Privacy-Aware OS *Salvia* におけるデータアクセス時のコンテキストに基づく適応的データ保護方式

鈴来和久[†] 一柳淑美[†]
毛利公一[††] 大久保英嗣[††]

我々は，個人情報をはじめとするプライバシデータの漏洩を防ぐ OS *Salvia* を開発している．プライバシデータは，データ提供者とデータ管理者の合意に基づいて取り扱う必要がある．そのため，*Salvia* ではプライバシデータを含むファイルの保護方法をデータ保護ポリシとして記述可能としている．データ保護ポリシには，従来のファイルの読み出し，書き込み，実行の権限設定に加えて，アクセス制限を課するための条件として，ファイルアクセスが発生した際の状況を示すコンテキストを記述できる．これによって従来の OS よりも細かく，かつ Trusted OS よりも柔軟な，プライバシデータに適したアクセス制御を実現できる．すなわち，*Salvia* は，データ保護ポリシが定義されたファイルにアクセスを試みたプロセスのアクセス要求のみをコンテキストに適応して制御可能としている．本論文では，*Salvia* の設計と実装について述べるとともに，データ保護が可能であることを実証的に示す．

An Adaptive Data Protection Method based on Contexts of Data Access in Privacy-Aware Operating System *Salvia*

Kazuhisa Suzuki,[†] Yoshimi Ichiyanagi,[†] Koichi Mouri[††] and Eiji Okubo[††]

We have been developing a privacy-aware operating system *Salvia* that prevents the privacy data such as personal information from leaking. It is necessary to manipulate the privacy data based on a mutual agreement between the data owner and the data administrator. In *Salvia*, in order to realize such an agreement, the protection methods of files that include the privacy data can be described as the data protection policies. In addition to the conventional permissions (read/write/execution), the context that shows the situation when the file access is generated can be described in the data protection policies. In *Salvia*, by enforcing these policies, the access control that is more detailed than the conventional operating systems and is also more flexible than the trusted operating systems can be achieved. Namely, by adapting to the context, *Salvia* is enabled only for the access request of the processes that have tried to access to the file associated with the data protection policies. In this paper, the design and implementation of *Salvia* is described, and also confirmed the effectiveness of *Salvia*'s context-aware data protection method by practical examples.

1. はじめに

2005 年 4 月より，個人情報の保護に関する法律（以下，個人情報保護法と記す）が施行された．個人情報を取り扱う事業者は，個人情報の取扱い方法に関する方針（プライバシポリシ）の策定，個人情報を処理する計算機ネットワークに対するセキュリティ機能の強化，個人情報を扱う業務に従事する作業者に対するプライバシポリシの周知徹底など，個人情報の漏洩を防ぐための対策を行っている．しかし，個人情報の漏洩は依然として発生し続けており，解決されていない．個人情報流出事故に関する事業者調査結果[1)] では，個人情報の漏洩などのデータ漏洩が発生する要因として，(1) 社員のミスによる紛失や漏洩，(2) 社員の外部への持ち出し，(3) 盗難，(4) 設定ミスなどの技術的不備があげられている．具体例をあげると，(1) の事例として，顧客情報を扱う業務を担当する従業員が，顧客情報が保存されているファイルを誤って電子メール

† 立命館大学大学院理工学研究科
Graduate School of Science and Engineering, Ritsumeikan University
†† 立命館大学情報理工学部
College of Information Science and Engineering, Ritsumeikan University

に添付し，その電子メールを送信してしまうといった操作ミスによって発生するデータ漏洩がある．(2) の事例として，個人情報が保存されている計算機が従業員に持ち出されることによって発生するデータ漏洩がある．本論文では，これらの (1) や (2) のような，正当なアクセス権限を持つ者によるデータ漏洩を防ぐ手法をオペレーティングシステム（以下，OS と記す）で防ぐ手法を提案する．

計算機では，取得したデータをディスクなどの 2 次記憶装置にファイルとして保存する．従来の OS が提供するファイルの読み出し，書き込み，実行の各アクセス権限の検査（DAC: Discretionary Access Control）は，プロセスがどのような目的でファイル入出力を行うかを意識するものではなく，当該ファイルに対するアクセス権限を示すものである．このため，ファイルを読み出したプロセスに対して，他のファイル，ソケット，パイプなどの資源に対するアクセスを制御することができない．

DAC における問題点を解決するデータアクセス制御手法の 1 つに，強制アクセス制御（MAC: Mandatory Access Control）がある．MAC によるアクセス制御は，DAC に加えて，計算機の管理者が設定したセキュリティポリシに基づき，要求されたアクセスの許可・不許可を制御する．このアクセス制御は，管理者権限を持つユーザとプロセスに対しても適用されるため，MAC によりすべてのユーザとプロセスに例外なくアクセスを制御することができる．しかし，MAC によるアクセス制御は，その制約が厳密過ぎるという問題がある．文献 2) では，MAC を実現する代表的なセキュリティモデルである Bell-LaPadula モデル[3] などの情報フロー制御に基づいたアクセス制御は，制約が強過ぎるために実際のアプリケーションの作成が困難であるという問題点を指摘している．特に，データ漏洩を防ぐために厳しい制約を課する場合，アプリケーションが処理を継続するために必要な計算機資源へのアクセスが禁止されるため，そのアプリケーションの実行が困難となる．また，制約を緩和した場合，DAC と同様に，読み出し権限を持つユーザによるデータ漏洩を防ぐことが困難となる．

本論文では，DAC や MAC の問題を解決するデータ保護手法を実現する OS *Salvia* について述べる．特に，個人情報などのプライバシを考慮すべきデータ（以下，プライバシデータと記す）は，そのデータごとに利用目的や提供範囲が異なるという特徴を持つ．したがって，*Salvia* では，保護対象となるプライバシデータを含むファイルを保護するためのデータ保護ポリシをファイルごとに設定可能とする．また，データ保護ポリシはデータ提供者の意思（保護方法や提供範囲）を反映可能とする．このポリシに基づくアクセス制御を実現するため，*Salvia* では，プライバシデータを含むファイルとデータ保護ポリシを組にして管理する．さらに，アクセス制御を柔軟にするため，*Salvia* はプライバシデータにアクセスを試みたプロセスに対して，このプロセスの過去の動作履歴を含めた，データアクセス時の状況（コンテキスト）とデータ保護ポリシに基づくアクセス制御を実現する．

以下，本論文では，2 章で関連研究について述べる．次に，3 章で *Salvia* のデータ保護モデルとその実現方式について，4 章で *Salvia* の設計と実装について述べる．さらに，5 章で機能評価について述べる．

2. 関 連 研 究

近年多発している情報漏洩事故の発生要因が，正当なアクセス権限を持つユーザによる不正アクセスである事例が多い．このため，不正アクセスを防ぐための様々なセキュリティ技術が提案されている．それらは，専用のハードウェアを用いた実現方式とソフトウェアによる実現方式に分類できる．

ハードウェアを用いたセキュリティ技術として，接続される機器間の相互認証機能とデータの暗号化機能を持つ外部記憶装置や，指紋認証機能を搭載した計算機がある．これらのハードウェアは，それらが利用可能な計算機やユーザを限定することができる．すなわち，正当な機器に接続する場合や正当なユーザが利用する場合のみ，それらに保存されたデータにアクセスすることができる．これらのハードウェアは不正アクセスを防止可能であるが，正当なアクセス権限を持つユーザによる操作ミスや不正アクセスを防止できない．

ソフトウェアによるデータへの不正アクセスを防ぐには，プログラムが利用可能な計算機資源や実行可能な処理を制限するサンドボックス環境やリファレンスモニタを適用する手法，DAC や MAC などのアクセス制御方式やそれらを適用した Trusted OS を利用する手法，プライバシを考慮したデータ保護モデルを適用する手法などがある．

