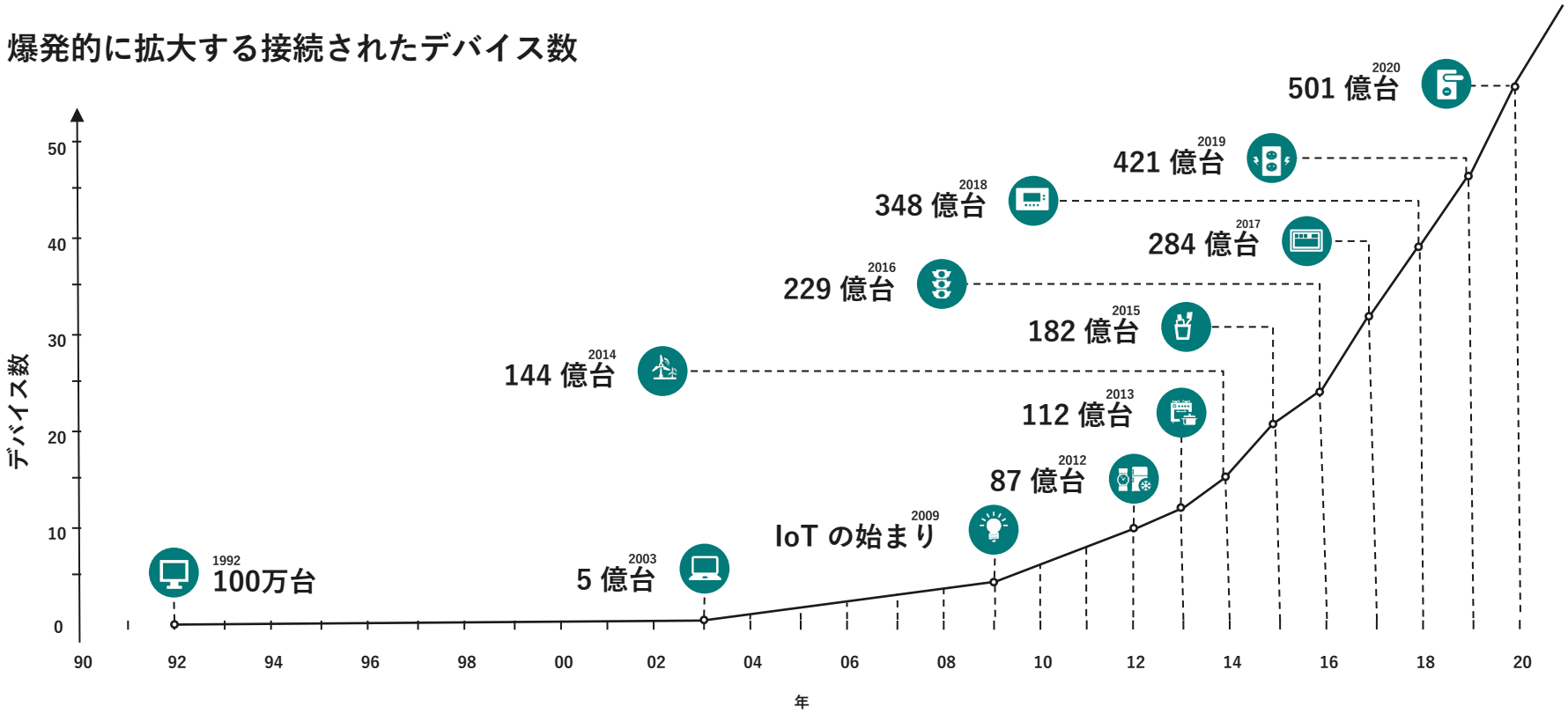
KASPERSKY OS の概要と できること できないこと

株式会社カスペルスキー
松岡正人

KASPERSKY lab

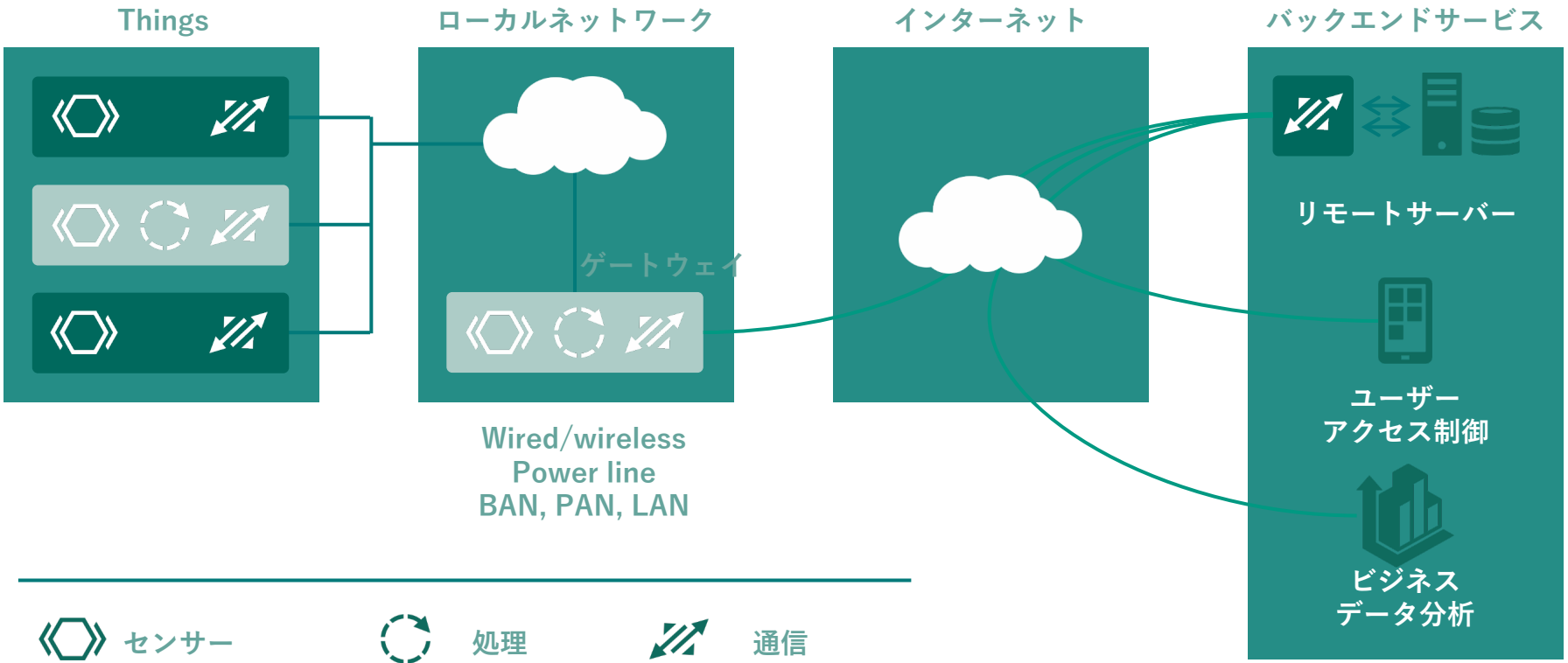
THE INTERNET OF THINGS

爆発的に拡大する接続されたデバイス数



KASPERSKY lab

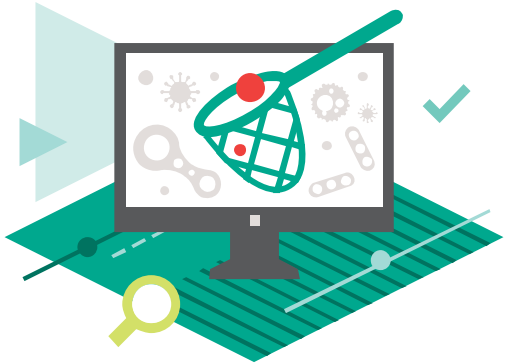
多機能な IOT 機器が最も危険



IOT に対する攻撃

MIRAI

Mirai は2016年8月に発見され、その名前はバイナリーの中の“mirai.()”からとられた。それ自身は ELF Linux 実行形式で、主にディスクレコーダー、ルーター、ネットワークカメラ、サーバー、その他の機器で、標準的な IoT 組み込み機器のツール Busyboxが動作しているものが対象である。



