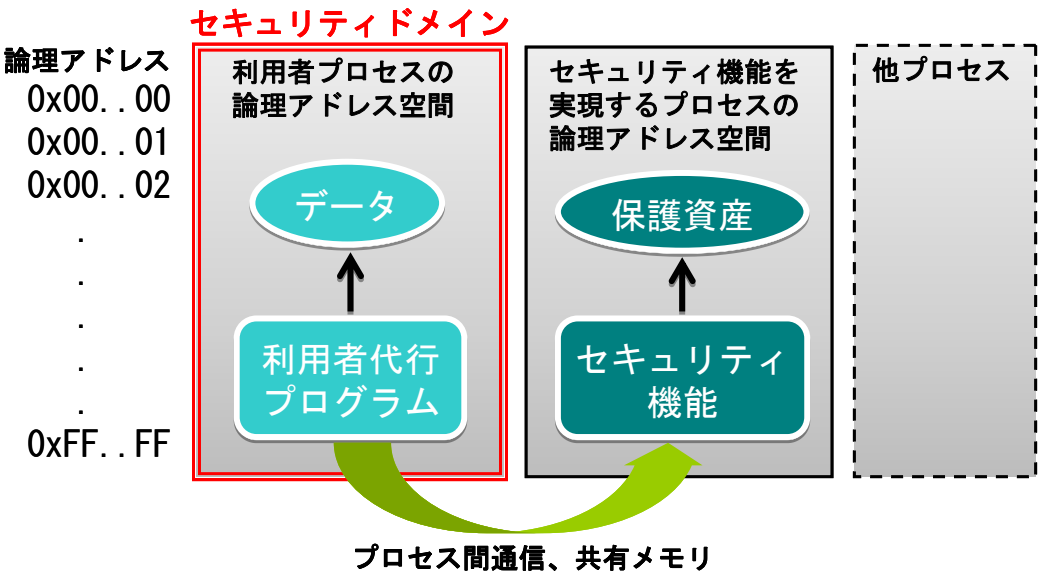


図 3-2 ドメイン分離の例（論理アドレス空間による場合）

この場合、OS では、プロセッサ等の備えるメモリ管理のしくみを利用して、プロセスごとに異なる論理アドレス空間を割り当てて、プロセスを実行するように制御します。それにより、各プロセスが参照可能な領域は自プロセスに割り当てられた論理アドレス空間に限定され、論理アドレス空間の異なる他プロセス内のプログラムやデータは参照できなくなります。

なお、プロセス間でデータのやり取りを行う場合には、プロセス間通信や共有メモリなどの特別なしくみを使用する必要があります。

■ ソフトウェア実行環境の制限によるドメイン分離メカニズム

上記の例では多くをハードウェアに依存していますが、主としてソフトウェアで実現されるドメイン分離メカニズムも考えられます。たとえば、Java プログラムコードを解釈しながら実行する JavaVM のようなソフトウェア実行メカニズムにおいて、利用者代行プログラムに対して自由にアクセスできる資源（プログラムやデータなど）を制限した実行環境を提供する場合が該当します。一方、セキュリティ機能は、利用者代行プログラムとは別のプログラムとして実装し、別の実行環境を割り当てます。それによって、ドメイン分離されることになります。

以下の実装形態の場合には、セキュリティ機能の配置されているメモリ領域は、保

護されておらず、利用者代行プログラムから干渉される恐れがあります。この場合、利用者代行プログラムは、ドメイン分離されていないことになります。

■ 同じ論理アドレス空間で同じプロセッサ動作モードでの実装

セキュリティ機能と利用者代行プログラムが、同じプロセス内で実装されている場合が該当します。これには、両者が明確に分離されていない実装である場合だけでなく、プロセス内で論理アドレス空間を共有して動作するスレッドや共有ライブラリの実装形態の場合も含まれます。

3.2.2 セキュリティドメインの記述

前節の分析の結果、ドメイン分離されている場合と、ドメイン分離されていない場合があります。両方の場合について、セキュリティアーキテクチャ記述に記載する内容を、以下に説明します。

(1) ドメイン分離が存在する場合

この場合は、セキュリティドメインの定義と、ドメイン分離のメカニズムを記述することになります。ドメイン分離のメカニズムには、すべてを製品自身が実現している場合と、製品が製品外部の機能を利用して実現している場合の両方が含まれます。

一例として、利用者代行プログラムはプロセスとして実現され、セキュリティ機能もまたプロセスとして実現されている場合について説明します。この場合、セキュリティドメインの定義としては、プロセスごとに割り当てられたアドレス空間を記述します。また、ドメイン分離のメカニズムを説明するために、セキュリティ機能もプロセスとして実現されていることを記述します。ドメイン分離のメカニズムは次のように説明できます。プロセスには、プロセスごとに論理アドレス空間が割り当てられます。それにより、各プロセスの参照可能な範囲は自プロセスの論理アドレス空間に限定されるため、他のプロセスの論理アドレス空間を参照することはできません。

(2) ドメイン分離が存在しない場合

まず、製品において本当にドメイン分離が必要ないかどうかを検討しなければなりません。利用者が入力した複雑なデータを処理する場合（たとえば、PDF 形式のデータの表示処理など)、それらのデータの解釈処理は、セキュリティ機能に干渉しないように制限されたセキュリティドメイン内で実行した方が、セキュリティはより高まります。もし、ドメイン分離が存在しない場合には、プログラムの実装ミス、誤動作、

欠陥を悪用した攻撃などによって、セキュリティ機能の動作に干渉する恐れがあります。それらの防止対策について考慮されていない場合、製品の実装アーキテクチャを再考する必要があります。

ドメイン分離が存在しないことが妥当である場合には、その根拠をセキュリティアーキテクチャ記述に記載することになります。根拠の例としては、次のような内容が考えられます。製品のインタフェースはメニューを選択する物理的なボタンのみであり、利用者による入力が厳しく制限されていて、予測できない動作の可能性はありません。そのためドメイン分離が存在しなくても、インタフェースの入力値を漏れなく検証するだけで、セキュリティ機能に対する悪影響を完全になくすことができます。

3.2.3 記述内容の確認

CC 評価では、セキュリティターゲットに記載されたセキュリティ機能要件に従って、セキュリティ機能が評価されます。そのため、セキュリティドメインの記述は、セキュリティ機能要件と一貫していることが求められます。以下の確認を行ってください。