細粒度保護ドメイン[4] は，サンドボックス環境を実現する手法の 1 つである．この手法の特徴は，1 つのプロセス内に計算機資源に対する異なるアクセス権限を持つ複数の保護ドメインを構成できる点と，保護のためのポリシと機能を分離するために，ポリシモジュールと呼ばれるカーネルとは独立したモジュールが決定する保護ポリシに基づいて，カーネルがシステムコー

ルの実行を制御する点にある．これにより，悪意のあるプログラムによるファイルへの不正アクセスの防止を実現している．しかし，細粒度保護ドメインでは，プロセスを対象として保護ポリシを作成，適用する．このポリシは，データ提供者の意思を反映することを想定していないため，プライバシデータの保護には適さない．SoftwarePot[5]は，サンドボックス環境を実現するソフトウェアである．SoftwarePot は，プログラムの実行環境として仮想的なファイルシステムを構築し，仮想的なファイルシステムと計算機資源へのアクセスをシステムコールフックによって制御する．この制御は，仮想的なファイルシステムに関連付けられたセキュリティポリシに基づいて行われる．このポリシの作成や変更は，仮想的なファイルシステムを構築するユーザか，それを利用してプログラムを実行するユーザによって行われる．したがって，データ提供者とポリシ記述者が異なる場合が考えられる．この場合，データ提供者の意思を反映したアクセス制御が実現されない．

REMUS[6]は，バッファオーバフロー攻撃の防止を目的とするシステムコールのリファレンスモニタを実装した OS である．その特徴は，攻撃が行われた際の危険度に基づいてシステムコールを分類し，危険度の高いシステムコールに関して，そのシステムコールごとに設定されたポリシに基づいて実行の可否を判定する．特に，管理者権限を持つプロセスは，システムコールの種類とその引数の内容が検査され，ポリシによって許可された特定のアクセスのみ実行可能とすることにより，システムの安全性を実現している．SysGuard[7]は，リファレンスモニタの 1 つで，システムコールの前後でアクセス権限の検査を行う OS である．アクセス権限の検査は，カーネルに組み込まれたガードと呼ばれるモジュールで行う．SysGuard の特徴は，多くの種類の専用のガードを組み合わせて適用することにより，ガードの実装を単純化してそれ自体の堅牢性を高めることができる点と，ガードによるアクセス制御の適用範囲を柔軟に設定できる点にある．REMUS は，攻撃者によってプロセスの制御が奪われた場合に，そのプロセスが要求する不正アクセスを制御することを目的としたリファレンスモニタであり，計算機の管理者がアクセス制御のポリシを設定する．管理者がアクセス制御のポリシを設定する点は，SysGuard も同様である．しかし，REMUS や SysGuard は，当該プロセスの過去の動作履歴に基づいたアクセス制御を行うものではないため，たとえば 1 章で述べた具体例の (1) に対処できない．すなわち，顧客情報ファイルを誤って添付した電子メールと通常の電子メールを区別し，顧客情報ファイルが添付された電子メールのみ送信を禁止するといった制御が実現できない．*Salvia* は，漏洩を防ぐ必要のあるファイルにアクセスしたプロセスに関するシステムコールの発行履歴を時系列データとして保持する．*Salvia* は，この履歴に基づき，上記のような電子メールの送信をシステムコールを制御することにより禁止することができる．

履歴を用いたアクセス制御方式として，モバイルコードを安全に実行するための Java のセキュリティ機構を拡張した Deeds[8]がある．Deeds では，監視や保護の対象である計算機資源に対するアクセス要求をセキュリティイベントとして定義し，プログラムごとにセキュリティイベントを履歴として記録する．Deeds は，プログラムの振舞いを示すセキュリティイベントの履歴のみに着目したアクセス制御手法を適用している．*Salvia* は，プロセスの振舞いを示すシステムコールの履歴に加えて，アクセス要求が発生した際のコンテキストである計算機の位置や時刻に着目し，コンテキストとデータ保護ポリシに基づくアクセス制御を実現している．コンテキストに適応可能なアクセス制御は，Deeds では実現されていない．また，Deeds は，プログラムの実行中に動的にクラスをロードすることがある場合，そのようなプログラムの実行を拒否するという制限があるが，*Salvia* はプログラムの実行を拒否しない．

アクセス制御方式として 1 章で述べた DAC，MAC に加えて，Role-Based Access Control[9]，TE（Type Enforcement）[10]などの方式がある．DAC は，ファイルの読み出し，書き込み，実行の各権限をユーザやグループを単位として設定する．しかし，1 章で述べた理由により，プライバシデータの保護には適さない．Bell-LaPadula モデルは MAC を実現するセキュリティモデルの 1 つで，数学的モデルとして記述された情報フローセキュリティモデルである．また，米国国防省における計算機システムのセキュリティ要件（TCSEC: Trusted Computer System Evaluation Criteria）[11]の基礎となったモデルである．情報フローセキュリティモデルは，秘匿性を重視したセキュリティモデルに分類される．Bell-LaPadula モデルの特徴は，アクセスの主体とアクセスの対象にセキュリティクラスを与え，セキュリティクラス間の情報フローの許可・不許可を制御することによって，システム全体の情報フローを制御する点にある．アクセスとは，参照，更新，生成，削除，実行，追加などのデータ操作を指す．一方，経済協力開発機構（OECD）で定義されたプラ

イバシ保護の原則[12]や，それに基づいて策定された個人情報保護法では，プライバシデータをその提供者の意思を反映して利用，管理することを義務付けている．Bell-LaPadula モデルなどの従来のセキュリティモデルや，それに基づいた MAC などのアクセス制御方式では，管理者が決定した保護ポリシに基づいてアクセスの可否を判定するため，利用目的や提供可能な範囲がデータごとに異なるプライバシデータの保護には適さない．Role-Based Access Control や TE は，ユーザやプロセスごとに計算機資源へのアクセス権限が設定可能である．最小特権の原則に基づき，ユーザやプロセスに対して必要なアクセス権限のみを設定することにより，アプリケーションの実行に必要となるファイルのみアクセス可能とすることができる．しかし，プライバシデータを含むファイルへアクセスが許可されたユーザやプロセスに対して，ソケットへのアクセス権限が与えられている場合，このユーザやプロセスによって，1 章で述べた具体例の（1）に対処できない．さらに，Role-Based Access Control や TE は，アクセス要求が発生した際の状況に応じてアクセスの可否を制御するための仕組みがない．このため，1 章で述べた具体例の（2）に対処できない．*Salvia* では，前者の問題に対して，プロセスが発行したシステムコールの履歴に基づき，保護を必要とするファイルにアクセスしたプロセスが発行するシステムコールを制御する．後者の問題に対しては，システムコールが発行された際の計算機の位置や時刻に基づき，データ保護ポリシに記述されている位置や時刻に一致するか否かに応じてシステムコールの実行の可否を制御する．

TrustedBSD[13]は，MAC を実現するセキュリティモデルとしてマルチレベルセキュリティ，Biba モデル[14]，TE（Type Enforcement）モデルが利用可能な Trusted OS で，TCSEC の B1 レベルの安全度を実現している．TE モデルは，Security-Enhanced Linux[15]でも用いられている．TrustedBSD などの Trusted OS は，MAC によってユーザの操作ミスや不正アクセスに起因するデータ漏洩を防ぐ．しかし，1 章で述べたように，MAC にはアクセス制御の制約が厳密過ぎる問題がある．さらに，MAC のためのアクセス制御ポリシは計算機の管理者によって設定されるため，複数の異なるデータ提供者の意思やプライバシポリシを保護ポリシに反映することができない．