BASHLITE

分散DoS攻撃のために Linux システムに感染する。 2014年に BASHLITE は Shellshock のバグを衝いて BusyBoxが動作している機器を攻撃した。
2016 年、100万台の機器が BASHLITE に感染したと報告された。

サイバーセキュリティの観点からの IOT の主な問題

脆弱性

- ✓ 間違った操作
- ✓ サードパーティのアプリケーションやライブラリ
- ✓ ソフトウェアの複雑性
  （ソースコード量が多すぎる）

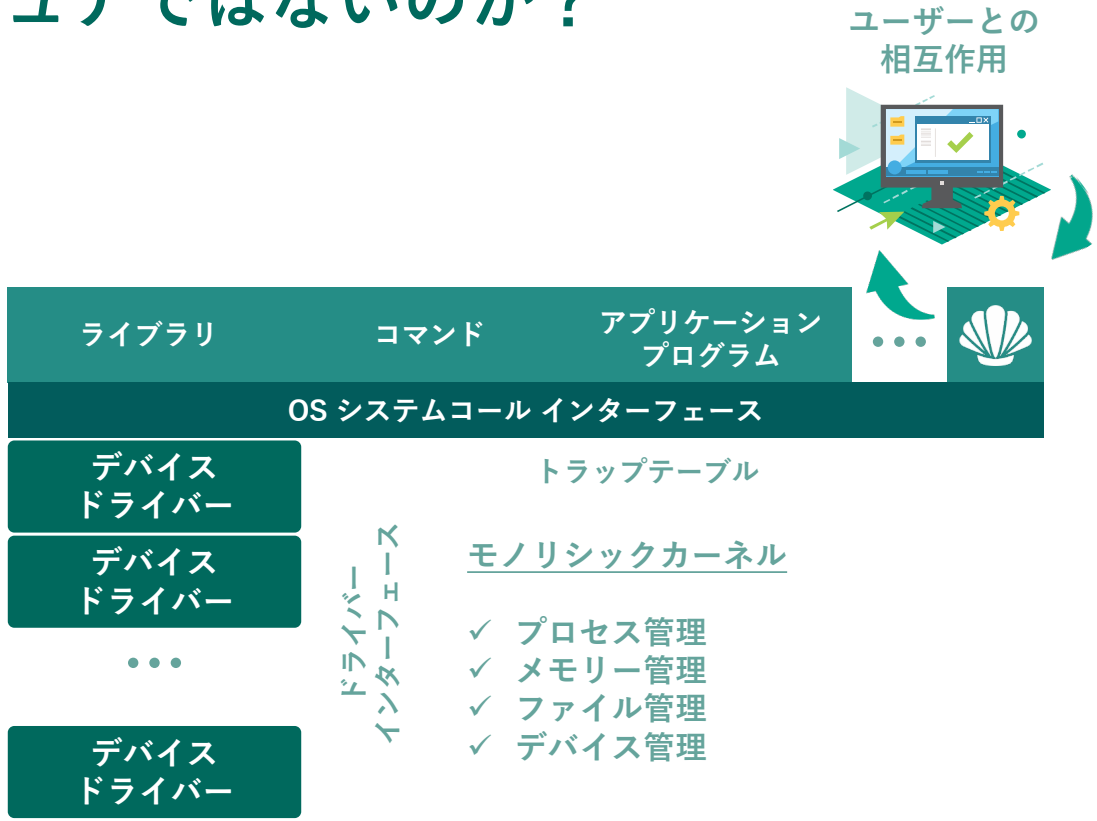
セキュリティを考慮していない設計

- ✓ 市場投入時期への圧力

従来のOSはセキュアではない

なぜ従来のOSはセキュアではないのか？

- モノリシックシステム：あらゆるモジュールを呼び出せる
- 任意のコードを実行できる脆弱性を悪用すればセキュリティの設定がなされていても悪意あるコードが実行可能
- ブラックボックスのサードパーティライブラリの使用によるリスク
- 攻撃者はひとつの脆弱性で対象システムのすべてを制御できる
- 安全性の低いセキュリティ設定（セキュリティに詳しくない、時間がないなど…）
- 広大な攻撃面を提供

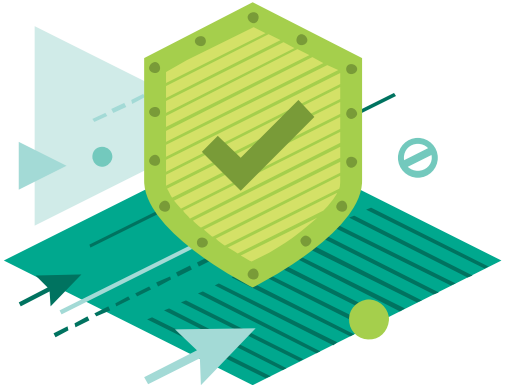


KASPERKSYの考える
安全な組込みシステム

KASPERSKY lab

KASPERSKYのアプローチ

脆弱性を悪用されることを防ぐために、
定義されていない機能を実行することの
できない環境を用意する



セキュアな組み込みOSの基本原則

- ✓ セキュア バイ デザイン
- ✓ MILS とリファレンスモニター
- ✓ マイクロカーネル
- ✓ 組み込みシステムの要件に沿う

理想的な組み込みシステム（OS）の要件

小さく、少ないリソースで動作

多くの組込みシステムは
ハードウェアリソースの制
約がある

攻撃下でも安定動作

事前に脅威や脅威ベクター
について十分に検討する

セキュリティは組み込み済み

多くの組込みシステムは
個々に異なるセキュリティ
要求があり、開発期間を短
縮しつつ必要なセキュリ
ティを構成する

業界標準準拠

業界の定めるセキュリティ
や安全基準や標準に準拠し
て設計、開発する

二つのアイデア

システムの下

マイクロカーネルアーキテクチャが可能にするドメイン分離と適用のための状態

＋

取り外されたセキュリティサブシステムは与えられたポリシーに基づいてセキュリティ判定を計算

ひとつめのアイデア

プロセス間通信(IPC)
の統合されたメカニズム

マイクロカーネルで実装

完全な調停を提供

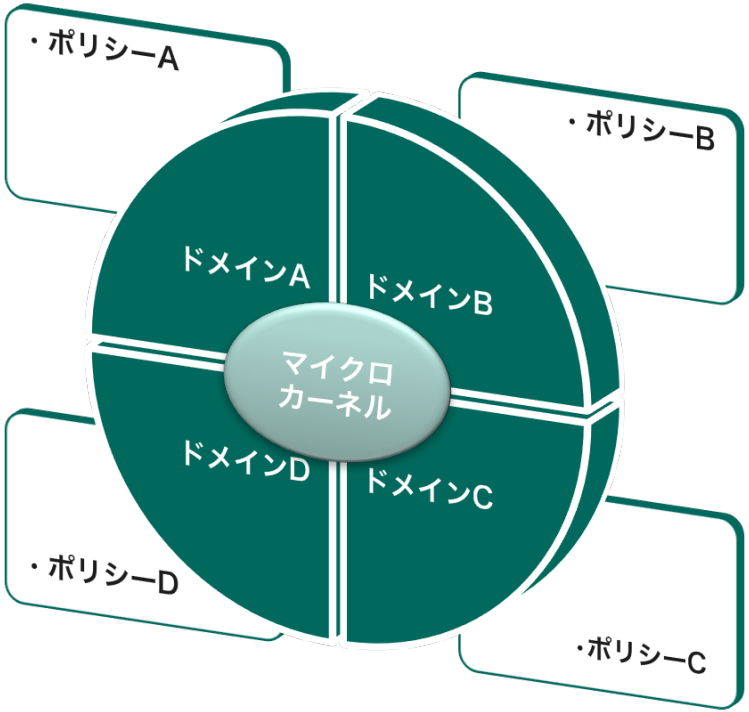
マイクロカーネル

IPCの実装

マイクロカーネルの提供するIPC以外にプロセス間で通信できない

非同期通信(ランデブースキーム)