(1) セキュリティ機能の観点

セキュリティ機能要件を満足するように実装されたセキュリティ機能のすべての部分が、ドメイン分離メカニズムで考慮されているかどうかを確認します。たとえば、セキュリティ機能が、OS のカーネル内のドライバと、OS 上で動作するプロセスの両方で実現されている場合があります。その場合には、利用者代行プログラムは、セキュリティ機能に干渉しないように、ドライバとプロセスの両方の観点でドメイン分離されている必要があります。

(2) 利用者代行プログラムの観点

セキュリティ機能要件のサブジェクト、つまり、セキュリティ機能要件が適用されるべき利用者代行プログラムのすべての部分が、ドメイン分離メカニズムで考慮されているかどうかを確認します。たとえば、一般利用者と管理者といった利用者の違いや、製品のコンソール装置とネットワーク経由といった利用者のアクセス形態の違いによって、ドメイン分離を検討する対象のプログラム部分が異なる場合があります。

3.2.4 ドメイン分離における注意事項

ドメイン分離の目的は、セキュリティ機能がバイパスされたり改ざんされたりすることを防止することです。そのため先に紹介したように、ドメイン分離は一般にプロセッサの実行モードやメモリ管理、あるいはソフトウェア実行環境など、セキュリティ機能要件で示された機能以外のメカニズムで実現されます。

したがって、セキュリティ機能である識別認証機能（ID・パスワードなど）やアクセス制御機能によって不正な利用者はドメイン分離されるといった主張は、セキュリティアーキテクチャとして期待される内容ではありませんので、注意が必要です。

3.3 自己保護

セキュリティアーキテクチャ記述の自己保護の項目では、製品のセキュリティ機能が改ざんされないように防止するためのセキュリティ機能のメカニズムを記述することが求められます。自己保護メカニズムには、大きくは次の 2 つが含まれます。

・ドメイン分離による自己保護メカニズム
・ドメイン分離以外の自己保護メカニズム
（利用者入力など、ドメイン分離だけでは対処できない改ざんの対策）

なお、改ざん防止は、製品のセキュリティ機能だけでなく、製品外部の機能を利用して実現される場合もあります。製品外部の機能は製品自身の機能ではないため「自己保護」ではありませんが、その機能を利用している製品内のメカニズムは「自己保護」に該当します。

セキュリティアーキテクチャ記述では、セキュリティ機能自身が実現している自己保護メカニズムと、製品外部の機能が提供しているメカニズムの役割分担を明確にすることが求められます。その理由は、製品内部のセキュリティ機能と製品外部の機能は、CC 評価において扱いが異なるためです。製品内部のセキュリティ機能は、設計・テスト・脆弱性分析といった CC 評価による保証の対象です。一方、製品外部の機能は、CC 評価による保証の対象ではありません。

以下、自己保護について、セキュリティアーキテクチャ記述として記載する方法の例を紹介します。

3.3.1 ドメイン分離による自己保護メカニズム

本節は、ドメイン分離が存在する場合だけが該当します。ドメイン分離が存在しない場合は該当しません。

3.2 節で説明したドメイン分離について、さらに、評価対象の製品内で実施している部分と、製品外部の機能など評価対象の範囲外で実施している部分を区別します。そして、評価対象の製品内で実施している部分を、自己保護メカニズムとしてセキュリティアーキテクチャ記述に記載します。

たとえば、3.2 節で説明したプロセッサの実行モードやプロセスの論理アドレス空間

を利用したドメイン分離の場合には、以下のような内容を記述します。

■ ドメイン分離を製品内で実現している場合

この場合は、プロセッサの実行モードによるアクセス権限の違いやプロセスごとの論理アドレス空間を実現するメモリ管理など、セキュリティドメインの分離に寄与する具体的なメカニズムをすべて記述します。

■ ドメイン分離の実現を製品外部の機能に依存している場合

この場合は、製品外部の機能が提供するドメイン分離メカニズムについて、評価対象の製品による利用方法やタイミングなどを記述します。

たとえば、次のような記述が考えられます。評価対象の製品はOS上で動作するソフトウェア製品であり、そのセキュリティ機能は起動時に管理者権限の常駐プロセスとして実行されます。セキュリティ機能は、利用者からの入力を受け取ると、利用者を識別認証した後、製品外部の機能であるOSの提供するシステムコールを利用して、利用者ごとにプロセスを生成し、一般利用者の権限で利用者代行プログラムを実行します。

3.3.2 ドメイン分離以外の自己保護メカニズム

本節は、ドメイン分離が存在する場合と存在しない場合の、両方の場合が該当します。

利用者や利用者代行プログラム部分（以下「利用者側」といいます。）からの入力について、セキュリティ機能が改ざんされないように保護するメカニズムをセキュリティアーキテクチャ記述に記載します。そのための手順の例を以下に示します。

(1) 自己保護対象のインタフェースの特定

まず、利用者側が、自身のセキュリティドメインの範囲の外にある保護資産を利用することのできる製品のインタフェースを漏れなく抽出します。たとえば、Webなどの画面入力、ネットワークインタフェース、製品内部のプログラム間のインタフェースなどがあります。図3-3に分析対象のインタフェースを示します。

