2004－OS－96　(2)
2004／6／17

プライバシ保護を実現するオペレーティングシステムにおけるコンテキスト管理手法

鈴来 和久[†]　一柳 淑美[†]　毛利 公一[††]　大久保 英嗣[††]
[†] 立命館大学大学院理工学研究科　[††] 立命館大学情報理工学部

現在, 我々は, データ漏洩を防止するコンテキスト適応型オペレーティングシステム (OS) を開発している. 本 OS は, ソフトウェアが介在することによってデータ漏洩が発生する場合を対象としたデータ保護を実現する. また, 変化しうる状況に適応したデータ保護を実現するために, 特に着目すべき重要な変化をコンテキストとして OS が監視する. 本 OS は, コンテキストの変化を検出した際に, ユーザプロセスが実行可能なシステムコールを制限することでデータ保護を実現する. 本論文では, コンテキストの管理手法, コンテキストに基づくシステムコールの実行制限手法について述べる. さらに, 提案手法を実現するソフトウェアの構成, 実装, 評価について述べる.

Context Management in Privacy-Aware Operating System

KAZUHISA SUZUKI[†]　YOSHIMI ICHIYANAGI[†]　KOICHI MOURI[††]　EIJI OKUBO[††]
[†]Graduate School of Science and Engineering, Ritsumeikan University
[††]Faculty of Information Science and Engineering, Ritsumeikan University

We have been developing a privacy-aware operating system based on the context-aware techniques. This operating system prevents user programs from leaking important data (such as personal information, confidential information, and copyright-protected contents) to other programs. Especially, the operating system monitors actions of user processes in order to realize the context-aware mechanism for data protection. When the operating system detects some kinds of context-change, it controls the execution of system calls. In this paper, the context management and context-aware control mechanism of system calls are described. Furthermore, the software architecture, implementaion, amd evaluation are also described.

1　はじめに

近年, 個人情報の保護を目的とした法律やガイドラインの制定が進んでいる. これらの動向は, プライバシ保護や機密保護に対する意識の高まりを反映したものといえる. しかし, 法律やガイドライン自体は, 情報漏洩を阻止する有効な手段とはいえない. 実際, 法律が交付された以降に, 顧客情報が流出し刑事事件に発展した事例などがある. このような背景から, 企業や行政機関における情報漏洩対策として, 顧客情報や機密情報などの重要な情報の取扱いに関する規約と, 情報を保護するためのハードウェアとソフトウェアを組み合わせたシステムを導入する事例が増加してきている.

また, 携帯可能な小型計算機の普及によって, 固定的な計算機ネットワーク構成から固定端末と移動端末が混在するノーマディック (normadic) な構成が主流となりつつある. ノーマディックな計算機ネットワーク構成では, 計算機の参加・離脱が発生することによってネットワーク構成が流動的に変化する. このほかに, ユーザがアクセスしているファイル, 計算機を利用するユーザの位置, 他のプロセスと通信しているプロセ

スの有無などが変化すると考えられる. ユーザの状況, 計算機の環境, プログラムの状態などが変化する場合, 満たすべき安全性の度合が変化するため, 安全性の度合をあらかじめ決定することが困難となる. このため, 状況の変化に適応したデータ保護を実現する必要がある.

以上の背景から, ソフトウェアの視点から情報漏洩を防止し, 変化しうる状況に適応したデータ保護を実現するソフトウェアプラットフォーム [1, 2] を開発している. 本プラットフォームでは, 特に計算機上で動作するプログラムが介在して情報漏洩が発生する場合を対象とし, プログラムの実行環境であるオペレーティングシステム (以下, OS と記す) においてデータ漏洩を防止する. OS がデータ漏洩を防止するため, アプリケーションプログラムの信頼性に関わらずデータを保護することができる. これは, コンピュータウィルスやワームなど, 不正行為を意図したプログラムに対しても有効なデータ保護手法であるといえる. また, 特別なシステムコールやライブラリ関数を追加せず, 既存のシステムコールの実行を制御することによりデータ保護を実現するため, 既存のアプリケーションに透過的に適用可能である.