セパレーションカーネル



セパレーションカーネルのタスクは物理的な分散システムによって提供されたものと見分けがつかない環境を作ること (John Rushby, 1981)

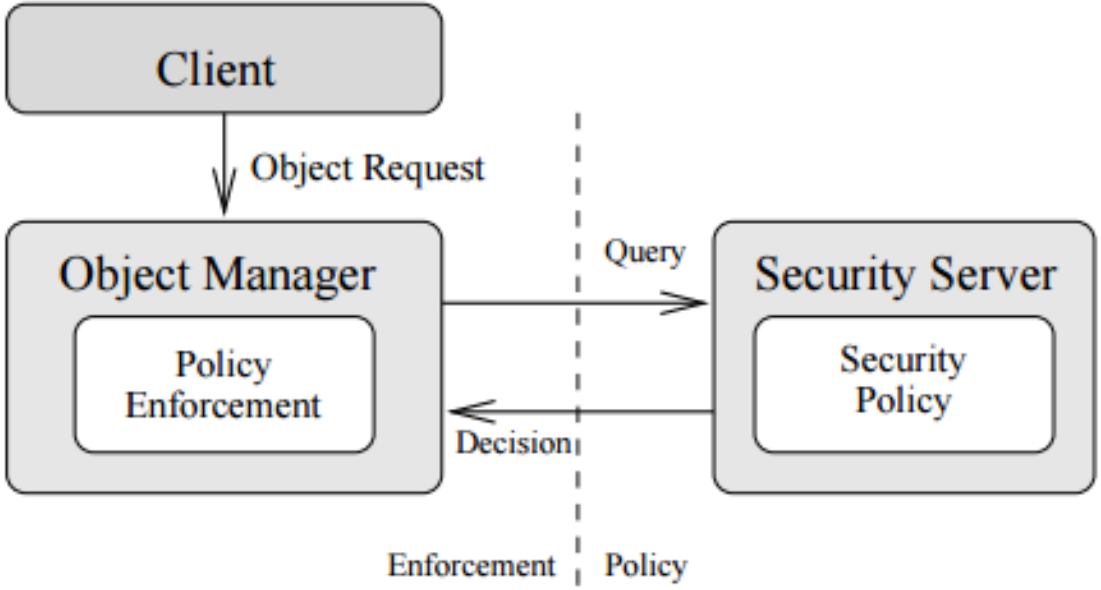
実装: マイクロカーネル / ハイパーバイザー

ふたつめのアイデア

分離したサブシステム

システムコンポーネント間の通信の制御が柔軟に

FLASK セキュリティアーキテクチャ



https://www.cs.cmu.edu/~dga/papers/flask-usenixsec99.pdf

KASPERSKY OS の概要

- Kaspersky 独自開発のマイクロカーネル OS
- 静的セキュリティの実装と提供
- MILS アーキテクチャ
  - ドメイン分離
  - Kaspersky Security SystemによるIPC制御
- アプリケーションとセキュリティを分離（開発とサポートを容易に、市場投入期間を短縮、セキュリティと安全性を向上）
- セキュリティドメイン粒度を最小にすることで制御範囲を最大化（全てのプロセス、ドライバーはセキュリティドメインに）
- POSIX API 互換（およそ 98% の APIを提供）
- 主要なCPUアーキテクチャをサポート： Intel x86とx64 および ARM（v6、v7 および v8）

注意
パッケージはありません！

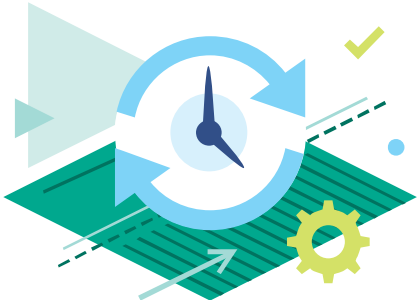
KASPERSKY OS の特徴

コードの実行管理

セキュリティポリシーに規定される振る舞い以外のコードは実行されません

柔軟なセキュリティポリシー

複数の異なるセキュリティポリシーを設定することで、アプリケーション毎に異なるポリシーを適用可能です



セキュリティは基盤

セキュリティはOSの中に実装され基盤として実装されるため、アプリケーションとは分離して管理されます

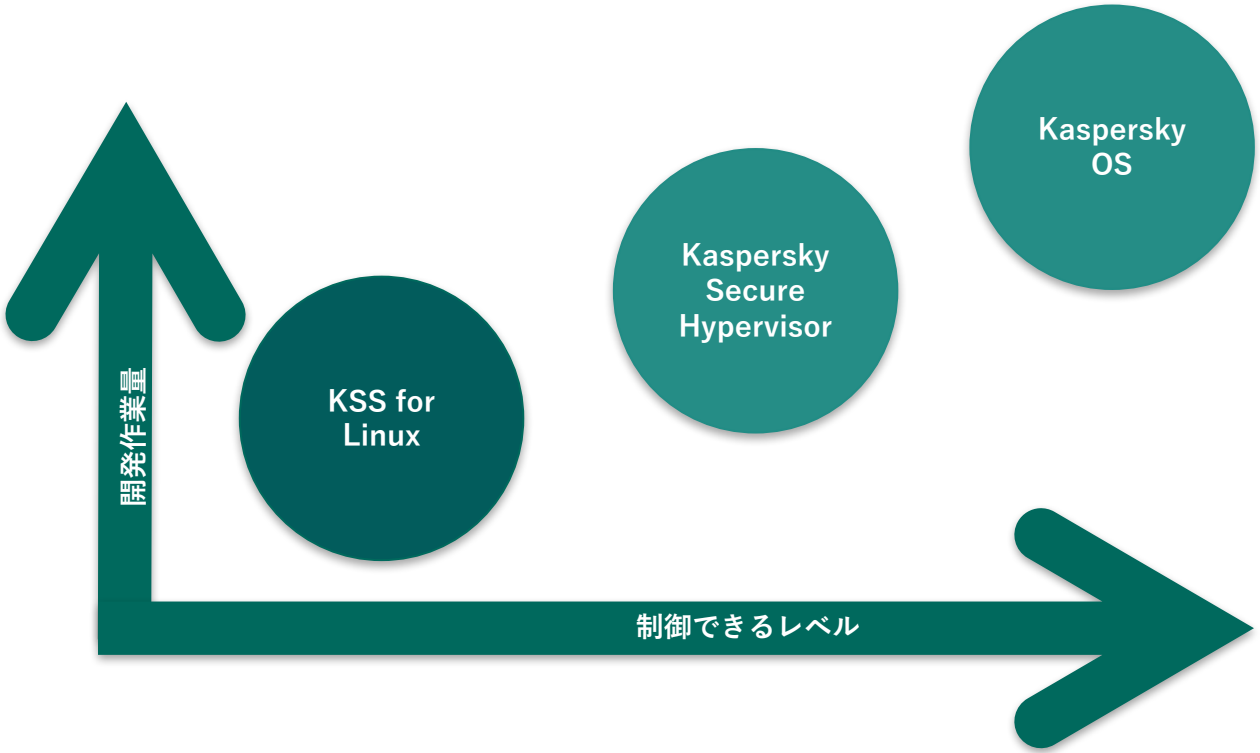
KASPERSKY OS の構成

複雑でユニークな組み込みシステムのセキュリティを向上するために、シンプルな仕組みで保護を提供できることを規範としています

そこで、基本的なセキュリティを提供するセキュリティサブシステムとハイパーバイザーを用意し、独自開発のカーネル上に載せています。

- ✓ Kaspersky OS （Hypervisor, KSS含む）
- ✓ Kaspersky Secure Hypervisor
- ✓ Kaspersky Security System for Linux

用途に応じて使い分ける



- Linuxコンテナによってプロセスを分離しコンテナ間の通信をサポート
- アプリケーションアーキテクチャを再考し開発し直す必要があります
- 最小限の開発でセキュリティを向上
- コンテナのサポートによってあらゆるLinux上で仮想化して動作

KOSとコンポーネントの概要

KASPERSKY OS

- すべてのコンポーネントが分離、制御される
- すべてのコンポーネントのアーキテクチャを最適化する必要がある
- アプリケーションやドライバーは移植するか書き直す必要がある
- サポート環境は限定的（組込みシステムのみ）