プライバシを考慮したデータ保護モデルとして，分散ラベルモデル[2]がある．分散ラベルモデルは，意図しないユーザやプログラムにデータが伝搬することを防ぐために，次に示す手法によってデータフローを制御する．

- データを単位としてラベルを付与する．
- ラベルは，その所有者とデータの読み出しを許可する reader の組の 2 つの要素を持ち，データの読み出しを制御するポリシとなる．
- ラベルは，その所有者か，所有者と同じ権限を持つ別の所有者のみが書き換え可能である．
- ある変数から別の変数へデータコピーが発生する場合，コピー元のデータのラベルがコピー先のデータにも適用される．

データ提供者がポリシを設定可能な点で，分散ラベルモデルはプライバシを考慮した情報フロー制御を実現するモデルであるといえる．文献 2) では，Java を拡張した Jif（Java Information Flow）と呼ばれる分散ラベルを適用した新しいプログラミング言語を提案している．しかし，Jif のような新しいプログラミング言語を導入する手法は，既存のアプリケーションへの適用が困難である．

P3P[16]は，データの利用と管理方法を定めたプライバシポリシをデータ提供者とデータ管理者の間で共有する技術である．データ提供者は，データ管理者から提示されたプライバシポリシに同意できる場合はデータを提供し，同意できない場合はデータの提供を拒否することができる．これにより，プライバシポリシの共有は可能となるが，P3P は，データ管理者から提示されたプライバシポリシに従ってデータが管理されることを保証するものとはなっていない．

3. *Salvia* におけるデータ保護方式

3.1 解決すべき課題

前章までの議論をふまえたうえで，従来のアクセス制御方式では適さないプライバシデータの保護方式を実現するために解決すべき課題を次に示す．

- プライバシを考慮したデータ保護方式の実現
  一般的に，プライバシデータは，その提供者と収集者との間でデータの取扱い方法に関して同意したうえで，データ提供者から提供される．すなわち，プライバシデータはデータ提供者の意思により，提供されたデータごとに利用目的や提供範囲が異なる特徴を持つ．このため，データごとに異なるアクセス制御を適用可能とする必要がある．既存のアクセス制御方式は，データごとにポリシが設定可能なものとはなっていない．本論文では，プライバシ保護を「データ提供者の意思を反映したプライバシデータの管理」と定義する．また，データごとに保護ポリシを設定可能とし，それに

基づくプライバシを考慮したデータ保護方式を提案する．

- コンテキストに適応可能なデータ保護方式の実現
  既存のデータ保護方式ではプロセスに課す制約が厳密過ぎる問題や，プロセスがデータ漏洩を発生させる危険性に基づくアクセス制御が実現されないという問題がある．本論文では，プライバシデータへのアクセスが発生した際に，コンテキストに適応してその可否を制御することにより，上記の問題を解決するデータ保護方式を提案する．
- 既存のアプリケーションに透過的に適用可能なデータ保護方式の実現
  ソフトウェアによるデータ保護を実現する手法には，アプリケーションレベルでデータ保護機能を実現する手法と，OS レベルでアクセス制御を実現する手法がある．アプリケーションレベルで実現する場合，個々のアプリケーションがデータ保護機能を実装するか，データ保護機能を提供する共有ライブラリを適用する必要があるが，いずれの手法も個々のアプリケーションで対応する必要があるため，既存のアプリケーションに透過的にデータ保護機能を適用することができない．本論文では，OS レベルでデータ保護機能を適用し，既存のアプリケーションを変更することなく透過的に適用可能なデータ保護方式を提案する．

3.2 プライバシデータの保護モデル

Salvia では，プライバシ保護の特徴であるデータ提供者の意思を反映したプライバシデータの管理を実現するため，P3P と同様にデータの提供者と管理者の合意に基づいてデータ保護ポリシを作成する．具体的には，データ管理者から提示されたファイルの操作方法とコンテキストに基づいてポリシを作成し，それをデータ管理者に提供する．これらの処理を P3P におけるプライバシポリシの提示と同様の処理方式により実現する．また，プライバシデータの特徴であるデータごとに利用目的や提供可能な範囲が異なる点に着目し，プライバシデータを単位として保護ポリシを設定する．特に，*Salvia* では，OS はファイルを単位としてデータを管理する点に着目し，プライバシデータを含むファイルごとにデータ保護ポリシを設定する．すなわち，ファイルとデータ保護ポリシを組にして管理し，データ保護ポリシも保護の対象とする．OS レベルでプライバシ保護を実現することにより，*Salvia* 上で動作するすべてのアプリケーションに対して透過的にデータ保護方式を適用することができる．さらに，MAC と比較して柔軟なアクセス制御を実現するため，

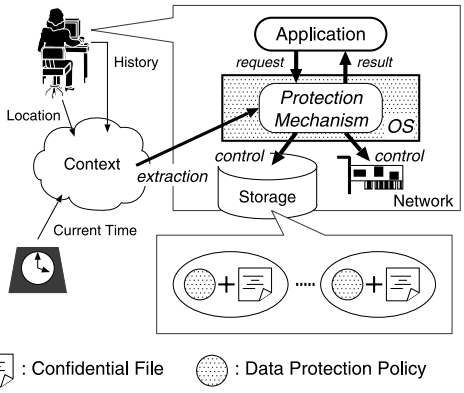


図 1 プライバシデータの保護モデル
Fig. 1 Protection model of privacy data.

ファイル，パイプ，ソケット，子プロセスといった計算機資源に対するアクセス要求が発生した際のコンテキストに適応して，その可否を制御する．計算機資源に対するアクセスはシステムコールを介して行われるため，*Salvia* ではシステムコールの実行の可否をデータ保護ポリシとコンテキストに基づいて判定する（**図 1** 参照）．本手法の詳細は 3.4 節で述べる．

Salvia は，上記の保護モデルに基づき，プライバシ保護に適したデータ保護機能を実現する．また，この保護モデルは，MAC と併用するものではなく，MAC を置き換えるものとして *Salvia* で実現する．しかし，この保護モデルにおいて，あるファイルを読み出し，そのデータを送信することが適切な処理なのか，あるいはデータ漏洩なのかをデータ保護ポリシに記述することはできない．このため，データ保護ポリシは，パケットファイアウォールのフィルタリングルールと同様に，原則的にはアクセスを禁止するポリシを記述し，許可すべきアクセスについてのみ，その制御条件を記述するといった運用方針をデータ管理者が決定する必要がある．

3.3 コンテキストとデータ保護ポリシ

Salvia は，プライバシデータへの正当なアクセス権限や管理者権限を持つユーザによる不正アクセスを制御することにより，その漏洩を防ぐ．これを実現するために，*Salvia* は前節で述べたプライバシデータの保護モデルに基づき，プライバシデータを含むファイルを単位としてデータ保護ポリシを設定する．データ保護ポリシは，ファイルにアクセスを試みたプロセスに課すアクセス制御として，プロセスに対して実行を許可，禁止するシステムコールと，それを制御するための条件としてコンテキストを記述可能としている．コンテキストは，プロセスや計算機の状況を示す情報

であり，その取得方法により内部コンテキストと外部コンテキストに分類される．コンテキストは，システムコールが発行された際に，その実行の可否を判定するために *Salvia* のデータ保護機構が利用する．