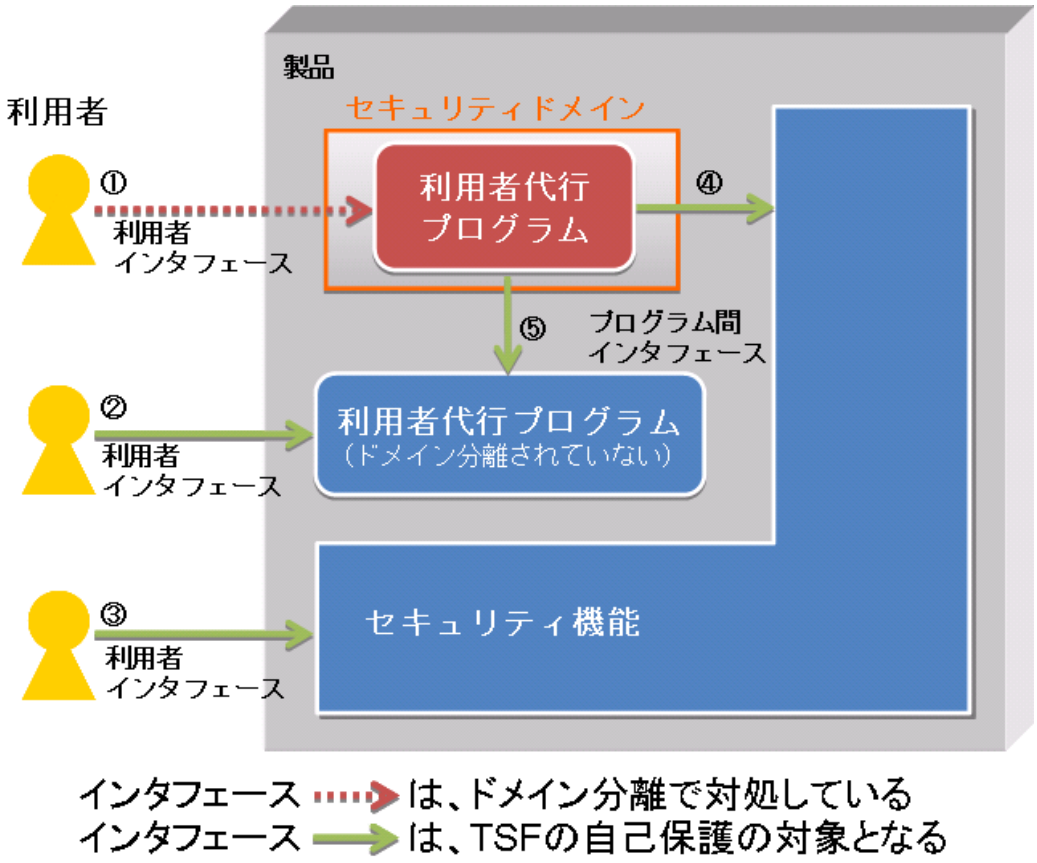


図 3-3 自己保護の対象インタフェース

図 3-3 において、インタフェースからの入力を保護しなければならないものは、②～⑤です。①のインタフェースは、それを処理する利用者代行プログラムがドメイン分離されていることでセキュリティ機能が保護されているため、①からの入力を保護する必要はありません。

セキュリティドメインの範囲の外部にある、ドメイン分離されていない利用者代行プログラムとセキュリティ機能は、互いに干渉しあう可能性があるため、それら全体を一体と考えて自己保護を検討する必要があります。

製品にドメイン分離が存在しない場合には、利用者と製品間のすべてのインタフェースが、自己保護の分析対象となります。

(2) インタフェースごとの自己保護メカニズムの記述

次に、特定したインタフェースごとに、セキュリティ機能が改ざんされないように保護するメカニズムを記述します。

インタフェースには、設計仕様で決められている範囲・大きさ・パターンなどから

外れた規定外の入力があったとしても、例外なく適切に処理されるメカニズムを備えている必要があります。たとえば、以下のようなものがあげられます。

■ バッファオーバフロー対策

バッファオーバフローは、文字列の入力処理などで、入力文字を読み込むために用意したバッファ領域以上に、長い文字列を読み込んだ場合に発生する問題です。バッファオーバフローが発生すると、プログラムが異常終了したり、外部から送り込まれた想定外のコードを実行されたりする恐れがあります。対策としては、入力処理において、バッファ領域に規定長以上の文字列を読み込まないような制限が必要です。セキュリティアーキテクチャ記述には、それらの内容を記載します。

なお、入力文字の長さをチェックする対策だけでは、チェックするために入力文字を読み込んだ時点でバッファオーバフローが発生する恐れがあるため注意が必要です。

■ スクリプトやコマンドなどの注入防止

外部から入力された文字列を処理する際に、文字列の中に含まれる意図しないスクリプトやコマンドが実行されてしまう恐れがあります。たとえば、SQL、OS コマンド、Java Script などの注入が知られています。対策としては、特殊文字の入力をチェックして排除したり、悪影響を受けないように別の文字に置き換えたりする処理などが必要です。また、誤りの発生し易い文字列の連結処理などを必要としない実装も考えられます（たとえば、SQL のバインド機構の利用など）。セキュリティアーキテクチャ記述には、それらの内容を記載します。

■ 利用者側から指定された不正なメモリ領域やファイル名の対策

製品のインタフェースを使用する際のパラメタとして、入出力データを格納するメモリ領域やファイル名を指定する場合があります。その際に、利用者側から本来アクセスできないはずの不正なメモリ領域やファイルなどが指定され、その領域などが製品内の入出力処理によって読み書きされてしまう恐れがあります。対策としては、利用者側から指定されたメモリ領域やファイル名などについては、アクセス権限などの妥当性チェックが必要です。セキュリティアーキテクチャ記述には、それらの内容を記載します。

■ 利用者側から指定されたメモリ領域の書き換え対策

製品が入力パラメタとして指定されたメモリ領域の内容を何度か参照して処理を行っている場合、その処理の間に利用者側がメモリ領域の内容を変更するか

もしれません。たとえば、製品が入力値の正当性を検証した後に、利用者側が入力値を不正な値に書き換えると、その後に行われる製品内の処理では書き換えられた後の不正な値が使用される恐れがあります。対策としては、インタフェースの処理が呼び出された時点で、パラメタとして指定されたメモリ領域の内容を、利用者側からはアクセスできない安全な領域にコピーしておき、以降はコピーした内容を参照して各種処理を実行することが考えられます。セキュリティアーキテクチャ記述には、それらの内容を記載します。

■ 不正に構成されたデータ形式やネットワークプロトコルの対策

データ形式やネットワークプロトコルが規定されていても、製品への入力が正当なデータであるとは限りません。たとえば、データ形式やネットワークプロトコルで可変長のデータを扱う場合、データの終了位置を示すために先頭からのオフセット値やデータ長を指定する場合があります。それらに不正な値が指定されるかもしれません。その場合に、製品が入力データを信頼して処理を実行すると、製品が領域外のデータをアクセスするなどし、予期しない動作をする恐れがあります。対策としては、データ内容の妥当性チェックが必要です。セキュリティアーキテクチャ記述には、それらの内容を記載します。

■ その他

上記以外でも、利用者側からのインタフェースの使用に伴うセキュリティ機能への悪影響を防止するためのメカニズムがあれば、その内容を記述します。たとえば、共有メモリを利用したインタフェースの競合対策や、スレッド間で共有されるプロセス内データの競合対策などがあります。セキュリティアーキテクチャ記述には、それらの内容を記載します。