SECURE HYPERVISOR

- 仮装マシンと重要な機能の分離、通信の限定的な制御
- アプリケーションのアーキテクタの最適化が必要
- 書き直すコンポーネントのは依存するもののみに限定
- 組み込みシステム以外でも利用可能（仮想化環境であるため）

KSS FOR LINUX

- Linux コンテナの分離、コンテナ間の通信を制御
- アプリケーションのアーキテクタの最適化が必要
- 書き直しは最も少ない
- 組み込みシステム以外でも利用可能（仮想化環境であるため）

KASPERSKY OS の長所と短所

KASPERSKY

長所: 完全な調停

アーキテクチャレベル:

- ✓ プロセス間通信 (IPC)のみを通じて調停

実装レベル:

- ✓ 評価可能なオープンデザイン (コンポーネントモデル)

構成レベル:

- ✓ 基本拒否の規則（Default Deny）

長所: 柔軟性

アーキテクチャレベル:

✓ 分離可能なセキュリティサーバー

実装レベル:

✓ 通信の意味論的分類が可能

構成レベル:

✓ 複数の構成

長所: 検証可能なセキュリティ

アーキテクチャレベル：

- ✓ マイクロカーネルアーキテクチャ、明白に定義された TCB

実装レベル：

- ✓ アプリケーションのためのコンポーネントモデル

構成レベル：

- ✓ 単純な構成言語、コードジェネレーター

長所: MANDATORY ACCESS CONTROL(MAC)のサポート

アーキテクチャレベル:

- ✓ 分離可能なセキュリティサブシステム＋MACのサポート

実装レベル:

- ✓ 透過的なセキュリティポリシーの適用

構成レベル:

- ✓ TCBによって提供される硬直的な構成

長所: 遵守すべきラベリングと構成

アーキテクチャレベル:

- ✓ GATEはアプリケーションに必要な構成を強制的に適用

実装レベル:

- ✓ 構成されていないアプリケーションはシステムにインストールできない

構成レベル:

- ✓ すべてのハードウェアとソフトウェアのリソースはセキュリティ属性で区別される

IPCの短所

- ✓ POSIX との完全互換性は無い
- ✓ リソースへのアクセスの問題
- ✓ 性能の問題

分離したセキュリティサブシステムの短所

- ✓ (セキュリティモデルが失敗した際) 適切なセキュリティポリシーの選択の問題
- ✓ がちがちの構成
- ✓ セキュリティ判定キャッシュの問題

疎結合アーキテクチャの短所

MACサポートと規制ポリシー強制適用による二つの独立したシステムにおいて

- ✓ アプリケーションのランタイム環境
- ✓ 不透過なコンテクストを含むセキュリティシステム

※気をつけて同期する必要がある

KASPERSKY OS アーキテクチャ

ドメイン分離に基づいたアーキテクチャ

- 統合されたIPCがセパレーションカーネルを実装
- セキュリティとセーフティの懸念の両方に対応できる

分離したセキュリティシステム

- 多様なセキュリティポリシーの実装
- 再利用可能で構成可能なポリシー

高度なセキュリティの保証

- マイクロカーネルとコンポーネントアプリケーションモデルによって検証可能
- 構成はシステムTCBに格納

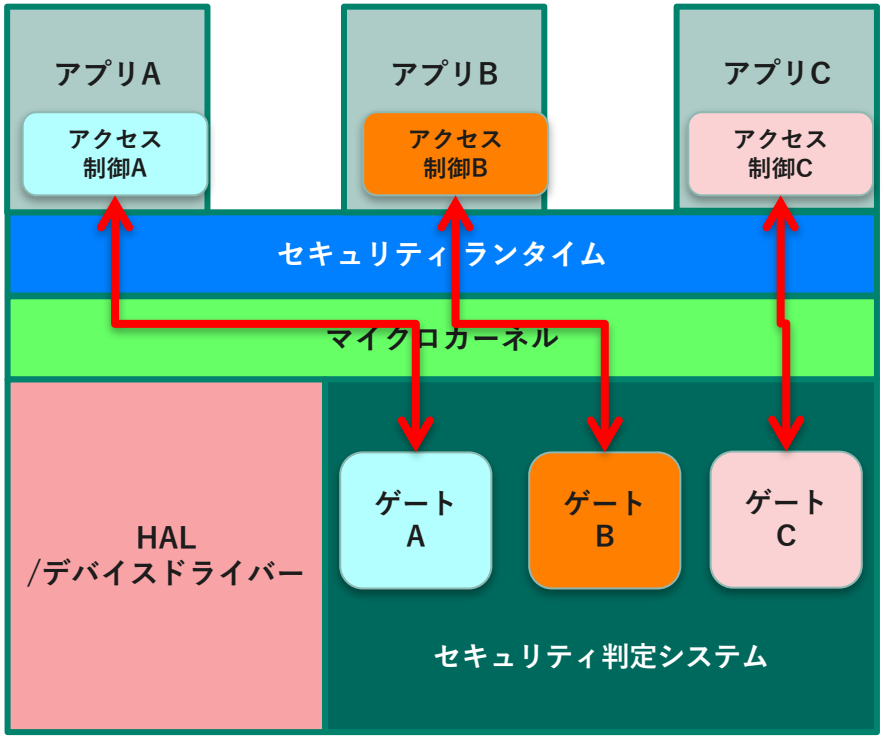
KSS: KASPERSKY OS のアクセス制御システム

- ✓ 汎用インターフェース
- ✓ 保護メカニズムをセキュリティポリシーの定義と分離
- ✓ 多様なセキュリティポリシーときめ細かな制御
- ✓ 判定を強制するための専用のレイヤー
- ✓ 汎用の相互作用の定義

KSS アーキテクチャ

KSSでは動作を管理、制御するポリシーをアプリケーションやファイル単位で設定できます。たとえば、「マルチレベル セキュリティ（MLS）」や「Capabilityベースアクセス制御（CBAC）」、「ロールベースアクセス制御（RBAC）」などが実装されてます。これらのアクセス制御方法は一般の商用OSの「アクセス制御リスト（ACL）」方式に比べて新しく、認証に伴うオーバーヘッドを小さくすることが可能となる。

RBACはマイクロソフトのActive Directoryなどロールを定義することでアクセス制御する仕組みで用いられている。したがって、各々のアクセス制御方式に求められるポリシー管理の仕組みはSecurity Verdict computingモジュールとしてカーネルメモリー空間で動作するため、ユーザーアプリケーションからのアクセスは専用の通信層、セキュリティランタイムを通じてのみ可能となる。