本論文では, 変化しうる状況のうち, 着目すべき重要な変化をコンテキストとして OS が監視し, コンテキストの変化に基づいたシステムコール実行の制限手法を提案する. 本手法により, 変化しうる状況に適応したデータ保護が実現できる.

以下, 2 章で, 適応的データ保護機構の目標と設計方針について述べ, 3 章で, コンテキスト管理機構とコンテキストに適応したデータ保護機構の実現手法について述べる. 4 章で, 本機構の性能評価を述べ, 5 章で関連研究について述べる.

2 目標と設計方針

2.1 ソフトウェアが介在するデータ漏洩の防止

データ漏洩が発生する要因として考えられるものは, ハードウェア機器の盗難, 人の行為に対する信頼性欠如, アプリケーションの信頼性欠如など [1] があるが, 本研究では, 特にソフトウェアが介在する場合のデータの漏洩や改ざんを防止する手法に焦点を当てている. ソフトウェアが介在するデータの漏洩や改ざんの例として, 以下に示すものが挙げられる.

- ユーザがアプリケーションの操作を誤った場合
  部外秘のドキュメントファイルを, 誤って電子メールに添付したまま送信してしまう例がある. ユーザがアプリケーションの操作法に詳しくない場合や, 当該ファイルが重要なデータを含んでいることを確認せずに電子メールを送信した場合に, データの漏洩や改ざんが起こる.
- データ漏洩の要因となるアプリケーションの操作をユーザが故意に行った場合
  上記で述べた例や, 外部記憶装置にデータを複製する操作などを, 悪意のあるユーザが故意に行った場合に, データの漏洩や改ざんが起こる.
- 攻撃を意図したプログラムが実行された場合
  コンピュータウィルスやクラックツールなど, ソフトウェアの不具合を利用し, 計算機を機能不全にすることを意図したプログラムが存在している. それらを利用して攻撃者が外部から計算機に侵入した場合に, データの漏洩や改ざんが起こる.

本論文では, これらに代表されるデータの漏洩や改ざんの問題を解決するために, OS がその原因となるプロセスの挙動を制限するデータ保護手法について述べる. 本論文で提案する手法は, プロセスの動作履歴をコンテキストとし, その変動に適応してシステムコール実行の可否を制御する. これにより, ユビキタス環境に適用可能なデータ保護手法を実現する.

2.2 コンテキスト変動に適応したデータ保護

これまで, コンテキスト変動に適応するソフトウェア技術は, アプリケーションレベル, あるいはミドルウェアレベルで実現する場合が多かった [7]. ソフトウェアがコンテキストの変動に適応するためには, 着目すべき事象と, 着目した事象が変化する度合・頻度・時期・期間などに応じて, ソフトウェアの動作をどのように制御するかといった複雑なポリシを定義する必要がある. アプリケーションレベルで実現する場合, そのアプリケーションに特化したコンテキスト適応性が実現できる. しかし, データ保護の視点では, アプリケーションごとにデータ保護機能を実現しなければならない. 本論文で提案する OS は, 文献 [1] で示したコンテキストの階層化手法に基づき, システムコールを契機にプロセスの動作履歴をコンテキストとして取得する. 本手法により, ポリシが複雑化する問題を解決することができる.

ユビキタス環境は, ノーマディックな計算機構成が

主流となり, ユーザの位置, 利用しているアプリケーション, アクセスしているデータなどが頻繁に変化すると想定できる. 従来の固定的な計算機構成では, データ保護ポリシをあらかじめ定義することにより, 必要な安全性を満たすことができた. しかし, ノーマディックな計算機構成では, 計算機が移動するため, 安全性の度合をあらかじめ決定することが困難である. 安全性の度合の指定を誤った場合, データ漏洩や改ざんが発生する危険性が高まる. そこで, OS がコンテキストの変動を監視し, コンテキストの変動に適応してシステムコール実行の可否を制御することによりデータを保護する.

2.3 データ保護ポリシとコンテキスト

ユビキタス環境においてデータの漏洩や改ざんを防止する場合, 以下に示すようなデータ保護ポリシを定義することが考えられる.