3.3.3 記述内容の確認

セキュリティアーキテクチャ記述に記載したメカニズムと、実際の製品開発で使用された仕様書を比較し、内容が一致しているか、また、相互に記述漏れがないかどうかを確認します。

セキュリティ機能の改変や停止、想定外のコマンド実行の防止など、セキュリティ機能の改ざん防止に寄与するメカニズムは、すべて記述されるべきです。

3.4 非バイパス性

セキュリティアーキテクチャの非バイパス性の項目では、利用者が保護資産にアクセスする際に、保護資産を保護するためのセキュリティ機能が必ず適用されることを保証するメカニズムの記述が求められます。

そのためには、製品のすべてのインタフェースは以下のどちらかに該当し、セキュリティ機能をバイパスされるインタフェースが存在しないことを論証することが必要です。

- 保護資産にアクセスできるインタフェースでは必ずセキュリティ機能が適用されます。
- 上記以外のインタフェースでは保護資産にアクセスできません。

セキュリティアーキテクチャ記述では、保護資産にアクセスする際にセキュリティ機能が適用されるインタフェース（以下「セキュリティ機能のインタフェース」といいます。）と、それ以外のインタフェース（以下「セキュリティ機能には関係がないインタフェース」といいます。）について、それぞれ次のような内容を記載します。

- セキュリティ機能のインタフェース

  インタフェースを通じて保護資産などにアクセスする際に、保護資産を保護するために必要なすべてのセキュリティ機能が、適切なタイミングで必ず適用されるメカニズムを記述します。インタフェースの内部で、必要なセキュリティ機能をバイパスするようなモードや設定がないことも含まれます。

- セキュリティ機能には関係がないインタフェース

  上記以外のインタフェースが、なぜ、保護資産やセキュリティ機能に関係がないのかを記述します。単に「関係がない」と主張するだけでは不充分であり、具体的な処理メカニズムを示し、セキュリティ機能に関係しないことを明確に記述します。

以下、非バイパス性について、セキュリティアーキテクチャ記述として記載する方法の例を紹介します。

3.4.1 セキュリティ機能のインタフェースの非バイパス性

(1) セキュリティ機能のインタフェースの特定

まず、利用者や利用者代行プログラムが、保護資産にアクセスするために使用可能なインタフェースを特定します。そのインタフェースには、保護資産を保護するための、識別認証やアクセス制御といったセキュリティ機能が実装されているはずです。もし、セキュリティ機能が介在せずに、保護資産にアクセスするインタフェースが存在する場合には、インタフェースやセキュリティ機能が妥当かどうか、再検討が必要です。

なお、例外的に、製品外部の機能や設置場所の入退出制限などによって安全性が確保され、製品によるセキュリティ機能を必要としないインタフェースが存在する場合があります。その場合には、3.4.2 節の「セキュリティ機能には関係がないインタフェース」で扱います。

(2) セキュリティ機能が必ず適用されるメカニズムの記述

次に、そのインタフェースにおいて、セキュリティ機能が必ず適用されるメカニズムを記述します。以下に記述内容の例を示します。

■ 単純なセキュリティ機能の場合

インタフェースとセキュリティ機能の関係が単純な場合には、セキュリティアーキテクチャ記述もそれに見合った処理内容を記述すれば済みます。たとえば、利用者入力に対して、入力パラメタ処理、セキュリティ機能、目的の保護資産アクセス、処理結果の応答の順に逐次的に処理が行われ、利用者の入力パラメタによってセキュリティ機能を迂回するような条件分岐が存在しないことや、利用者が処理の順序を変更できないように設計・実装されていることを明確にします。

もし、インタフェースの内部で、設定値などを参照してセキュリティ機能をオフにするような処理が存在する場合には、運用中は常にセキュリティ機能がオンとなっていることを保証するメカニズムや運用対策が必要です。たとえば、オンオフの設定値は管理者だけがアクセス可能であり一般利用者はアクセスできないこと、さらに、管理者向けのガイダンスでセキュリティ機能がオンの設定で運用を行うような注意喚起が記載されていることを記述します。

■ 保護資産にアクセス可能な経路が多数存在する場合

保護資産にアクセス可能な経路が多数存在する場合、すべての経路についてセキュリティ機能が漏れなく適用されることを保証するために、特別なメカニズム

が必要となることがあります。

例として、HDD へデータを保存する場合に暗号化するセキュリティ要件を考えます。利用者は、HDD へデータを書き込むために、OS の様々な API やコマンドを利用することができます。それらのすべてに対してセキュリティ機能が必ず適用されるメカニズムが必要です。そのような場合、OS の提供するすべての HDD 入出力はデバイスドライバを介して行われることを利用して、デバイスドライバのレベルで HDD の入出力データに暗号化を適用する方式が一般に用いられます。それによって、利用者がどのような API やコマンドを使用しても、セキュリティ機能が必ず適用されることになります。セキュリティアーキテクチャ記述には、それらの内容を記載します。

■ 利用者による様々な操作が可能な場合

利用者がインタフェースを操作する順序や入力するパラメタを変更できる場合、利用者がどのような操作をしてもセキュリティ機能が漏れなく適用されることを保証するために、特別なメカニズムが必要となることがあります。

例として、Web システムで保護対象となるデータを参照する場合を考えます。正しい画面遷移では、最初にログイン画面が表示され、ログイン画面に入力した ID とパスワードによって識別認証が成功した場合のみ、目的のデータ参照画面にアクセスできます。ところが Web ブラウザでは、最初のログイン画面だけでなく、URL を直接指定することによって任意の画面にアクセスを試みることができます。その機能を使用して、ログイン無しで本来はアクセスできないデータ参照画面が表示される恐れがあります。どのような URL が指定されても、識別認証やアクセス制御などのセキュリティ機能が必ず適用されるメカニズムが必要です。対策としては、識別認証が成功した状態（セッション）を維持管理し、正当なセッションのアクセスだけを許可するような制御が必要です。セキュリティアーキテクチャ記述には、それらの内容を記載します。

なお、Web システムのセッション管理は、多くの攻撃者の興味の的であり、様々な攻撃が知られています。開発者は、セッション管理メカニズムに対する攻撃の対策にも配慮が必要です。