3.3.1 内部コンテキスト

内部コンテキストは，OS 内部で取得可能なプロセスの属性値とシステムコールの履歴がある．プロセスの属性値は，プロセスの所有者などを示す状態で，コンテキストとして実ユーザ ID，実効ユーザ ID，プロセス ID がある．これらは，プロセスを管理するためのデータ構造であるプロセス構造体から取得可能である．実ユーザ ID はシステムコールを発行したプロセスの所有者を，実効ユーザ ID はどのユーザの権限で実行されているプロセスかを特定するために使用する．すなわち，特定のユーザによって起動されるプロセスを制御対象とする場合，実ユーザ ID や実効ユーザ ID を用いてデータ保護ポリシを記述する．実ユーザ ID や実効ユーザ ID は，既存の OS で利用可能なパラメータである．したがって，これらを用いてプロセスを特定する手法は，*Salvia* を他の OS へ移植する場合に実装を変更する必要がないという利点がある．

システムコールの履歴はプロセスの動作履歴を示す状態で，システムコール番号，引数，返り値がある．引数がポインタの場合は，プロセスが所有するユーザ領域からデータを取得することも可能である．システムコール番号は，発行されたシステムコールの種類を特定するために使用する．システムコールの引数は，引数の内容に応じてシステムコールを制御するため，データ保護ポリシの制御条件に記述することができる．システムコールの返り値は，システムコールの実行の可否，ファイルディスクリプタなどの識別子の記録として使用する．システムコールの履歴は，時系列データとしてプロセスごとに記録される．

3.3.2 外部コンテキスト

外部コンテキストは，周辺機器やネットワーク経由で得られる状態で，計算機の位置，絶対時刻，相対時刻がある．*Salvia* では，位置を示すコンテキストとして無線 LAN アクセスポイントの ESSID と電波強度，GPS（Global Positioning System）の測位データが利用可能である．ESSID，電波強度，GPS の測位データは，デバイスドライバを通じてハードウェアから取得可能である．位置を示すコンテキストとして ESSID と電波強度を用いる場合，ESSID からアクセスポイントを特定し，その電波強度から当該アクセスポイントと計算機の距離を推定することができる．GPS の測位データを利用する場合，緯度と経度を示すパラメータ値から現在位置を特定することができる．しかし，GPS の測位データをカーネル内で取得する機能が未実装のため，現在の *Salvia* では GPS の測位データをコンテキストとして利用できない．

また，無線 LAN の電波は，干渉や減衰の影響により電波強度にゆらぎが生じる．電波強度を安定化させることは不可能であるため，現在の *Salvia* では，電波強度のゆらぎによりアクセス制御が安定しない場合がある．この問題は，*Salvia* に位置情報を管理する機能を追加することにより解決することを現在検討している．*Salvia* において本機能は未実装であるが，本機能により，無線 LAN 以外に利用可能なデバイスから位置に関する情報を取得し，位置情報の精度を高めることが可能となる．さらに，位置コンテキストの表現を抽象化することも可能となる．

絶対時刻は，現在の時刻を示すコンテキストで，RTC から取得可能である．相対時刻は，ある絶対時刻を起点（時刻ゼロ）とする時刻を示すコンテキストで，絶対時刻から算出可能である．絶対時刻と相対時刻は，システムコールの実行の可否を判定する条件として，データ保護ポリシに記述することができる．また，絶対時刻は，システムコールの履歴の 1 つであるシステムコールが発行された時刻の記録にも用いる．

以上で述べた外部コンテキストに着目することにより，1 章で述べた具体例のうち（2）の事例を防ぐことが可能となる．すなわち，*Salvia* では，アクセス制御の条件を位置や時刻に適応可能とすることにより，従来の MAC などのアクセス制御手法では実現されていない「適切な場所・時刻の場合のみアクセスを許可する」といった制御を実現可能としている．

3.3.3 制御対象となるシステムコールの分類

Salvia では，データ漏洩を防ぐ視点から制御対象となるシステムコールを次のように分類する（**表 1** 参照）．

- **restrict** クラス
  データの読み出し，書き込み，通信など，データ漏洩の要因となるシステムコールが属する．これらのシステムコールは，データ保護ポリシとコンテキストに基づき，その実行の可否を制御する．かつ，制御結果をコンテキストとして利用するため，システムコールの履歴を記録する．
- **context** クラス
  コンテキストとして参照するパラメータ値を変更するシステムコールが属する．パラメータ値が変更されることにより，アクセス制限が課されなくなる可能性があるため，これらのシステムコー

表 1 システムコールの分類
Table 1 System call classification.

class	system calls
restrict	read, write, readv, writev, pread64, pwrite64, mmap, mmap2, munmap, sendfile, sendfile64, remap_file_pages, readahead, swapon, swapoff, socketcall, ipc, execve, ptrace, mq_timedreceive, mq_timedsend, init_module, delete_module
context	ioctl, nfsservctl, mremap, setuid16, setgid16, setpgid16, setreuid16, setregid16, setgroups16, setfsuid16, setfsgid16, setresuid16, setresgid16, setreuid16, setregid, setgroups, setresuid, setresgid, setuid, setgid, setfsuid, setfsgid, settimeofday, stime, adjtimex, clock_settime
manage	open, close, socket, pipe, fork, vfork, clone, exit, kill, tkill, tgkill, reboot, exit_group, dup, dup2, mknod, fcntl, fcntl64, iopl, ioperm, mq_open, mq_unlink

表 2 restrict クラスのシステムコールのグループ化
Table 2 System call groups of restrict class.

group	system calls
read	read, readv, pread64, mmap, mmap2, munmap, readahead
write	write, writev, pwrite64, sendfile, sendfile64, mmap, mmap2, munmap
send_local	write, writev, sendfile, sendfile64, mmap, mmap2, socketcall, ipc, mq_timedsend
send_remote	write, writev, sendfile, sendfile64, socketcall

ルの履歴を記録し，パラメータ値の不正な変更を防ぐ．

- **manage** クラス
  システムコールの発行履歴を管理するための前処理や後処理を実行する契機となるシステムコールが属する．これらのシステムコールは，データ漏洩を引き起こすシステムコールではないため，その実行の可否を制御しない．ただし，システムコールの履歴を管理するデータ構造を操作する必要があるため，これらのシステムコールを *Salvia* の監視対象とする．

データ保護ポリシには，表 1 の分類に基づき，データ漏洩を引き起こすシステムコールの実行の可否を判定する制御条件をポリシとして記述する．しかし，個々のシステムコールに対するポリシを記述することは，ポリシの数が多くなるため，抜けや誤りといった記述ミスが起こる可能性がある．そこで，システムコールの処理内容やデータが伝搬する範囲に着目し，システムコールを次のグループに分類する（**表 2** 参照）．

- **read** グループ
  ファイルからデータを読み出すシステムコール群．
- **write** グループ
  ファイルにデータを書き込むシステムコール群．
- **send_local** グループ
  同一計算機上の他のプロセスのデータ領域にデータを書き込むシステムコール群．
- **send_remote** グループ
  他の計算機上のプロセスのデータ領域にデータを書き込むシステムコール群．

3.3.4 データ保護ポリシの記述

以上に基づいてデータ保護ポリシを記述することにより，簡易なポリシの記述と柔軟なアクセス制御を実現する．データ保護ポリシには，表 2 のそれぞれのグループに対してシステムコールの実行の可否を制御するアクセス制御ポリシを記述する．アクセス制御ポリシは，表 2 のグループ名，制御条件，制御方法の組で表現し，1 つの組をルールと呼ぶ．すなわち，アクセス制御ポリシは，1 つ以上のルールから構成される．制御条件は，システムコールの実行の可否を判定するための条件をコンテキストを用いて記述する．たとえば，特定のユーザ権限で動作するプロセスのみデータの書き込みを許可する場合，そのユーザを示す実効ユーザ ID を記述する．また，位置や時刻に適応したアクセス制御を実現する場合，位置を示すパラメータ値や時刻を記述する．制御方法は，システムコールの制御方法を指定するパラメータで，「すべて許可」「読み出し許可」「書き込み許可」「データの読み出し後，他のファイル・パイプ・ソケットへのデータ書き込み禁止」の 4 つが指定可能である．

また，データ保護ポリシには，デフォルトのポリシを記述することができる．デフォルトのポリシは，システムコールが発行された際のコンテキストが，アクセス制御ポリシの制御条件に一致しなかった場合に課するアクセス制御を指定する．この制御方法は，アクセス制御ポリシと同様の 4 つの制御方法を記述することができる．また，デフォルトのポリシが存在しない場合，アクセス制御ポリシにより許可されたアクセスを除くすべてのアクセス要求を禁止する．

データ保護ポリシの記述形式の例を**図 2** に示す．図 2

default : ***all*** : *deny;*

uid : 1000 && *location* : *ESSID* : solnet
+ *link* : 75 && time: from 8:00 to 20:00
: ***read, send_local*** : *allow;*

uid : 1000 : ***all*** : *deny;*

図 2 データ保護ポリシの記述形式
Fig. 2 Format of data protection policies.