- ユーザの位置に基づくアクセス制御
- プロセスの動作履歴に基づくアクセス制御
- 通信の有無に基づくアクセス制御
- 時限付きアクセス制御

上記で示した保護ポリシには, コンテキストとして位置情報, 通信の有無, プロセスの動作履歴, 時刻が含まれている. 位置情報は, OS が管理する情報に含まれていないため, Publish / Subscribe モデルを適用し, 位置情報システムから取得する. すなわち, OS は位置情報システムに対して着目すべきユーザを通知し (Subscribe), 位置情報システムは OS にユーザの位置を広告する (Publish). プロセスの動作履歴は, プロセスが発行したシステムコールを時系列のデータとして記録することで実現できる. その際, システムコールの引数に着目し, システムコールの種類と引数をプロセスごとに記録する. 通信の有無は, システムコールを契機に検出することができる. ただし, 共有メモリ方式のプロセス間通信は, メモリコピーによって通信を実現するため, システムコールによる検出は困難となる. この問題を解決するために, 文献 [3] で提案されている手法の適用を検討している.

2.4 適応性の実現

2.3 節で述べたコンテキスト検出手法により, OS が検出すべきコンテキストは, システムコールの種類, 引数, システムコールを発行したプロセスの ID, 時刻となる. このように, ある 1 回のシステムコールに着目したコンテキストは, データ保護ポリシと比較して単純な表現となる. プロセスごとに時系列データとしてコンテキストを管理することにより, プロセスの生成後にアクセスしたファイル, ファイルアクセス後の時間経過などのプロセスの動作を把握できる. これにより, データ保護ポリシに記述される複雑なコンテキストが成立したか否かを, 時系列データにおけるコンテキストの条件分岐結果との比較結果で判断できる (図 1 参照).

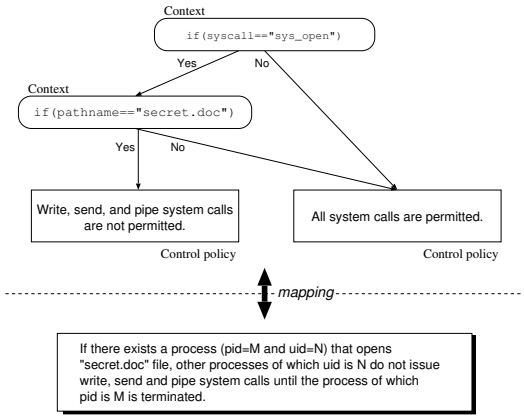

図 1　複雑なコンテキストの表現法

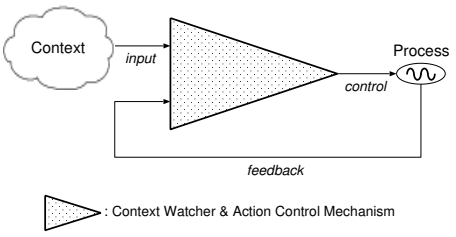

図 2　フィードバックシステムモデル

本論文で提案するコンテキスト管理手法では, プロセスの挙動を着目すべきコンテキストの 1 つに挙げている. プロセスの挙動は, データ保護を実現するために制御される対象である. コンテキストを入力とし, プロセスの挙動を出力と考えた場合, 本手法はフィードバックシステムと捉えることができる (図 2 参照). フィードバックシステムは, 設定された数値目標を維持するために, 出力結果に応じた入力を与えるシステムである. 本手法では, このフィードバックシステムモデルに基づき, データの漏洩や改ざんの原因となるプロセスの挙動を制限するため, プロセスの動作履歴をコンテキストとして与える. 以上の特徴より, 本論文では, 適応性の実現をフィードバックシステムモデルに基づくプロセスの挙動制御の実現と定義する.