3.4.2 セキュリティ機能には関係がないインタフェースの非バイパス性

本節の対象となるインタフェースには、前節のセキュリティ機能のインタフェースには該当しない、その他すべてのインタフェースが含まれます。セキュリティ機能や保護資産と、当該インタフェースが関係ないことが理解できるように、インタフェー

スの処理メカニズムを記述します。以下に記述内容の例を示します。

■ ドメイン分離に依存している場合

当該インタフェースによって起動されるプログラム部分がドメイン分離されている場合には、当該インタフェースとセキュリティ機能や保護資産は関係がないことになります。この場合には、当該インタフェースによって起動されるプログラム部分が含まれているセキュリティドメインを特定して、ドメイン分離によってバイパスが防止されていることを記述します。

■ 特別なメカニズムが存在しない場合

当該インタフェースによって起動されるプログラム部分がドメイン分離されていない場合、そのプログラムは様々な経路でセキュリティ機能や保護資産にアクセスする恐れがあります。該当するプログラム内に、そのような処理が存在しないことを記述します。

たとえば、当該インタフェースによって起動されるプログラム部分の処理内容と影響を与えることができる範囲(アクセスするデータの範囲、他のプログラム部分への影響など)を記述し、セキュリティ機能や保護資産に到達するような機能が実装されていないことを説明します。

■ 製品以外の対策に依存している場合（例外的な場合）

保護資産にアクセスすることのできるインタフェースでありながら、保護資産を保護するためのセキュリティ機能が必要でない場合には、その理由を記述します。たとえば、製品外部の機能や運用対策に依存している場合、その依存内容を記述すると共に、その依存内容が確実に実施されるように管理者向けのガイダンスで注意喚起が記載されていることを記述します。

3.4.3 開発者の見落としがちなインタフェース

これまでは、開発者が開発し利用者に提供しているインタフェースについて、そのバイパス防止メカニズムを検討しました。しかし、開発者が意図していないインタフェースや、一般の利用者に公開していない非公開インタフェースも、セキュリティ機能のバイパスに悪用される恐れがあります。そのようなインタフェースを使用してセキュリティ機能がバイパスされないようにするために、開発者は何らかの対策が求められます。セキュリティアーキテクチャ記述には、その対策内容を記載します。以下に開発者の見落としがちなインタフェースの例を示します。

(1) 開発者が意図していないインタフェースの例

■ OS のインタフェース

OS は、物理メモリ空間、プロセスごとの論理アドレス空間、デバイスへの直接アクセスなど、様々なインタフェースを提供しています。それらのインタフェースから、保護資産にアクセスできる恐れがあります。対策としては、それらのインタフェースは管理者だけが使用でき、一般利用者はアクセスできないようにアクセス権限を設定するよう、ガイダンスで注意喚起することなどが考えられます。

■ Web サーバ

Web サーバは、Web ブラウザから Web サーバのディレクトリやファイル名を直接指定すると、その内容を表示する機能を備えています。その機能により、非公開情報が公開されてしまう恐れがあります。対策としては、Web サーバの設定やコンテンツの配置（公開用のディレクトリに非公開情報を置かない）などに注意する必要があります。

■ 暗号鍵の解読

保護資産の機密性を確保するために暗号化を行う場合、何らかの方法でその暗号鍵を知ることができれば、製品の提供している復号メカニズムを使用しなくても、暗号化された保護資産を復号できる恐れがあります。たとえば、暗号鍵を生成する際に、安全性の保証がないアルゴリズムの使用や、時刻情報や装置の個体番号などの安易な使用は、暗号鍵の解読や推測につながる恐れがあります。対策としては、暗号鍵が容易に解読されたり推測されたりしないように、暗号鍵の生成メカニズムや暗号鍵生成に使用するシード情報に、充分な配慮をすることが必要です。

■ 隠れチャネル

IC カードなどの場合、通常の入出力インタフェースではなく、電力波形を解析することによって、暗号鍵が解読される恐れがあります。また、パスワードの照合で１文字ずつ比較するような実装の場合、その照合に要する時間を測定するなどの方法で、何文字目で照合が失敗したかを知ることができれば、短時間でパスワードが解析される恐れがあります。対策としては、目的の演算と、観測される可能性がある消費電力や処理時間などとの相関関係がなくなるように、処理内容を工夫することなどが考えられます。

(2) 非公開インタフェースの例

■ 保守用インタフェース

製品には、保守などの目的で非公開のインタフェースが存在する場合があります。インタフェースの使用方法を秘密にしておくだけでは、十分な対策ではありません。たとえ非公開であっても、何らかの方法で攻撃者に見つけ出され、悪用されてしまう恐れがあります。対策としては、識別認証に加えて識別認証のメカニズムを補強する対策を併用するなど、安全対策に十分な配慮をする必要があります。

■ 開発時のデバッグ用インタフェース

開発時に利用されていたデバッグ用のインタフェースが製品に残ったままの場合、保守用インタフェースの場合と同様に、攻撃者に悪用される恐れがあります。対策としては、製品の中に不要なインタフェースが残存していないことを確認します。

3.4.4 記述内容の確認

自己保護の場合と同様、セキュリティアーキテクチャ記述に記載したメカニズムと、実際の製品開発で使用された仕様書を比較し、内容が一致しているかどうかを確認します。また、識別認証やアクセス制御だけでなく、監査ログ機能をはじめ、保護資産を保護するために必要なすべてのセキュリティ機能について非バイパス性が論証されていることを確認します。さらに、各セキュリティ機能について、利用可能なすべてのインタフェースを対象に非バイパス性が論証されていることも確認します。

3.5 セキュアな初期化

これまでに述べたセキュリティアーキテクチャ記述の自己保護やバイパス防止の項目では、製品の運用中においての状態を対象にしていました。それに対して、セキュリティアーキテクチャ記述のセキュアな初期化の項目では、電源オンなどにより製品を起動してから運用状態に至るまでの間の、セキュリティ機能が初期化される処理に焦点をあてて、そのセキュリティを確保するメカニズムを記述します。