は，1 章で示した事例のうち，計算機の持ち出しに起因するデータ漏洩を防ぐためのデータ保護ポリシの記述例である．図 2 の 1 行目は，デフォルトのポリシを設定する．この場合，2 行目以下のポリシに一致しないすべてのアクセスを禁止する．*all* とは，表 2 で示した 4 つのグループの総称である．2 行目は，**read** グループと **send_local** グループに属するシステムコールによるアクセスを許可するためのアクセス制御ポリシである．このポリシは，当該ファイルへのアクセスが発生した際のコンテキストについて，実ユーザ ID が 1,000，ESSID が solnet のアクセスポイントの電波強度が 75 以上，時刻が 8 時から 20 時の間である場合，当該ファイルの読み出しと，ファイル読み出し後に同一計算機上で動作するプロセスとの通信を許可する．最終行は，実ユーザ ID が 1,000 のユーザによるアクセス要求もデフォルトではすべて禁止することを指定するためのポリシである．

以上のデータ保護ポリシを設定することにより，*Salvia* は位置情報と時刻によってファイルへのアクセスを制御する．このため，計算機が不正に外部へ持ち出された場合，図 2 の 2 行目のポリシに反するため，ファイルを読み出すことができない．また，MAC では，図 2 の 2 行目のようにファイルを読み出した後に実行可能なアクセスを制限するポリシを記述することができない．

データ保護ポリシは，3.2 節で述べたように，データ管理者がデータ提供者に提示するファイルの操作方法とコンテキストを用いてデータ提供者が記述する．その際，制御対象となるシステムコールの種類や，実ユーザ ID，ESSID などのコンテキストをデータ提供者がデータ保護ポリシとして直接記述することは困難であると考えられる．そこで，データ提供者に対してデータ保護ポリシの記述を支援するポリシエディタの提供を現在検討している．このポリシエディタは，データ管理者によって簡易な表現に変換されたファイルの操作方法やコンテキストをデータ提供者に提示する．具体的には，ファイルの操作方法を示すシステムコールの場合，「読み出し」「書き込み」といった表現を提示し，時刻に関する条件の場合，「業務時間中」といった表現を提示する．これにより，データ提供者は簡易な表現を用いたデータ保護ポリシの記述が可能となる．また，データ管理者にとって具体的な位置情報が機密情報に該当する場合も，「社内」「オペレーションルーム」といった表現を提示することにより，データ提供者に対して位置情報を隠蔽しつつデータ保護ポリシを記述することが可能となる．

3.4 コンテキストに適応したシステムコール制御

Salvia はデータ保護ポリシに基づき，コンテキストに適応したシステムコール制御を実現する．一方，プロセスは `open` システムコールによりファイルアクセスを開始する．`open` システムコールは，その呼び出しが発生する要因となったファイルアクセスがユーザの誤操作や不正行為であるか否かにかかわらず発行される．このため，*Salvia* は，`open` システムコールが発行された際に，システムコールの前処理としてデータ保護ポリシを読み出す．すなわち，プロセスは，データ保護ポリシによって保護されたファイルに対する `open` システムコールの発行を契機として，*Salvia* のアクセス制御の対象となる．これにより，*Salvia* は，ユーザの誤操作，不正行為，ファイルのアクセス権限を持つユーザによって実行されるプロセスに対するアクセス制御を実現可能としている．このことを具体例をあげて述べる．1 章で述べた事例の（1）のようにユーザが電子メールにファイルを誤って添付した場合，*Salvia* によってそのメーラの TCP ソケットへの書き込みアクセスが制御される．また，ファイルやソケットへのアクセス権限を有するロールを不正に利用してファイルの送信を試みる場合も，ファイルアクセスの有無によってそのユーザが起動したプロセスのアクセスを制御する．コンピュータウィルスによって正常に動作しないアプリケーションのように，プログラムの信頼性が損なわれている場合も，上記の例と同様に *Salvia* によってアクセスが制御される．

データ保護ポリシの読み出し処理において，データ保護ポリシに記述されているアクセス制御ポリシのうち，システムコールを発行したプロセスの実ユーザ ID や実効ユーザ ID に一致するアクセス制御ポリシが存在しない場合，デフォルトのポリシに基づいて当該プロセスに課する制約を決定する．たとえば，デフォルトのポリシにより **read**，**write** グループのシステムコールが禁止されている場合，そのグループに含まれるシステムコールの実行を禁止する．また，実ユーザ ID や実効ユーザ ID が一致し，時刻や位置といった外

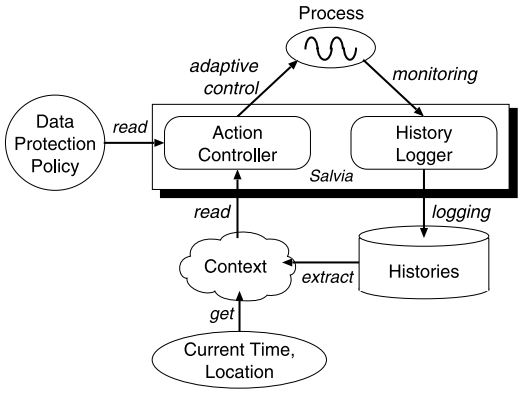


図 3 システムコール履歴に基づくアクセス制御
Fig. 3 Access control based on system call histories.

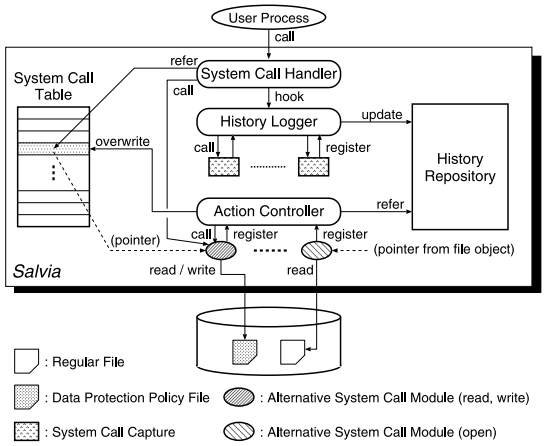
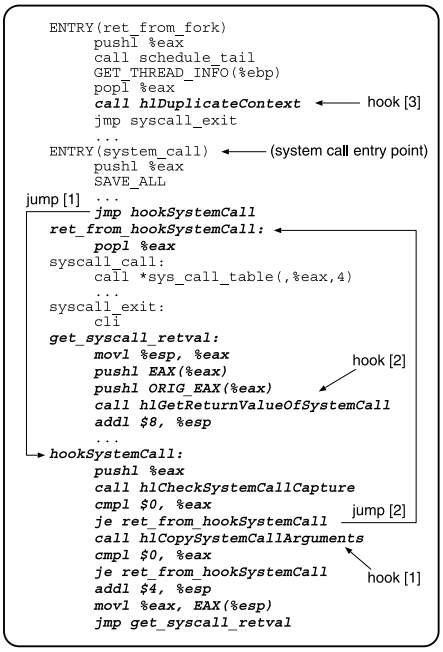


図 4 *Salvia* のデータ保護機構の全体構成
Fig. 4 Structure of data protection mechanism of *Salvia*.