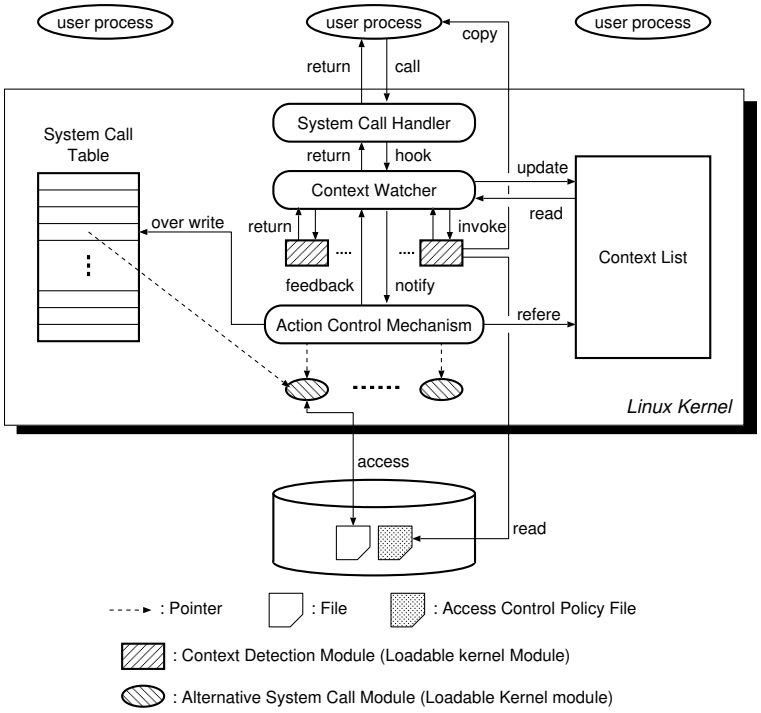

図 3　データ保護機構の全体構成

3　実装

3.1　全体構成

データ保護機構を, Linux カーネル 2.6.1 に変更を加える形で実装した. 本機構は, コンテキスト管理部 (Context Watcher) とプロセス挙動制御部 (Action Control Mechanism) から構成される (図 3 参照).

3.2　コンテキスト管理部

コンテキスト管理部の構成を, 図 4 に示す. コンテキスト管理部は, すべてのシステムコールに共通した処理とシステムコール依存の処理を分離し, システムコール依存の処理を LKM (Loadable Kernel Module) で実装した. すべてのシステムコールに共通した処理を行うモジュールは, 静的に Linux カーネルに組み込まれるように実装した. コンテキスト管理部の処理の流れを, 以下に示す.

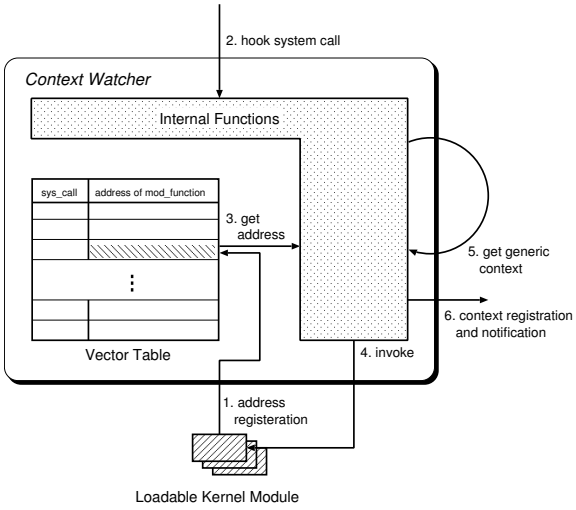

図 4　コンテキスト管理部

1. モジュールをロードする際に, コンテキスト管理部が呼び出すモジュール関数へのポインタをベクタテーブルに登録する.
2. システムコールハンドラ内でフックしたシステムコールを特定する.
3. ベクタテーブルを参照し, フックしたシステムコールを処理するモジュール関数のアドレスを取得する. アドレスが登録されていない場合, そのシステムコールに関するコンテキストを取得する必要がないものと判断し, 通常のシステムコール処理を実行する.
4. モジュール関数を呼び出し, システムコール依存のコンテキストを取得する.

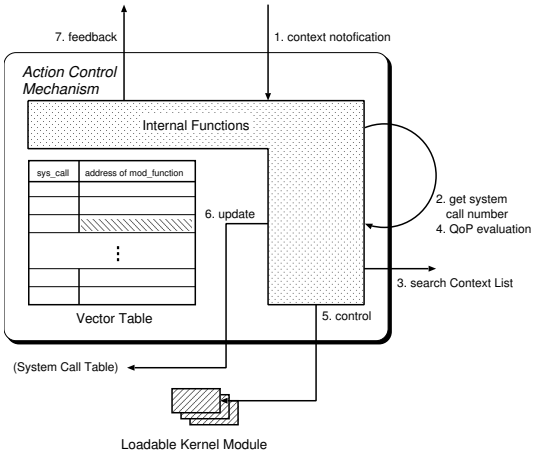

図 5　プロセス挙動制御部

5. 各システムコールに共通したコンテキストを取得する.
6. 各プロセスごとに作成したリストに取得したコンテキストを登録する. また, プロセス挙動制御部にコンテキストを通知し, プロセス挙動制御部に制御を移行する.