KSS の概要

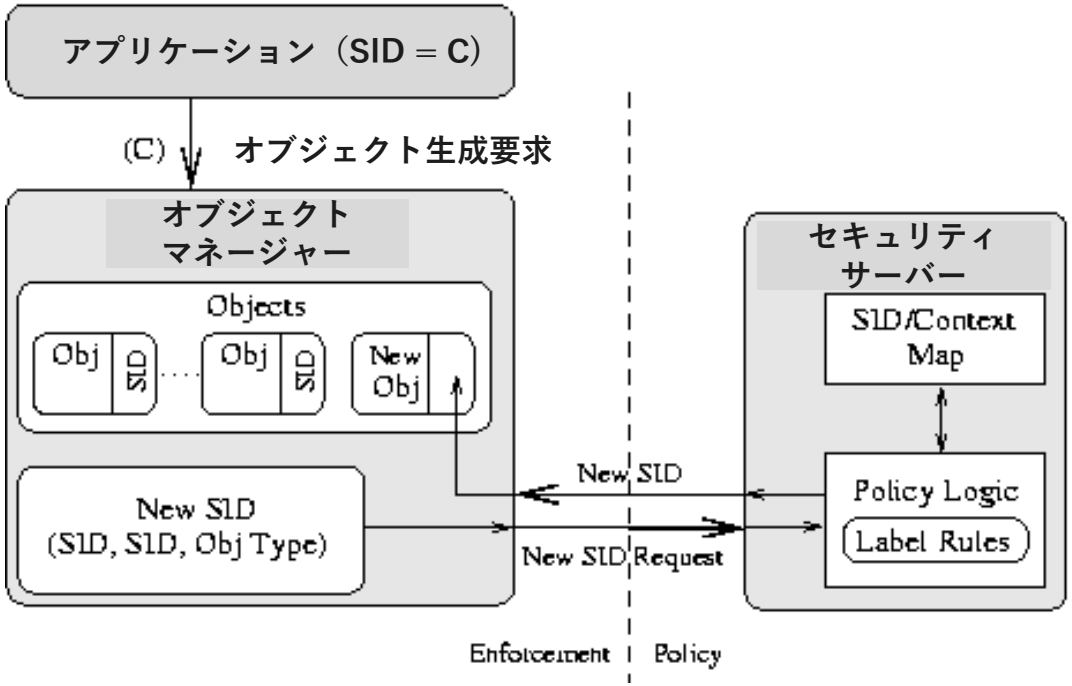
アプリケーション（Entity）

↓（SIDを渡す）

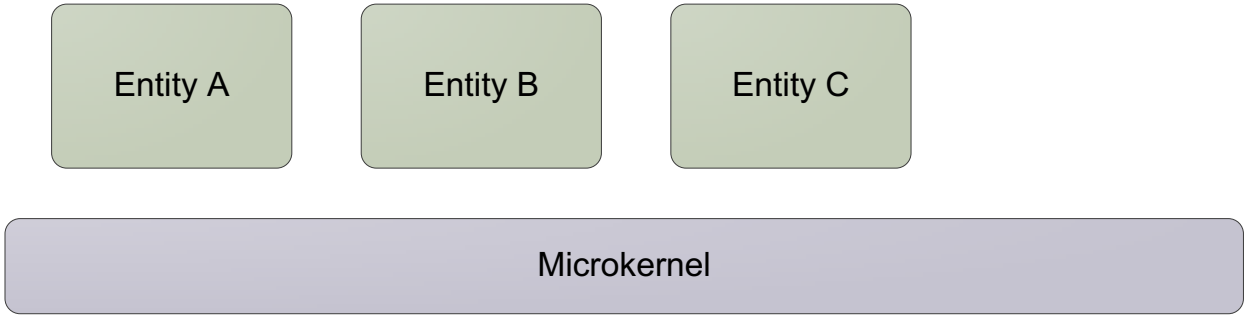
オブジェクトマネージャー

↓

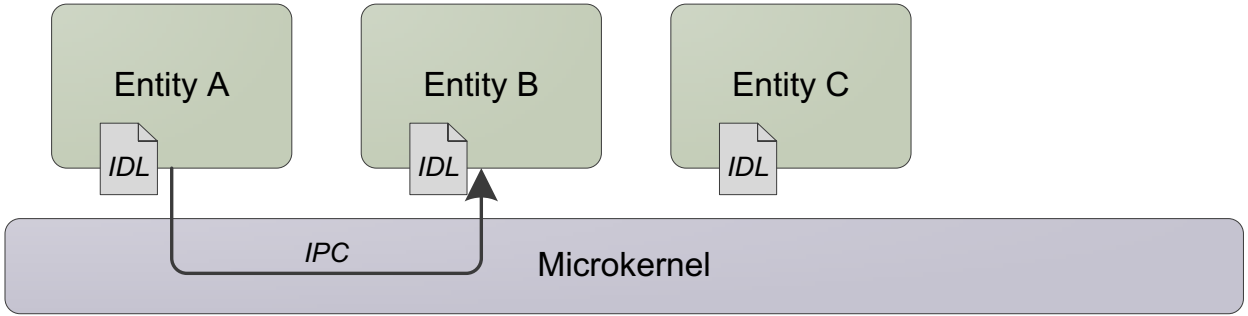
セキュリティサーバー



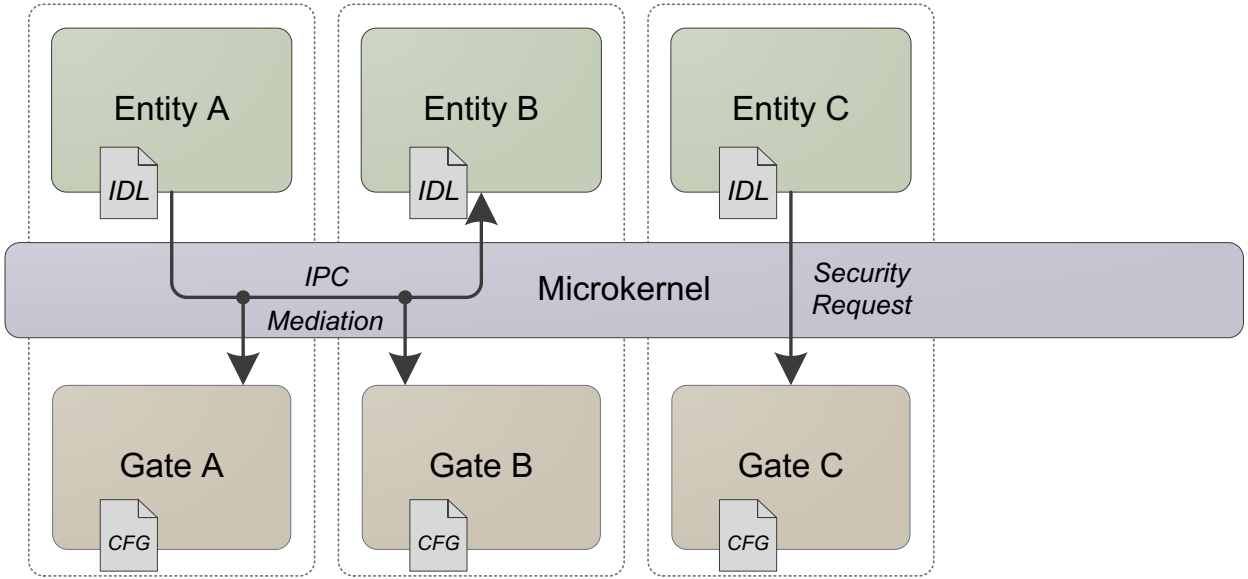
KSS の概要



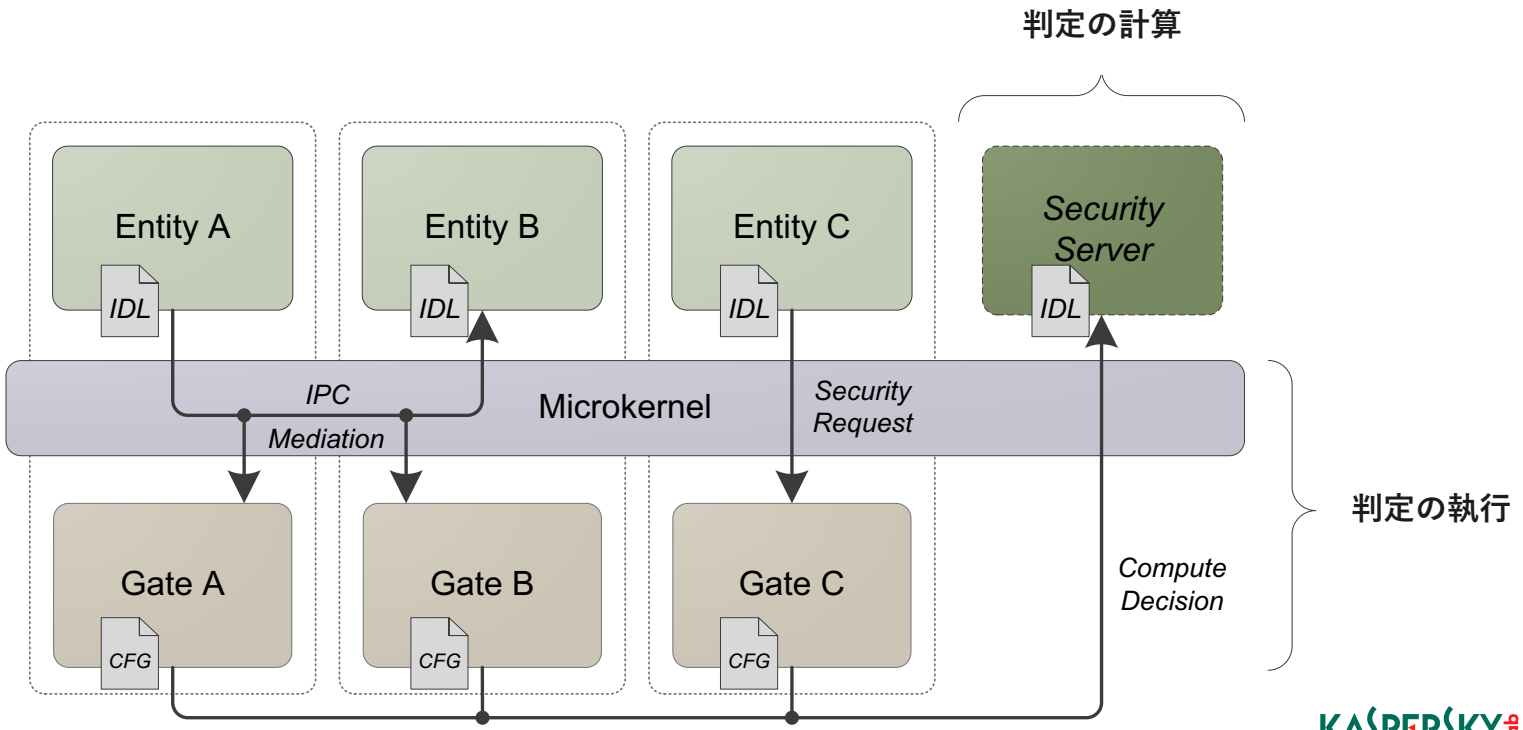