図 3-4 にセキュアな初期化の概要を示します。

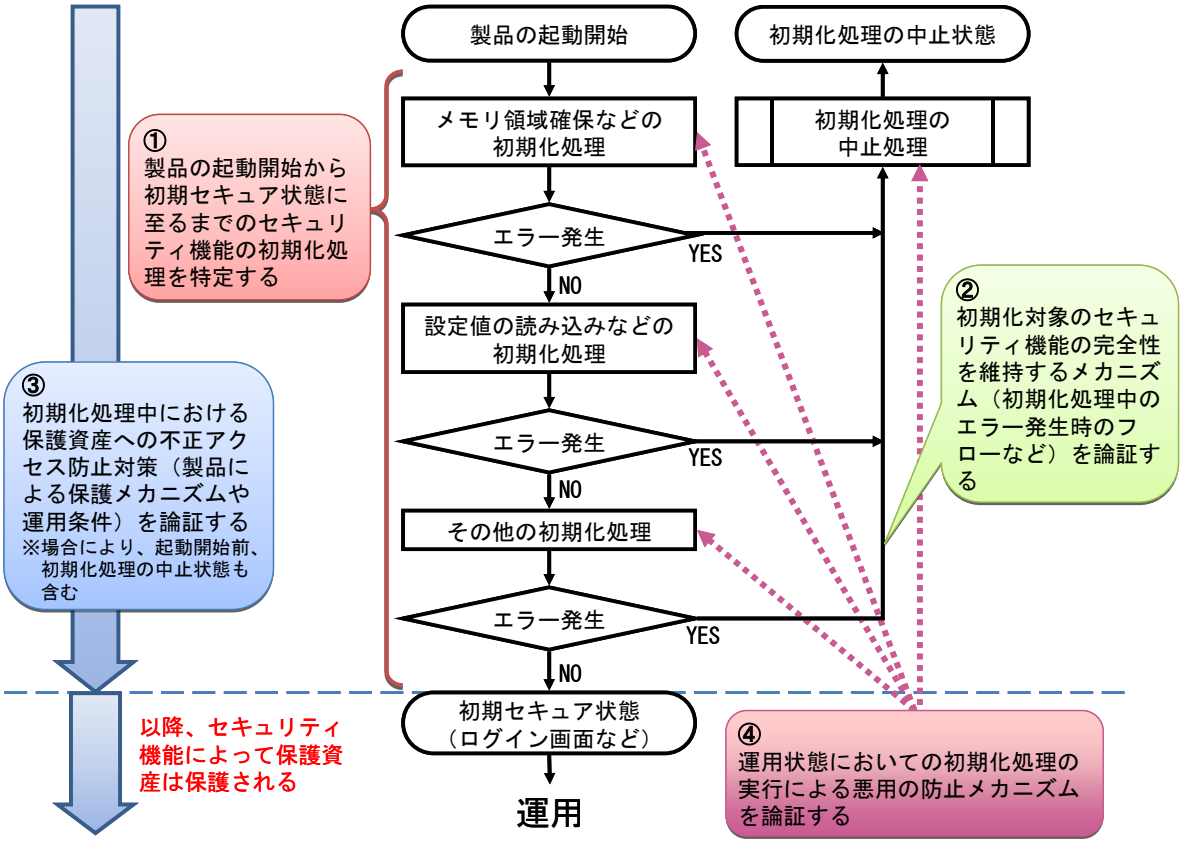


図 3-4 セキュアな初期化の概要

図 3-4 の①～④に示すように、セキュアな初期化には以下の内容が含まれます。

① セキュリティ機能の初期化処理の特定

製品内部において、セキュリティ機能の初期化処理を実施している部分を特定し、その処理の概要を記述します。ここで特定した初期化処理部分について、次項目以降の観点でセキュアであることを保証するメカニズムを記述していくことになります。

②初期化対象のセキュリティ機能の初期化処理の完全性確保

前項目で特定したセキュリティ機能の初期化処理において、初期化されるセキュリティ機能の完全性を確保するメカニズムを記述します。

③初期化処理中の保護資産の保護

セキュリティ機能がまだ動作していない初期化処理中において、保護資産への不正アクセスを防止するメカニズムを記述します。

④初期化処理の悪用防止

運用状態において、セキュリティ機能の初期化処理の実行による悪用を防止するメカニズムを記述します。

以下、セキュアな初期化について、セキュリティアーキテクチャ記述として記載する方法の例を紹介します。

3.5.1 セキュリティ機能の初期化処理の特定

初期化処理の対象は、製品を起動してから、製品のセキュリティ機能のすべての部分が運用可能な状態（以下「初期セキュア状態」といいます。）に移行するまでの間の、製品内の処理部分です。

以下のように初期化処理を特定し、セキュリティアーキテクチャ記述に記載します。

(1) 製品の起動方法の特定

まず、製品の起動方法を特定します。製品の起動は、ハードウェア機器を含む製品やハードウェア全体を制御するソフトウェア製品の場合には、電源オンやリセットなどが契機となります。OS 上で動作するソフトウェア製品の場合には、OS のスタートアップ時に自動起動されたり、OS の起動後に管理者の指示で起動されたりする場合があります。また、製品の再起動や、製品が運用の停止や再開の機能を備えている場合にはその再開指示も、製品の起動方法に含まれます。

(2) 初期セキュア状態の定義

製品によって初期セキュア状態に該当する状態は異なります。製品の特性に合わせて、製品起動後の初期セキュア状態（すなわち運用可能な状態）を定義します。

たとえば、コンピュータ機器の電源オンで自動起動されるソフトウェア製品が利用者にログインプロンプトを表示した状態が該当する場合もあれば、ファイアウォール製品においてはフィルタリング処理がネットワークパケットの入力待ちになった状態などが該当する場合もあります。

(3) 初期化処理の特定

製品を起動して初期セキュア状態に至るまでの間の処理で、セキュリティ機能の初期化処理に該当する処理を特定します。セキュリティ機能に直接関係する処理だけでなく、セキュリティアーキテクチャをはじめとして、セキュリティ機能をサポートするための製品内のすべてのメカニズムの初期化処理が対象となります。

たとえば、セキュリティ機能で使用する設定値の読み込みなどセキュリティ機能固有の処理だけでなく、ドメイン分離を行うためのメモリ管理のセットアップなども、その処理が製品の範囲に含まれている場合は、セキュリティアーキテクチャ記述に記載すべき初期化処理に含まれます。

また製品によっては、最初の起動時と再起動・再開時とでは、実行される処理が異なる場合があります。そのような点にも注意して、製品のダウン状態から初期セキュア状態に移行するまでの間のセキュリティ機能の初期化処理部分を漏れなく特定します。