3.3　プロセス挙動制御部

プロセス挙動制御部の構成を, 図 5 に示す. プロセス挙動制御部は, コンテキスト管理部と同様に, すべてのシステムコールに共通した処理とシステムコール依存の処理を分離した. このため, ソフトウェア構成は, コンテキスト管理部と同様の構成になっている. すなわち, システムコール依存の処理を LKM で実装し, すべてのシステムコールで共通の処理をするモジュールは, 静的に Linux カーネルに組み込まれるように実装した. また, 既存のシステムコールを置き換えるモジュール関数へのポインタをベクタテーブルに登録する. プロセス挙動制御部の動作を, 以下に示す.

1. コンテキスト管理機構から, コンテキスト変動の通知を受ける.
2. 通知されたコンテキストから, 制御対象となるプロセスを特定する.
3. コンテキストリストを参照し, 以下の情報を取得する.
   i. 制御対象プロセスに対して現在許可されている操作
   ii. 制御対象プロセスが過去に発行したシステムコールと引数
   iii. アクセス中のファイルのデータ保護ポリシ
4. 上記で取得したコンテキストに基づき, 安全性の度合 (QoP: Quality of Protection) [1] を導出する.
5. QoP を満たすように, システムコールの実行可否を決定する. 実行する場合, 既存のシステムコールとモジュール関数のどちらを実行するか決定する.
6. 必要に応じてシステムコールテーブルのポインタを変更する.
7. システムコールの制御結果をコンテキスト管理部にフィードバックする.

なお, ディスクに保存されたファイルに対するアクセス制御を実現するため, open, read, write, close の各システムコールに対応するモジュールの実装を行った. モジュールの実装は, 文献 [2] で示している.

次に, 既存のシステムコールを置き換える利点を, 以下に示す.

- 柔軟なアクセス制御が実現可能
  セキュアオブジェクト [1] は, データ操作メソッドを介したデータアクセスのみを許可する. しかし, 既存のアプリケーションは, データ操作メソッドによるデータアクセスを想定していない. このため, 既存のアプリケーションに対して, セキュアオブジェクトによるデータ保護を適用することが困難である. そこで, データ操作メソッドの呼出しを代行するモジュール関数と既存のシステムコールを置き換えることにより, この問題を解決できる.
- 通常のシステムコール処理への復帰が容易
  プロセスの挙動を制御する処理を, 既存のシステムコール中に実装した場合, システムコールテーブルのポインタを変更する必要がなくなる. しかし, プロセスの挙動を制御する必要がない場合もこの処理は実行されるため, そのためのオーバヘッドが問題となる. プロセスの挙動を制御する処理をモジュールとして実装した場合, システムコールテーブルのポインタを, モジュール関数に置き換える前の値に復帰することにより, このオーバヘッドを削減できる.

表 1　実験環境 (コンテキスト管理部)

Machine	IBM PC / AT 互換機
CPU	Intel Celeron Processor 850MHz
Memory	256MB
HDD	Ultra ATA100 80GB
Kernel	Linux Kernel 2.6.1 + コンテキスト管理部
File System	Reiser File System

4　評価

4.1　コンテキスト管理部

本論文で提案したコンテキスト検出手法の性能を評価するために, 表 1 に示す実験環境を構築した. 実験では, open システムコールについて, オープンするファイルの数が 1, 100, 1000, 10000 個の試行を 10 回繰り返し, システムコール処理にかかった時間の平均値を計測した. なお, コンテキスト管理部のオーバヘッドのみを計測するため, プロセス挙動制御部を無効化した Linux カーネルで実験を行った. さらに, 標準の Linux カーネルにおける open システムコールの処理時間を計測し, 25.60 μs という結果を得た.