(roman font): Linux original source codes
(italic font): Source codes for system call hook

図 5 システムコールハンドラ
Fig. 5 System call handler.

部コンテキストが制御条件に含まれている場合，システムコールが発行されるごとに外部コンテキストを取得し，システムコールの実行の可否を判定する必要がある．このため，*Salvia* では，当該ファイルへのアクセスを監視対象として *Salvia* のデータ保護機構に登録する．

また，データ保護ポリシとして「データ読み出し後，他のファイル・パイプ・ソケットへのデータ書き込み禁止」が設定されているファイルをプロセスが読み出した場合，*Salvia* は内部コンテキストの 1 つであるシステムコールの履歴を参照する．履歴を参照することにより，プロセスが現在利用可能なすべてのファイル・パイプ・ソケットに対するデータの書き込みアクセスを制御可能としている．

以上のように，*Salvia* のデータ保護方式は，システムコールを制御する際にコンテキストを参照する．特に，システムコールの履歴を参照することは，プロセスの過去の動作が現在の動作に影響を及ぼすことを意味する．すなわち，データ保護ポリシによって保護されたファイルにアクセスしたプロセスは，そのポリシに基づき，プロセスが終了するまでの間にわたって制約を受ける（**図 3** 参照）．

4. 実　　装

4.1 全 体 構 成

現在 *Salvia* は，プロトタイピングとして Linux カーネル 2.6.6 を基に *Salvia* のデータ保護機構を実装している．データ保護機構の構成を**図 4** に示す．データ保護機構は，システムコールの履歴を記録する History Logger，システムコールの履歴を時系列データとして格納する History Repository，コンテキストに適応してシステムコールの実行の可否を制御する Action Controller から構成される．これらは，プロセスが発行するシステムコールを契機として動作する．

4.2 システムコールの履歴の取得

History Logger は，図 4 に示すように，カーネル内に静的に組み込むモジュールである．History Logger は，履歴を取得する対象となるシステムコールがプロセスから発行された際に，システムコールハンドラに挿入したシステムコールフック関数の呼び出しによって起動される（**図 5** 参照）．History Logger で履

歴を取得するシステムコールは，表 1 で示している．History Logger は，システムコールが発行された時刻とプロセスの属性値を取得する．システムコールの引数と返り値はシステムコールによって型や数が異なるため，これらを取得するためのモジュール（System Call Capture）をシステムコールごとに用意している．System Call Capture は LKM（Loadable Kernel Module）で実現している．システムコールの履歴を取得する処理の流れを以下に示す．

(1) System Call Capture をロードする際に，インタフェースとなる関数のポインタを History Logger に登録する．
(2) 当該システムコールに対応する System Call Capture が登録されているか否か検査する．また，プロセスが *Salvia* のアクセス制御対象か否か検査する（図 5 の jump [1]）．
(3) (2) のいずれにも条件一致しない場合，このシステムコールの履歴を記録する必要がないため，History Logger を呼び出さずに通常のシステムコール処理を実行する（図 5 の jump [2]）．
(4) `hlCopySystemCallArguments` 関数の呼び出しにより History Logger に制御を移行する（図 5 の hook [1]）．
(5) History Logger と System Call Capture で，時刻，システムコールの種類，引数，プロセスの属性値を取得し，History Repository に記録する．
(6) システムコールハンドラに制御を戻し，システムコール処理を継続する．コンテキストに適応したシステムコールの制御を実現する処理は，4.4 節で述べる．
(7) システムコールの後処理をフックし，システムコールの返り値を取得する．その値を当該システムコールの履歴として履歴リポジトリに追記する（図 5 の hook [2]）．

プロセスが子プロセスを生成する場合，*Salvia* は，親プロセスに課した制約を子プロセスにも課する．このため，親プロセスのコンテキストを子プロセスに継承させる．これを実現するため，`fork` システムコールの後処理にフック関数を挿入し，コンテキストの継承を行っている（図 5 の hook [3]）．

4.3 システムコールの履歴の管理

History Repository は，History Logger が取得したシステムコールの履歴を時系列データとしてプロセスごとに管理する．Action Controller は，システムコールの実行の可否を判定する際，History Repository で管理しているコンテキストを参照する．この参照は頻繁に発生する可能性が高く，またデータ保護ポリシに記述されている制御条件に一致する履歴を検索，抽出する必要がある．このため，プロセス ID をハッシュ値とするハッシュ・チェイン法を用いて履歴を管理することにより，履歴へのアクセスにかかるコストを小さくする．

```
/*  Declaration of data structure(s)  */
typedef struct _SysCallLog_t SysCallLog;
typedef struct _SysCallCtrl_t SysCallCtrl;

typedef struct _HDList_t {
    struct _HDList_t *next;
    struct _HDList_t *prev;
} HDList;

typedef struct _HistoryList_t {
    struct _HistoryList_t *next;
    struct _HistoryList_t *prev;
    SysCallLog            *log;
    SysCallCtrl           *ctrl;
    unsigned int           status;
} HistoryList;

typedef struct _HistoryDescriptor_t {
    unsigned int           desc_id;
    struct _HDList_t       hd_list;
    struct _HistoryList_t *history;
    unsigned int           count;
} HistoryDescriptor;

/*  Hash table  */
HDList HistoryDescriptorTable[1024];
```

図 6 システムコールの履歴を管理するデータ構造
Fig. 6 Data structure for logging system call histories.

システムコールの履歴を管理するための主なデータ構造を**図 6** に示す．ハッシュテーブルの大きさは，Linux カーネルにおけるプロセス構造体を管理するハッシュテーブルの大きさが 1,024 となっていることから，その値を参考にした．また，同じハッシュ値を持つプロセスは，`HistoryDescriptor` 構造体を双方向リストとして管理している．この手法も，Linux カーネルにおけるプロセス構造体の管理手法と同様のものとなっている．システムコールの履歴は，`HistoryList` 構造体のメンバである `SysCallLog` 構造体で管理する．また，`HistoryList` 構造体を双方向リストとして管理し，最新の履歴をリストの先頭に挿入する．これにより，システムコールの履歴を時系列データとして管理している．

`fork` システムコールにおいて，親プロセスのシステムコールの履歴を子プロセスに継承する処理手順を次に示す．

(1) 親プロセスのシステムコールの履歴を管理する `HistoryDescriptor` 構造体へのポインタを取得する．
(2) 子プロセス用の `HistoryDescriptor` 構造体のメモリを確保する．

(3) 親プロセスの HistoryList 構造体を走査し，SysCallLog 構造体と SysCallCtrl 構造体の参照カウンタを 1 増加させる．
(4) 子プロセス用に HistoryList 構造体のメモリを確保し，SysCallLog 構造体と SysCallCtrl 構造体へのポインタを HistoryList 構造体のメンバ（log，ctrl）にコピーする．
(5) 親プロセスのシステムコール履歴の記録数だけ (3)，(4) を繰り返す．

4.4 システムコールの制御

Action Controller は，データ保護ポリシの読み出しと解釈，システムコールの実行の可否の制御を行うモジュールである．このうち，システムコールの制御は，その引数のデータ構造を理解する必要があるため，システムコールごとに LKM で実装した制御モジュール（Alternative System Call Module）で行う．このモジュールによって既存のシステムコールを置き換えることにより，データ保護ポリシとコンテキストに適応したシステムコールの制御を実現する．open システムコールにおける Action Controller の処理の流れを次に示す．