(4) セキュリティアーキテクチャ記述の記載

セキュリティアーキテクチャ記述には、製品の起動方法と初期セキュア状態の定義を明記します。また、初期化処理の概要を製品の設計仕様との対応がわかる程度に記述します。

3.5.2 初期化対象のセキュリティ機能の完全性確保

製品を起動して初期セキュア状態に至るまでの間に、不正アクセスによる改ざんや何らかの原因による初期化処理の失敗によって、セキュリティ機能が正常に動作しない状態になっているにも関わらず、管理者がそのことに気付かずにセキュリティ機能が不完全のまま運用してしまえば大変危険です。そのような事態を防止してセキュアな運用状態を達成するためには、セキュリティ機能の初期化処理において、セキュリティ機能の完全性を確保するためのメカニズムが必要となります。

この観点について、セキュリティアーキテクチャ記述では、以下のような内容を記載します。

(1) 初期セキュア状態の達成

初期セキュア状態が確立されるためには、セキュリティ機能のための設定値の読み込みや、セキュリティ機能が使用するためのメモリ領域の確保など、様々な条件が達成されなければなりません。セキュリティアーキテクチャ記述では、初期セキュア状態が確立されるための条件が、前節で特定した初期化処理によって確実に達成されるメカニズムを記載します。

記載にあたっては、一連の初期化処理が正常に行われると初期セキュア状態に至る、のような説明だけでは不十分です。初期化処理が失敗する可能性のある要因を特定し、いかなる場合であっても安全性が損なわれることがないことを論証する必要があります。

多くの場合、初期化処理はシーケンシャルに実行され初期セキュア状態に至ります。ここで注意しなければならないのは、処理中にメモリ領域の確保の失敗などのエラーが発生した場合です。もし、エラーが発生しているにもかかわらず処理を続行させている場合には、セキュリティ機能が正しく動作しない状態で運用を開始してしまうかもしれません。したがって、この場合には関数の戻り値などエラー発生のチェックは確実に行い、エラーを検出した場合には初期化処理を中止するなどのメカニズムが必要となります。

(2) 初期化処理の中止状態の保護

エラー発生などで初期化処理を中止した状態を攻撃者に悪用される恐れもあります。たとえば、OS によっては管理者権限でコマンド入力待ちになったり、Windows のセーフモード起動のように特殊なモードで OS を起動することによりセキュリティ機能が十分でない状態で使用できたりする場合があります。そのような事態を防止するために、初期化処理の中止状態では利用者に可能な操作は電源オフに限定するなど、初期化処理の中止状態を悪用して保護資産に不正アクセスされないことを保証するメカニズムが必要となります。セキュリティアーキテクチャ記述には、それらの内容を記載します。

なお、製品の機能による対策が困難な場合には、3.5.3 節で説明する運用による対策と同様の対策が必要になります。

(3) 再起動時の注意

製品の再起動や再開時の扱いは、基本的には「(1)初期セキュア状態の達成」の場合と同じです。ただし、製品の再起動や再開時に特有の問題が発生する場合も考えられるので注意が必要です。

たとえば、管理者が再起動を行う際に最初の初期化処理がすべて成功しているとの前提の上で一部の初期化処理を省略するなどしてしまうと、実際には最初の初期化処理中にエラーが発生して処理が不完全だったという場合には、セキュリティ機能が不十分な状態で運用に至る恐れがあります。

別の例として、運用状態の際に管理者が行ったセキュリティに関する設定が再起動時にはリセットされるような製品の場合、管理者は再起動時に設定がリセットされたことに気付かないまま運用を開始する場合もあります。この場合の対策として、製品の再起動時や運用再開時などのセキュリティに関する設定が変更されるタイミングを管理者が認識できるように、警告メッセージを表示したり、ガイダンスで注意を喚起したりすることも考えられます。

3.5.3 初期化処理中の保護資産の保護

初期化処理の途中においては、セキュリティ機能がまだ運用状態になっていないため、保護資産が不正にアクセスされる恐れがあります。また、保守のための特別な利用モードや、USB メモリ等の媒体からプログラムを起動する機能が存在し、保護資産が不正にアクセスされる恐れもあります。そのような初期化処理中のバイパスを防止し、保護資産を保護するためのメカニズムが必要となります。

この観点について、セキュリティアーキテクチャ記述では、以下のような内容を記載します。

(1) 製品による対策

製品による対策がされている場合には、そのメカニズムをセキュリティアーキテクチャ記述に記載します。一般に、初期化の途中では、保護資産にアクセスするための機能が動作しないようにしておき、セキュリティ機能の動作準備ができて初めて、保護資産へのアクセス機能が動作するようにします。

たとえば、ファイアウォール製品の場合は、最初の状態では TCP/IP のパケット中継動作を禁止しておき、フィルタリングの動作準備ができた後に、TCP/IP のパケット中継動作を許可します。

セキュリティアーキテクチャ記述では、初期化処理で実施される処理内容の順序を具体的に記述し、初期化途中に保護資産へのアクセスができないことを論証します。

(2) 運用による対策

製品による対策がされていない場合には、製品の運用条件として、運用管理者が実施しなければならない内容をセキュリティアーキテクチャ記述に記載します。

たとえば、前述のようなファイアウォール製品を例にすると、運用管理者が必ず製品の起動に立ち会い、製品の起動前には LAN ケーブルを抜いておき、製品が運用状態になった後に LAN ケーブルを接続するといった運用対策も考えられます。なお、この例のように、運用管理者が初期化処理の間だけ立ち会う運用対策を想定している場合、たとえば機器のランプ表示やコンソールメッセージの確認など、運用状態に至ったことを判断する方法も明確になっている必要があります。

別の例として、サーバ機器等においては、運用管理者以外がコンソールの操作をできないようにするために、物理的に隔離する対策が採用される場合があります。

また、PC 上で動作する製品では、PC 内蔵の HDD からのみブートできるようにする

ために、初期設定時に、PC の BIOS 設定でブート可能なデバイスを制限したり、BIOS にパスワードを設定することで BIOS の設定を変更できないようにしたりする対策が採用される場合があります。