実験結果を, 表 2 に示す. 実験結果から, open システムコールでは, コンテキストを取得することによって, 約 49 μs のオーバヘッドが発生することがわかった. また, 連続して open システムコールを発行する場合, ファイル数の増加にともなってオーバヘッドが急激に増加することがわかった.

4.2　プロセス挙動制御部

ハードディスクに保存されているファイルに対して, データ保護ポリシに基づいたアクセス制御が実現されているかを検証するために, Linux カーネル 2.4.22 にプロセス挙動制御部を実装し, open, read, write システムコールを制御する LKM を実装し, 実験した. 具体的な実験内容を以下に示す.

表 2　コンテキスト管理部の性能評価

ファイル数	平均処理時間
1	77.54 μs
100	77.65 μs
1000	238.07 μs
10000	2683.90 μs

- 保護対象ファイルのみをオープンし, 読出し処理や書込み処理をしてからファイルをクローズする.
- 保護対象外のファイルのみをオープンし, 読出し処理や書込み処理をしてからファイルをクローズする.
- 保護対象ファイルと保護対象外ファイルをオープンし, これらのファイルに読出し処理や書込み処理をしてからクローズする.

上記の実験により, 保護対象ファイルのデータ保護ポリシに違反する読出し処理や書込み処理は実行されず, 保護対象外ファイルはアクセス可能であることを確認した.

5　関連研究

5.1　プロセスの動作を制限する手法

悪意のあるプログラムの実行を制限するために, プログラムの実行実体であるプロセスの動作を監視し, 制御する手法 [4, 5, 6] がある. 文献 [4] では, 1 つのプロセスを構成する複数のモジュールに対してそれぞれ保護ドメインを割り当てている. これにより, 1 つのプロセスの中でモジュールごとに異なるアクセス権限を持たせ, モジュール間の保護を実現している. また, 1 つのプロセス内では, 1 つの細粒度保護ドメインだけがシステムコールをフックする特権を持ち, システムコールが発行された際にそれらの内容をチェックし, システムコール実行の可否を決定する. このように, システムコールに着目している点で, 本論文で提案した手法と類似している. しかし, 本論文では, コンテキストに適応した手法を提案し, ユビキタス環境に適応可能なデータ保護を実現している点で異なる.

文献 [5] では, 信頼性の低いプログラムを安全に動作させるための実行環境を提供する. 具体的には, ソフトウェアを仮想的なファイルシステム内に閉じ込めた状態で実行する環境を提供する. また, プログラムとプログラムの実行に必要なファイルを, 仮想的なファイルシステムを利用して流通させることができる. 信頼性の低いアプリケーションを対象としている点は, 本研究の背景と共通している. また, 仮想的なファイルシステムを利用したファイルの流通手法を, 文献 [1] で提案したセキュアオブジェクトに応用することにより, 安全な遠隔データアクセス方式が実現できると考えている.

文献 [6] では, セキュリティポリシを動的に切替え

ることができるリファレンスモニタを実現している. 具体的には, 任意の関数から他の関数へ制御を移す命令を, 権限の切替えを行う契機とする手法を実現している. 一方, 本論文では, コンテキストの変動を契機とするプロセスの挙動制御手法を述べた. 両者を統合することで, より適応的なプロセスの挙動制御が実現できると考えている.

5.2 コンテキスト適応性の実現

文献 [7] は, コンテキストに適応可能なアプリケーションの開発を支援するフレームワークを実現する手法が提案されている. 本論文で提案したコンテキスト適応性の実現手法と比較した場合, ユビキタス環境に適応したソフトウェアの実現を目的としている点が類似している. また, アプリケーションの動作について記述された抽象的なコンテキスト (ベースコンテクスト) と, その動作を実現するためにアプリケーションの処理に必要な情報 (メタコンテクスト) を階層化している点も類似している.

フレームワークとは, メタコンテクストと抽象度の高いコンテクスト情報を抽出するための情報を設定する API を提供するソフトウェアアーキテクチャであり, アプリケーションから独立している. ベースコンテクストはアプリケーション開発者が設定し, メタコンテクストは, フレームワーク提供者がフレームワーク内部で設定する. しかし, 本論文で提案した手法では, フィードバックモデルを適用することにより, プロセスの挙動を制御するためのコンテキスト (メタコンテクスト) を取得する点で異なっている.