(1) open システムコールを置き換える制御モジュールをロードする際，Action Controller はシステムコールテーブル中の open システムコールへのポインタを保持しているエントリを当該制御モジュールに実装した open を置き換える関数へのポインタに変更する（図 4 参照）．
(2) History Logger の処理を終えた後，システムコールハンドラに制御が戻り，システムコールテーブルを介して (1) で置き換えられた open の代替関数が呼び出される．
(3) open の代替関数において，アクセス要求のあったファイルと組となるデータ保護ポリシの有無を検索し，存在する場合はその読み出しと解釈を実行する．
(4) 当該プロセスが別のファイルアクセス要求によってアクセス制御の対象となっているか否かを確認する．
(5) (3)，(4) のいずれかによって制御が必要であると判定された場合，アクセス制御の対象となる関数へのポインタを変更する．
(6) 通常の open システムコールの処理を完了させ，プロセスに制御を戻す．

(5) の処理において制御の対象となるファイルへのアクセスは，システムコールから呼び出されるファイル操作関数の実行を制御することにより実現する．ファイル操作関数は，open システムコールにおいて，ファイルオブジェクトを生成する際にファイル操作関数へのポインタがオブジェクトのメンバとしてコピーされ，そのポインタを介して関数が呼び出される．このため，(5) において，アクセス制御の対象となるファイルに関連付けられたファイルオブジェクトが持つ，読み出しや書き込み用のファイル操作関数へのポインタを変更し，制御用の代替関数へのポインタに置き換える．代替関数では，データ保護ポリシとコンテキストに基づき，要求されたアクセスの実行の可否を制御する．また，アクセス制御の対象がパイプやソケットの場合，pipe，socket システムコールにより生成されるオブジェクトによってアクセス要求を処理するため，ファイルの場合と同様の手法によりデータ保護ポリシとコンテキストに基づくアクセス制御が適用可能である．

5. 機能評価

5.1 プロセスの動作履歴の取得・記録にかかるオーバヘッド

Salvia において，コンテキストを取得することによるオーバヘッドを計測するために，**表 3** に示す計算機を用いて実験を行った．実験では，*Salvia* におけるデータ保護ポリシを定義したファイルに対する open システムコールの処理時間を計測した．また，標準の Linux カーネル 2.6.6 で，同じ内容のファイルを用いて同様の計測を行った．具体的には，ファイルを 10 個用意し，それらに対して 1 回ずつ open システムコールを呼び出した．ただし，2 次記憶装置へのアクセスによる処理時間の影響を避けるため，あらかじめファイルをキャッシュさせて計測した．なお，この処理時間に Action Controller の処理時間は含まれない．

実験結果を**図 7** に示す．実験結果より，*Salvia* は標準 Linux と比較してファイル 1 のアクセスは 26.59 μs，ファイル 2 から 10 は最大で 10.35 μs 処理時間が増加している．ファイル 2 以降の 10.35 μs は，プロセスの動作履歴の取得と記録にかかった時間である．ファイル 1 の 25.69 μs は，それに加えて，プロセスの動作履歴を記録するために必要なデータ構造の初期化にかかった時間も含まれる．このため，標準 Linux と比

表 3 実験環境
Table 3 Experimental environment.

Machine	IBM PC / AT 互換機
CPU	Intel Pentium III 1.0 GHz
Memory	256 MB
HDD	Ultra ATA100 40 GB
File System	Reiser File System

表 4 記録されたシステムコールの履歴
Table 4 Histories of recorded system call.

file size	open	read		write		socket	connect	send	recvfrom	total	
		sock	other	sock	other					count	size
1 KB	29	10	62	46	41	5	5	4	4	206	7.42 KB
10 KB	30	9	33	208	36	5	5	4	4	334	11.49 KB
100 KB	31	8	47	1,825	47	5	5	4	4	1,976	62.80 KB

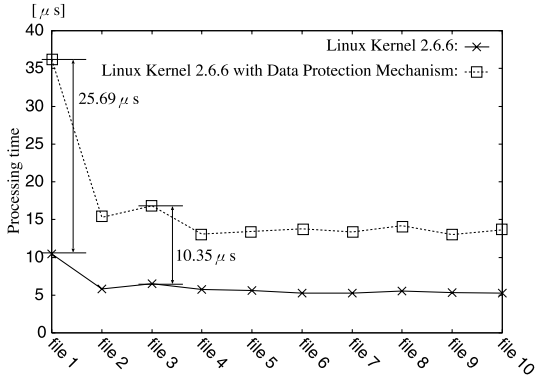


図 7 open システムコールの処理時間
Fig. 7 Processing time of open system call.

較して，初めて保護されたファイルにアクセスする場合では約 2.6 倍，2 回目以降は約 1.6 倍の処理時間がかかる．ただし，いずれも 10 μs オーダのオーバヘッドとなっている．

次に，1 章で述べた具体例の 1 つである「顧客情報ファイルが添付された電子メールの送信の防止」を例として，*Salvia* がシステムコールの履歴を記録するために消費するメモリサイズを評価する．実験では，GNU Emacs 上で動作するメーラ Wanderlust 2.10.1 を用いて，添付ファイルのファイルサイズが 1 KB，10 KB，100 KB の場合に *Salvia* が消費するメモリサイズと，*Salvia* によって記録されたシステムコールの履歴の個数を計測した．

実験結果を**表 4** に示す．表 4 から，添付ファイルのファイルサイズの増加に従ってソケットへの書き込み回数が増加し，*Salvia* によって記録されるシステムコールの履歴が増加している．また，本実験の場合，約 2,000 個のシステムコール履歴を記録するためにカーネルのメモリを 62.80 KB 消費していることが分かる．

Salvia では，プロセスが終了した場合，そのシステムコールの履歴を削除し，メモリを解放する．このため，本実験のような，プロセスの生存時間が短い場合，カーネル用の空きメモリが不足する状況にはならないと考えられる．しかし，システムコールの履歴の増加にともなってメモリ消費量が線形的に増加している．このため，HTTP サーバプロセスや FTP サーバプロセスなどの daemon プロセスによるシステムコールの履歴が増大した場合，空きメモリが不足する状況が想定される．この問題に対しては，増大したシステムコールの履歴の一部をディスクなどに移動させる手法を現在検討している．

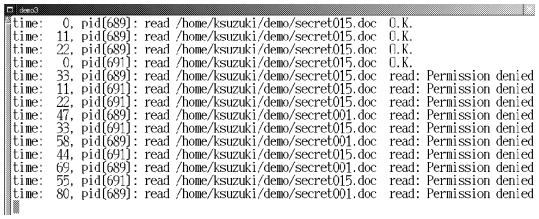


図 8 プログラムの出力結果（1）
Fig. 8 Output of the program (1).