KSS の概要



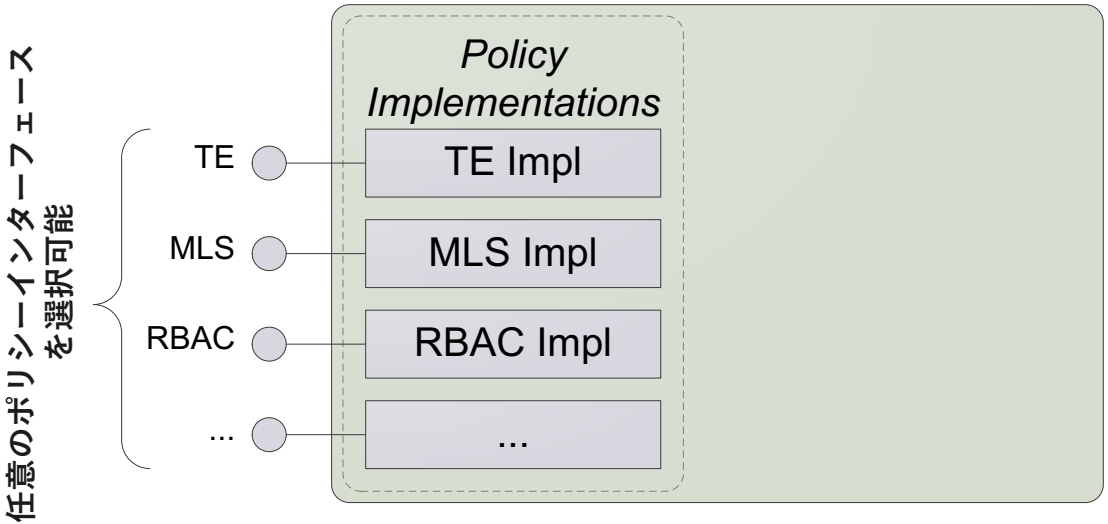
KSS の概要



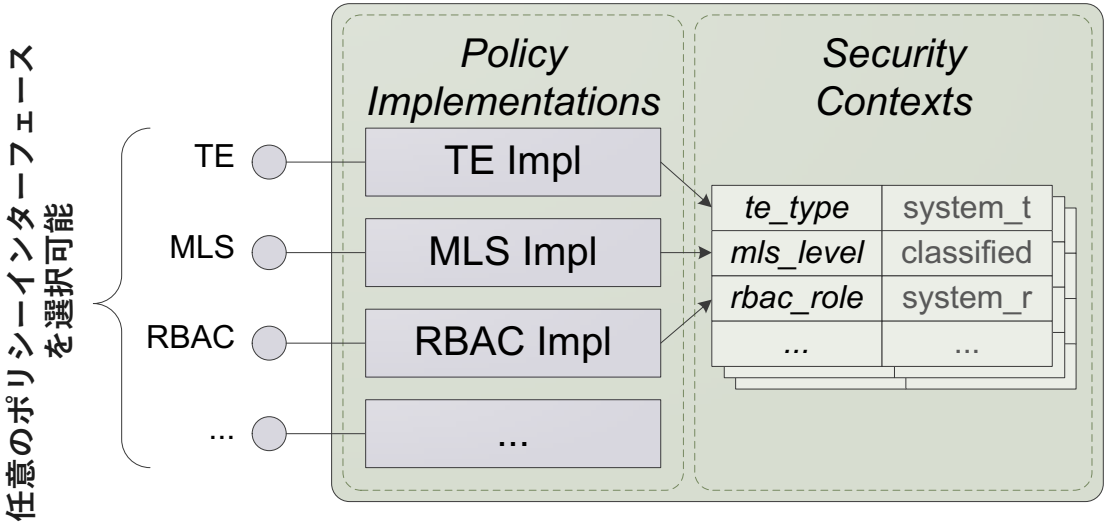
KSS の概要


判定の計算
Entity A
IDL
Entity B
IDL
Entity C
IDL
Security Server
IDL
IPC
Mediation
Microkernel
Security Request
判定の執行
Gate A
CFG
Gate B
CFG
Gate C
CFG
Compute Decision