5.3 プライバシ保護のための アプリケーションフレームワーク

本研究と同様にプライバシ情報の漏洩防止を目的とした研究として, 文献 [8] で提案されている手法では, 個人情報を送信することなくサービスを利用可能とするアプリケーションフレームワークを実現している. 提案手法では, ユーザの携帯端末で保持している個人情報に基づいてアプリケーションを制御するコマンドを生成し, アプリケーションが稼働している計算機に制御コマンドを送信する. 我々の提案手法と比較した場合, アプリケーションを信頼していない点, 個人情報を送信しない点など, 研究背景について我々と共通している点が多い. しかし, 我々の提案手法は, 不正アクセスや情報漏洩を意図したプログラムによる情報漏洩も防止することができる. さらに, コンテキスト変動に適応可能なデータ保護を実現している点で異なっている.

6 おわりに

本論文では, データの漏洩防止を実現するコンテキスト適応型 OS を実現するためのコンテキスト管理手法について述べた. 本 OS は, フィードバックシステムモデルを適用したコンテキスト管理を実現する. 本 OS が着目すべきコンテストは, システムコールの種類, 引数, システムコールを発行したプロセスの ID, 時刻となる. プロセスごとに時系列データとしてコンテキストを管理することにより, プロセスの動作を把握できる. これにより, データ保護ポリシに記述される複雑なコンテキストが成立したか否かを, 時系列データにおけるコンテキストの条件分岐結果との比較結果で判断できる.

また, 本論文では, コンテキストに適応可能なデータ保護の実現手法について述べた. コンテキストに適応可能なプロセスの挙動制御を実現することにより, ユビキタス環境に適用可能なデータ保護が実現できる. また, ユーザの操作ミスや悪意のある操作が発生した場合に, データ漏洩を防ぐことができる.

今後は, QoP パラメータの定量的算出手法を検討する. また, セキュアオブジェクトによる遠隔データ制御方式を実現する.

参考文献

[1] 鈴来 和久, 毛利 公一, 大久保 英嗣: データの拡散防止を実現するコンテキスト適応型ソフトウェア基盤, 情報処理学会研究会報告 2004-OS-95, Vol.2004, No.17, pp.57–64 (2004).

[2] 一柳 淑美, 鈴来 和久, 毛利 公一, 大久保 英嗣: コンテキストに適応可能なデータ保護機構におけるファイルアクセス制御, 情報処理学会第 66 回全国大会講演論文集 (1), pp.95–96, 2004.

[3] 河野 健二, 品川 高廣, 高橋 雅彦, 益田 隆司: 細粒度保護ドメインのための多重保護ページテーブルの提案と実装, 情報処理学会研究会報告 1999-OS-81, Vol.1999, No.32, pp.89–94 (1999).

[4] 品川 高廣, 河野 健二, 益田 隆司: 細粒度保護ドメインによる軽量サンドボックスの実現, 情報処理学会研究会報告 2002-OS-90, Vol.2002, No.60, pp.87–94 (2002).

[5] 大山 恵弘, 神田 勝規, 加藤 和彦: 安全なソフトウェア実行システム SoftwarePot の設計と実装, コンピュータソフトウェア, Vol.19, No.6, pp.2–12, 日本ソフトウェア科学会 (2002).

[6] 阿部 洋丈, 加藤 和彦: セキュリティポリシーの動的切替機構を持つリファレンスモニタシステム, コンピュータソフトウェア, Vol.20, No.3, pp.2–16, 日本ソフトウェア科学会 (2003).

[7] 藤波 香織, 山邉 哲生, 中島 達夫: コンテクストアウェアなアプリケーションフレームワークにおけるメタコンテクスト情報の利用方法の提案とその応用, コンピュータソフトウェア, Vol.21, No.1, pp.46–59, 日本ソフトウェア科学会 (2004).

[8] 田丸 修平, 岩谷 晶子, 高汐 一紀, 徳田 英幸: プライバシを考慮したパーソナライゼーションを実現するアプリケーションフレームワーク: 情報処理学会研究会報告 2003-OS-93, Vol.2003, No.42, pp.49–56 (2003).