なお、これらの内容は、セキュリティアーキテクチャ記述に記載するだけでなく、製品のガイダンスに記載し、運用管理者が確実に実施するよう注意喚起することが必要です。

3.5.4 初期化処理の悪用防止

初期化処理は、セキュリティ機能を動作可能にするために、セキュリティ機能の動作に必要な特別なデータやハードウェアの設定を行っています。もし運用状態においてそれらの初期化処理が実行された場合、セキュリティ機能を改ざんされる恐れがあります。そのような事態を防ぐために、初期化処理は運用状態においては実行できないようにする必要があり、そのことを実現するメカニズムをセキュリティアーキテクチャ記述に記載します。たとえば、電源オンと同時に初期化処理はシーケンシャルに実行されるが運用状態に至ってからは初期化処理が実行できるインタフェースは提供していないこと、などを記述します。

また、初期化処理は OS 上で動作するプログラムで実現されていて、プログラムの再実行などによって、本来一度だけ実行されるはずの初期化処理が複数回実行されることを防止するために、初期化処理を実施済であるか否かを管理するメカニズムを実装する場合もあります。セキュリティアーキテクチャ記述にはそのような内容を記載することになります。

もし、運用状態においても初期化処理が実行できるインタフェースを提供している場合には、攻撃者がそのインタフェースを使用してセキュリティ機能を改ざんすることができないことを保証するためのメカニズムを記載する必要があります。

3.5.5 記述内容の確認

セキュリティアーキテクチャ記述に記載したメカニズムと、実際の製品開発で使用された仕様書を比較し、初期化処理も他のセキュリティ機能と同様に漏れなく記述され、内容が一致していることを確認します。

3.6 セキュリティアーキテクチャ記述の詳細度

これまで説明したように、セキュリティアーキテクチャ記述には、セキュリティ機能のバイパス・改ざんを防止するために、製品が備えているメカニズムを記述するこ

とが求められます。その記述内容について、どの程度の詳細度で記述すべきかについて説明します。

CC 評価では、保護資産を守るための識別認証やアクセス制御などのセキュリティ機能が、正確に実装され攻撃に対抗できることを、開発者の設計資料に基づいて評価します。評価に必要な設計資料は、7 段階の評価保証レベル（EAL）によって決められており、EAL が高いほど詳細な情報が求められます。また、セキュリティアーキテクチャのメカニズムは、保護資産を守るためのセキュリティ機能と同じレベルの詳細情報が求められます。以下、EAL ごとの詳細度を以下に示します。

①EAL 1

EAL 1 では、製品の外部インタフェース仕様が評価されます。そのため、製品内部のメカニズムであるセキュリティアーキテクチャ記述は、CC 評価には必要ではありません。

②EAL 2、EAL 3

EAL 2、EAL 3 では、製品の外部インタフェース仕様に加えて、製品内部の概要レベルの設計情報が評価されます。特に、保護資産を守るためのセキュリティ機能を実現する部分については、異なる機能間でやり取りされる主要データやその処理概要を明らかにするような詳細度が求められます。

たとえば、外部インタフェース仕様に記述されている入出力パラメタ、設定パラメタ、識別認証後にアクセス制御のために製品内部で保持される利用者 ID などの情報、それらの処理概要が含まれます。

また、EAL3 では、それらに加えてセキュリティ機能内部で使用される主要なデータの情報が含まれます。セキュリティアーキテクチャ記述においても、それらと同等の詳細度で、主要データと処理概要を明らかにすることが求められます。

③EAL 4 以上

EAL 4 以上では、製品の外部インタフェース仕様や製品内部の概要レベルの設計情報に加えて、プログラム実装レベルの詳細なデータ構造や処理フローを記述した設計情報と、ソースコードが評価されます。つまり、開発者以外の第三者（評価者）がソースコードを解読できるだけの詳細な設計情報が求められることになります。

セキュリティアーキテクチャ記述においても、それらと同等の詳細度で、詳細なデータと処理フローを明らかにすることが求められます。

なお、コーディング規則もセキュリティ機能の保護の証明に寄与する場合があり、

セキュリティアーキテクチャ記述に記載すべき内容に含まれます。たとえば、確保された入出力バッファの範囲を超えないように、格納するデータ長のチェックを行うような規則などは、バッファオーバフローを防止するための自己保護対策に該当します。

4 おわりに

セキュリティアーキテクチャは、保護資産を守るためのセキュリティ機能に対して、そのセキュリティ機能自体を不正アクセスから守るためのしくみです。本書では、CC 評価で求められるセキュリティアーキテクチャ記述の内容とその記述方法を説明しました。

本書の内容には、製品のセキュリティに関する開発設計時に見落としがちな点に対する注意喚起も多く含まれています。製品の設計開発にあたっては、その製品の CC 評価を検討しているか否かに関わらず、本書を参考にすることによって、よりセキュアな製品の実現に役立つものと思われます。

なお、開発者は以下の注意が必要です。

- セキュリティアーキテクチャ記述に記載する内容は、製品が実際に採用している設計方針やメカニズムです。したがって、セキュリティアーキテクチャ記述と製品開発で実際に作成された設計書とは、必ず一貫している必要があります。
- 採用している OS や既存のライブラリ・フレームワークなどの実行環境において、セキュリティ機能の保護対策が実現されている場合もあります。たとえば、実行環境によるドメイン分離、バッファオーバフローや SQL インジェクションなどの問題対策につながるメカニズムの採用などが考えられます。開発者は、それらにも留意してセキュリティ機能のバイパス・改ざんを防止するメカニズムを検討し、依存関係を明確にした上でセキュリティアーキテクチャ記述に記載します。
- セキュリティアーキテクチャ記述の内容で、製品のメカニズムで実現している部分は、一般の機能と同じように、それが正しく動作することをテストしなければなりません。
- セキュリティアーキテクチャ記述の内容で、運用対策で実現している部分は、利用者が確実に実施するように、ガイダンスなどで注意喚起する必要があります。

最後に、セキュリティアーキテクチャを検討・実装する際に参考となる文献を紹介しますので、ぜひお役立てください。

・安全なウェブサイトの作り方
https://www.ipa.go.jp/security/vuln/websecurity.html
・IPA セキュア・プログラミング講座
https://www.ipa.go.jp/security/awareness/vendor/programming/index.html

開発者のためのセキュリティ解説書（セキュリティアーキテクチャ編）

2012 年 3 月 21 日　　初版発行

2018 年 4 月 18 日　　改訂版発行

著作・発行　　独立行政法人情報処理推進機構（IPA）

執　筆　者　　情報セキュリティ認証室