SECURITY SERVER の概要



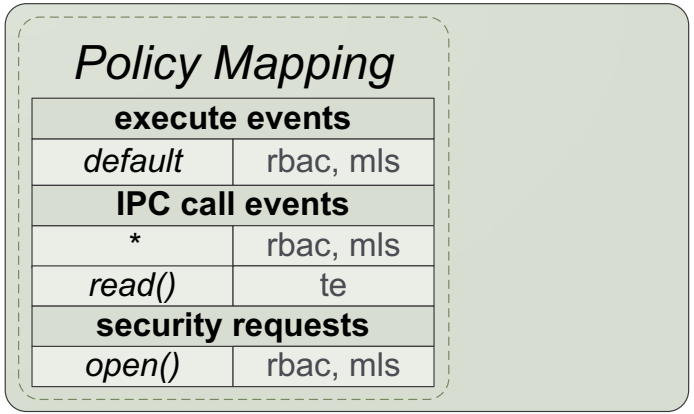
SECURITY SERVER の概要



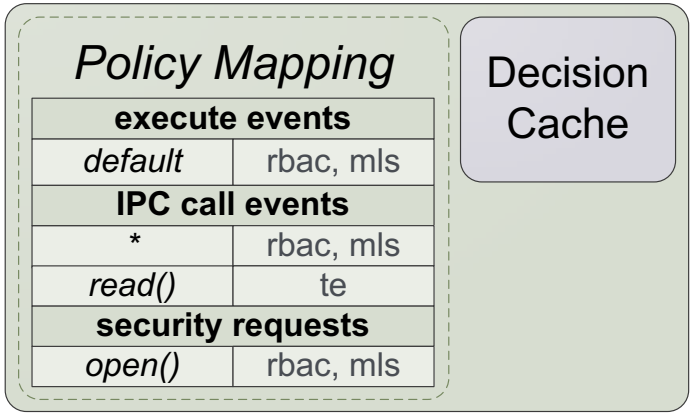
SECURITY SERVER の概要

- ✓ セキュリティ判定の通知
- ✓ ポリシーの実装と構成を管理
- ✓ セキュリティコンテクストの管理

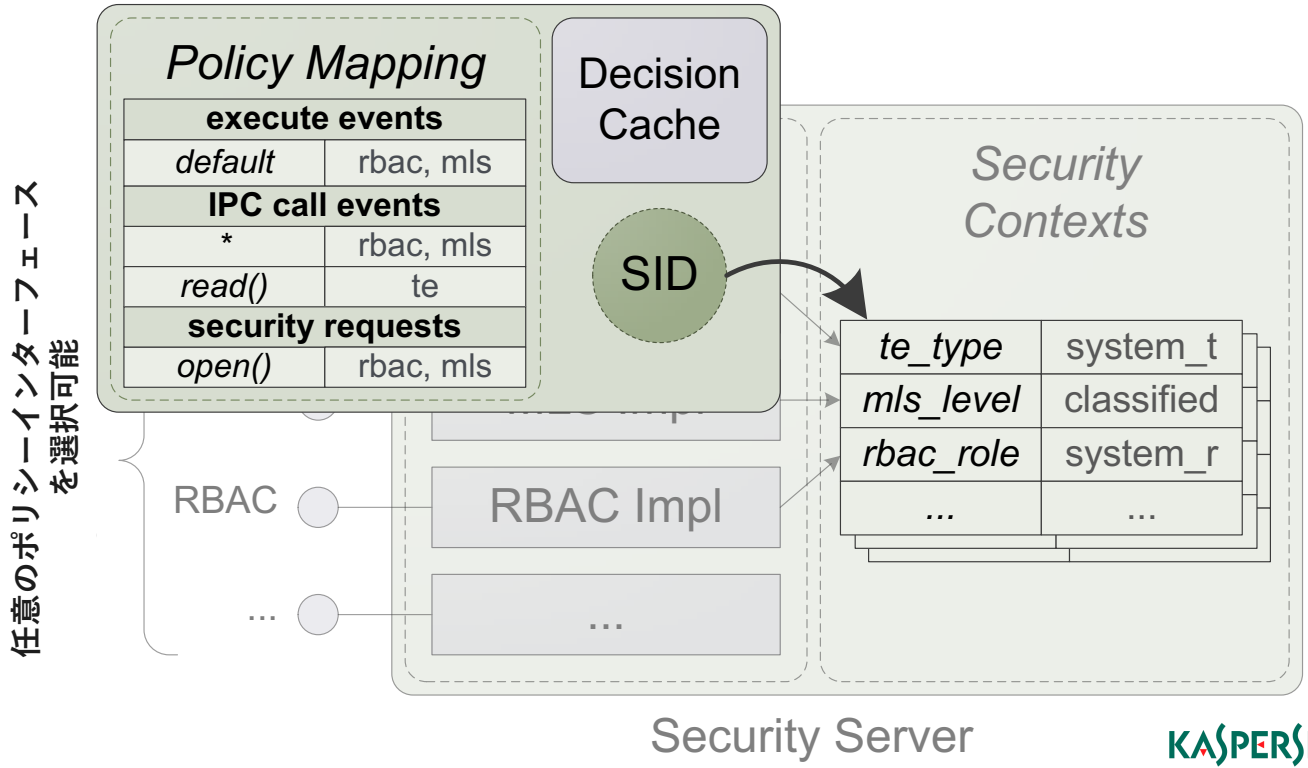
GATE の概要


Policy Mapping
execute events
default
rbac, mls
IPC call events
*
rbac, mls
read()
te
security requests
open()
rbac, mls

GATE の概要



GATE の概要


任意のポリシーインターフェースを選択可能
Policy Mapping
execute events
default
rbac, mls
IPC call events
*
rbac, mls
read()
te
security requests
open()
rbac, mls
Decision Cache
SID
Security Contexts
te_type
system_t
mls_level
classified
rbac_role
system_r
...
...
RBAC
RBAC Impl
...
...
Security Server

GATE の概要

- ✓ エンティティのイベントをセキュリティポリシーと紐付け
- ✓ セキュリティサーバーからの判定へのアクセスを要求
- ✓ SIDによってエンティティとセキュリティコンテクストを関連づけ
- ✓ 判定キャッシュへのアクセスの管理

ユースケース例

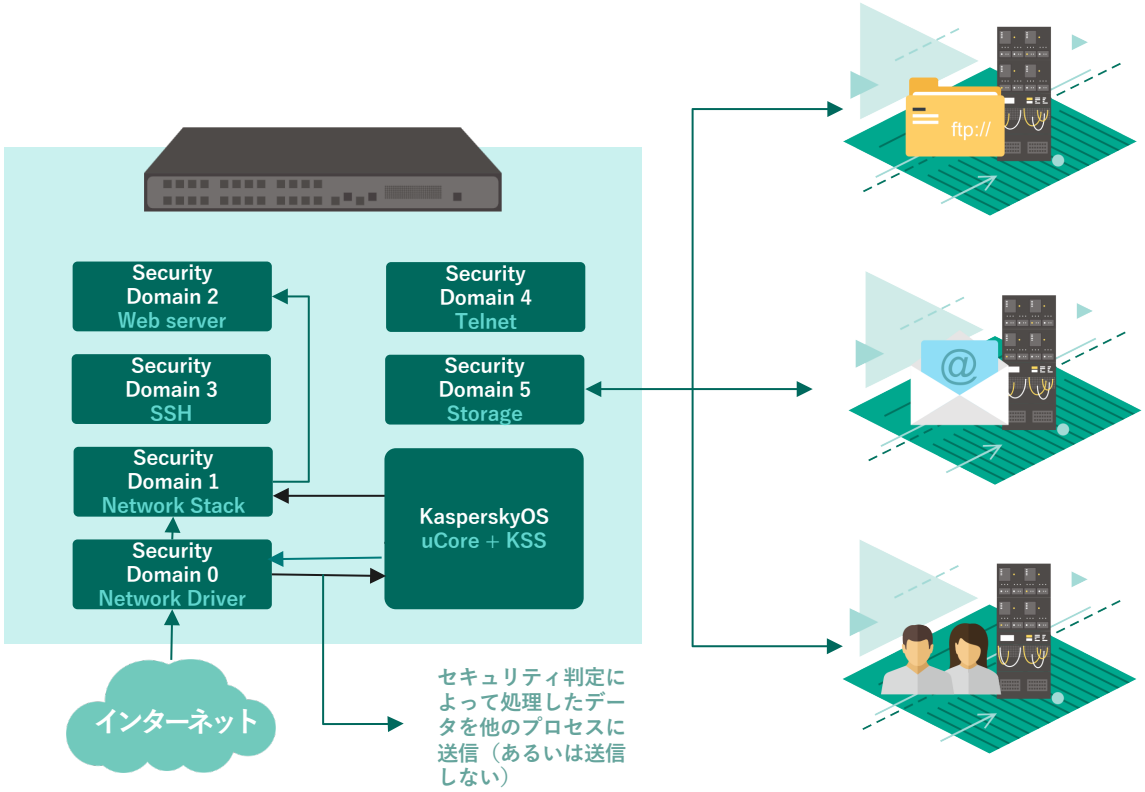
KASPERSKY lab

ユースケース – 通信機器

KASPERSKY OS

- ✓ セキュアブートと組み合わせ、OSとアプリケーションの完全性を担保
- ✓ アプリケーションをすべてのモジュールに分離しセキュリティドメインを適用
- ✓ 脆弱性やマルウェア保護による影響の極少化
- ✓ アクセス制御による暗号鍵など重要なデータを保護

▪ルーター、スイッチ、ファイアウォール、VPNなど



ユースケース – コネクテッド カー

KASPERSKY SECURE HYPERVISOR

- ✓ インフォテイメントシステムとECUの分離 (advanced driver assistance systems, AUTOSAR)
- ✓ 脆弱性による影響の極少化
- ✓ 暗号鍵、テレマティクスデータなど重要なデータの未承認アクセスからの保護
- ✓ セキュアブートと組み合わせ、未承認の改変からのファームウェアやアプリケーションを保護
- ▪ セントラルゲートウェイ、ヘッドユニットあるいは特定のECUに実装可能

ユースケース – 組み込みLINUXシステムをより安全に

KASPERSKY SECURITY SYSTEM

- ✓ OTAや遠隔保守プロセスの安全な実装をサポート
- ✓ 更新されるコンポーネントと更新用リモートエージェントの分離（コンシューマ機器にあるように）
- ✓ 信頼できないコンポーネントをサンドボックス化
- ✓ コンポーネント間の通信のプロパティを堅牢に

- ▪ PLC / 産業用 IoT 機器
- ▪ コンシューマ IoT 機器

開発環境

定義のための言語体系

インターフェース定義言語

- インターフェースとデータ構造を定義するのに用いる

コンポーネント定義言語

- 実装上のコンポーネントとインターフェースを定義するのに用いる

エンティティ定義言語

- エンティティとコンポーネントインスタンスのセットを定義するのに用いる
- セキュリティインターフェースを定義することも可能

IDL/CDL/ELD のサンプル

```
package FS

import kos.types

const UInt32 MaxPath  = 4096;
const UInt32 MaxData  = 4096;
const UInt32 MaxLabel = 4096;

typedef sequence<Char,  MaxPath>  Path;
typedef sequence<UInt8, MaxData>  Data;
typedef sequence<UInt8, MaxLabel> Label;
typedef UInt16 Mode;

interface IFileSystem {
  open(in Path path, in Mode mode, out Handle h);
  read(in Handle h, out Data data);
  write(in Handle h, in Data data);
}

interface IFileSystemSecurity {
  open(in Label label);
  read();
  write();
```

FS.idl

```
component ExtFS


ext2: FS.IFileSystem

ext3: FS.IFileSystem

ext4: FS.IFileSystem
```

ExtFS.cdl

```
entity FS


ext: ExtFS

security FS.IFileSystemSecurity
```

FS.edl

KASPERSKY lab

セキュリティ構成言語 (CFG)

セキュリティゲートの構成を定義を宣言するために用いる

エンティティのイベントとセキュリティポリシーとを関連付ける：

- エンティティの実行
- IPC 呼び出し
- セキュリティインターフェース要求

ポリシー固有の構成を追加することが可能

CFGのサンプル

```
execute EM.IExecute;

use execute policy init = SecurityServer.generic.exec.init;

use call policy allow = SecurityServer.generic.call.allow;
use call policy neq   = SecurityServer.generic.call.notEquals;

entity FS {
  execute default = init;

  call in  = allow;
  call out = allow;

  call in ext.ext2.open(mode) = neq {0} (mode), neq {-1} (mode);

  security read = allow;
}
```

KASPERKSY OSで できないこと

KASPERSKY lab

できないこと

- ✓ 起動システムの完全性チェック
- ✓ メモリーインジェクション攻撃対策（e.g.バッファオバーフローなど）
- ✓ 外部通信の改竄対策
- ✓ 開発環境、設計情報の偸盗対策

ご静聴ありがとうございました

KASPERSKY lab

制御システム向け
KICS FOR EMBEDDED

KASPERSKY lab

KASPERSKY INDUSTRIAL CYBER SECURITY (KICS)

レベル 4

ビジネス計画
および物流

ERP

レベル 3

生産管理

MES/LIMS

レベル 2

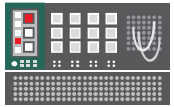
バッチ制御

連続制御

HMI/SCADA

レベル 1

離散制御（ディスクリート）



PLC/DCS/RTU

レベル 0

物理的

フィールドデバイス/制御機器

一般的なアンチウィルス製品

- ✓ アンチウィルス（ブラックリスト）
- ✓ アプリケーション管理（ホワイトリスト）
- ✓ Windows OS 対応（組み込み用でない）

Kaspersky Industrial Cyber Security (NETWORKS/NODES)

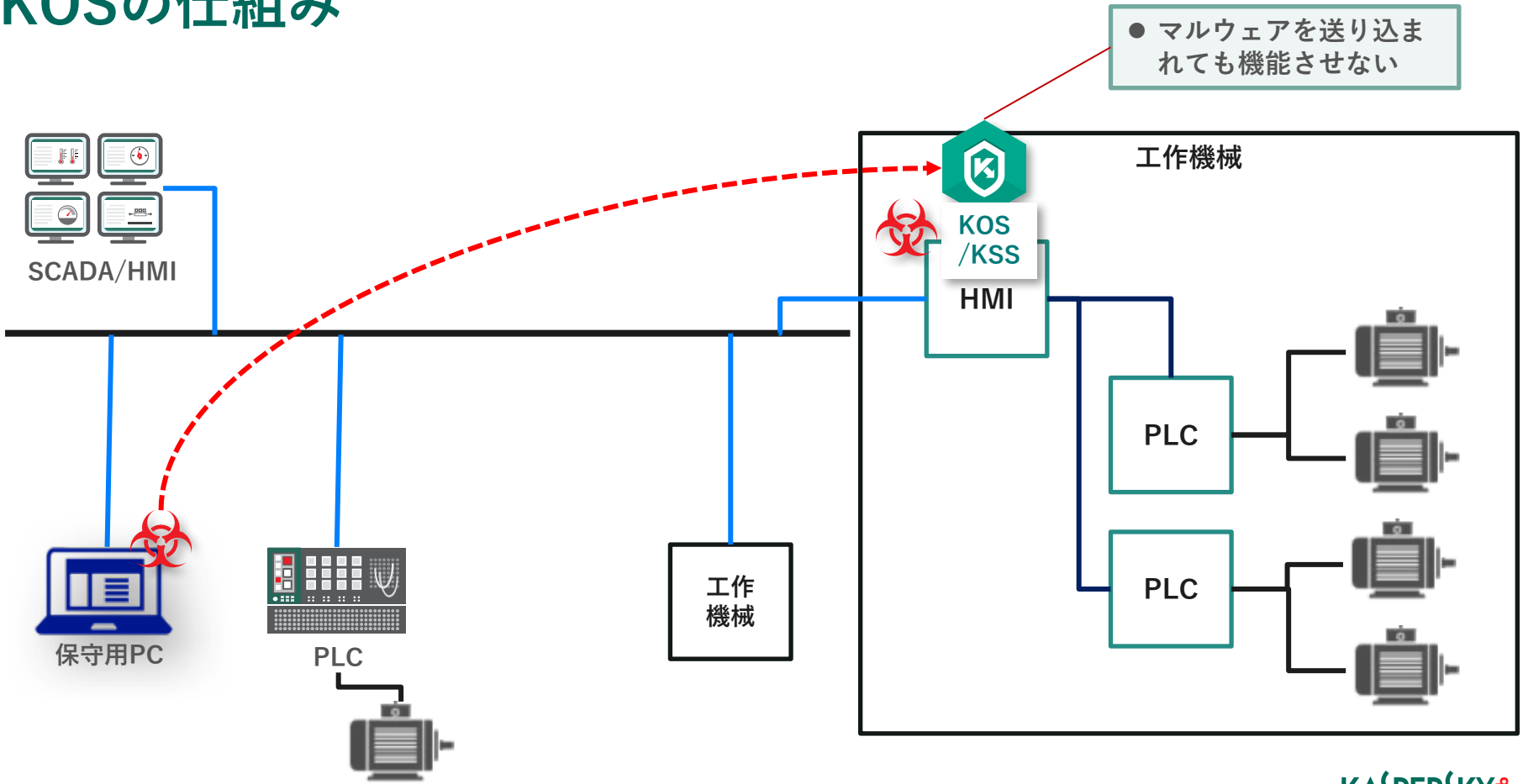
- ✓ Windows Embedded 対応（組み込み用）
- ✓ ネットワークアクセス制御
- ✓ デバイス、アプリケーション管理（ホワイト/ブラックリスト）
- ✓ 脆弱性評価・管理
- ✓ SCADA/HMI/PLC の監視と通知
- ✓ SIEM 統合
- ✓ アンチウィルス（エンドポイント保護）

KICS for Embedded (Kaspersky OS/KSS)

- ✓ OSレベルのポリシー制御
- ✓ 自社開発のマイクロカーネルリアルタイムOS

KASPERSKY lab

KOSの仕組み


● マルウェアを送り込まれても機能させない
SCADA/HMI
KOS
/KSS
HMI
工作機械
PLC
PLC
保守用PC
PLC
工作
機械