5.2 データ保護ポリシとコンテキストに基づくシステムコールの適応的制御

コンテキストに適応してシステムコールの実行が制御されることを示すために，2 つの実験を行った．1 つ目は，以下に示す保護されたファイルを周期的に読み出すプログラムを実行した．

(1) ファイルには，「このファイルにアクセスしたプロセスは，30 秒経過した後は，すべてのファイルに対する入出力処理を禁止する」というデータ保護ポリシが定義されている．
(2) 上記のファイルに対して open システムコールを実行する．
(3) 上記のファイルに対して，read システムコールを 11 秒周期で 4 回発行する．
(4) (3) の繰返しの途中で fork システムコールを発行し，子プロセスを生成する．
(5) (3) の実行後，親プロセスは (3) を再開する．子プロセスは，(2) と (3) を実行する．
(6) 親プロセスは，さらに別のファイルに対して，(3) と同様の処理を実行する．

このプログラムの出力結果を**図 8** に示す．出力結果の時刻は，プロセスの実行が開始されてからの相対時刻を示す．プログラム開始後，時刻 22 までは，ファ

イルに対する read システムコールが許可されている．時刻 22 で，子プロセスが生成され，子プロセスは親プロセスと同じファイルに対して read システムコールを発行している．この read システムコールは許可されている．親プロセスは，時刻 33 において，データ保護ポリシの条件に一致したため，read システムコールが拒否されている．子プロセスは，時刻 11 において，read システムコールが拒否される．これは，子プロセスは親プロセスの動作履歴を継承しているため，ファイルに初めてアクセスしてから 30 秒経過しているものと判定されたことによる．さらに，親プロセスは，時刻 47 において，別のファイルに対して read システムコールを発行している．しかし，データ保護ポリシによって他のファイルへのアクセスが禁止されているため，この read システムコールも拒否される．以上のことから，*Salvia* において，子プロセスは親プロセスの動作履歴と保護ポリシを継承し，さらにコンテキストに適応したシステムコールの制御が実現されていることが分かる．

この実験で適用した相対時刻によるアクセス制御は，実験を短時間で終了させるために設定したポリシである．相対時刻によるアクセス制御が有効な例として，「一定時間経過した後にファイルを削除する」というポリシを設定する例が考えられる．現在の *Salvia* では，OS がデータ保護ポリシに従ってファイルを削除する機能は未実装となっているが，文献 17) において本機能について検討している．本機能の実現は，今後の課題である．

2 つ目の実験として，図 2 で示したデータ保護ポリシを持つファイルを cat コマンドで読み出す実験を行った．その際に，電波強度が異なる 2 つの場所でコマンドを実行した．ただし，この実験では，電波強度として Link Quality の値を用いた．実験結果を**図 9** に示す．データ保護ポリシの制御条件に基づき，Link Quality 値が 58 の場合はファイルの読み出しが禁止され，Link Quality 値が 90 の場合は読み出しが許可されている．

また，文献 17) では，コンテキストとしてユーザ ID，時刻，無線 LAN の ESSID と電波強度に着目した場合のファイルアクセス制御の性能を実験により示している．実験では，Intel Celeron プロセッサ 900 MHz，メモリ 256 MB の計算機を用いて，ファイルの読み出しと書き込みにかかる時間を計測している．その結果，ファイル読み出しのオーバヘッドが 390 μs 以内，書き込みのオーバヘッドが 490 μs 以内であることを示している．さらに，ESSID と電波強度を取得するために要した時間は 300 μs，時刻を取得するために要した時間は 0.8 μs であることを示している．

図 9 プログラムの出力結果（2）
Fig. 9 Output of the program (2).

6. おわりに

本論文では，プライバシデータの保護を目的とした OS *Salvia* におけるデータ保護モデル，設計，実装，評価について述べた．*Salvia* の特徴は，プライバシデータとデータ保護ポリシを組として管理する点，データ保護ポリシとコンテキストに基づいて適応的にシステムコールを制御する点，コンテキストの 1 つにプロセスが発行したシステムコールの履歴を取り入れた点があげられる．これにより，ユーザの誤操作，不正行為，プログラムの信頼性にかかわらずプライバシデータの漏洩を防止可能としている．また，機能評価では，プロセスの動作履歴の取得・記録によるオーバヘッドが 10.35 μs と，十数 μs オーダであることを示した．さらに，子プロセスの生成時にコンテキストを継承し，システムコールの制御が可能な点，外部コンテキストに基づくシステムコールの制御が可能な点を示した．今後の課題として，本論文中で述べた検討課題に加えて，プライバシデータが複数の計算機を伝搬していく場合に，データ保護ポリシを継承させる手法について検討する必要がある．

参考文献

1) 独立行政法人国民生活センター：個人情報流出事故に関する事業者調査結果 (2005). http://www.kokusen.go.jp/cgi-bin/byteserver.pl/pdf/n-20050325_1.pdf
2) Myers, A.C. and Liskov, B.: Protecting Privacy using the Decentralized Label Model, *ACM Trans. Software Engineering and Methodology* (*TOSEM*), Vol.9, No.4, pp.410–442

(2000).
3) Bell, D.E. and LaPadula, L.J.: Secure Computer System: Unified Exposition and Multics Interpretation, MTR-2997, MITRE Corporation; ESD-TR-75-306 (1976).
4) 品川高廣，河野健二，高橋雅彦，益田隆司：拡張コンポーネントのためのカーネルによる細粒度保護ドメインの実現，情報処理学会論文誌，Vol.40, No.6, pp.2596–2606 (1999).
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 (2002).
6) Bernaschi, M., Gabrielli, E. and Mancini, L.V.: REMUS: A Security-Enhanced Operating System, *ACM Trans. Information and System Security*, Vol.5, No.1, pp.36–61 (2002).
7) 榮樂恒太郎，新城　靖，板野肯三：システム・コールに対するラッパ/リファレンスモニタ SysGuard の設計と実装，情報処理学会論文誌，Vol.43, No.6, pp.1690–1701 (2002).
8) Edjlali, G., Acharya, A. and Chaudhary, V.: History-based Access Control for Mobile Code, *Proc. 5th ACM Conference on Computer and Communications Security*, pp.38–48 (1998).
9) Ferraiolo, D.F., Sandhu, R., Gavrila, S., Kuhn, D.R. and Chandramouli, R.: Proposed NIST standard for role-based access control, *ACM Trans. Information and System Security* (*TISSEC*), Vol.4, No.3, pp.224–274 (2001).
10) Boebert, W.E. and Kain, R.Y.: A Practical Alternative to Hierarchical Integrity Policies, *Proc. 8th National Computer Security Conference*, pp.18–27 (1985).
11) Department of Defense: Trusted Computer System Evaluation Criteria, DoD 5200.28-STD (1985).
http://csrc.nist.gov/publications/history/dod85.pdf
12) Organisation for Economic Co-operation and Development: OECD Guidelines on the Protection of Privacy and Transborder Flows of Personal Data (2004).
http://www.oecd.org/document/18/0,2340,en_2649_201185_1815186_1_1_1_1,00.html
13) The TrustedBSD Project: TrustedBSD.
http://www.trustedbsd.org/
14) Biba, K.J.: Integrity Considerations for Secure Computer System, MTR-3153, MITRE Corporation; ESD-TR-76-372 (1977).
15) National Security Agency: Security-Enhanced Linux. http://www.nsa.gov/selinux/
16) W3C: The Platform for Privacy Preferences 1.0 (P3P1.0) Specification (2002).
http://www.w3.org/TR/P3P/
17) 一柳淑美，鈴来和久，毛利公一，大久保英嗣：プライバシ保護を実現するオペレーティングシステムのファイルアクセス制御手法，情報処理学会研究報告 2005-OS-98, Vol.2005, No.16, pp.1–8 (2005).

付録　データ保護ポリシの管理

A.1　データ保護ポリシの記述項目

データ保護ポリシは，3.3.4 項でも述べたように，制御対象となるシステムコールのグループ名（表 2 参照），システムコールを制御する条件，システムコールの制御方法の組で表現する．*Salvia* では，この 3 つの組で表現されるデータ保護ポリシを次の記述形式で表現する．

<システムコールの制御条件>:<システムコールのグループ名>:<制御方法>

制御条件には，デフォルトの制御条件を示す呼称である `default` と，コンテキストを用いた制御条件を記述する．コンテキストを用いた制御条件は，コンテキストの種類を示すラベルと値を記述する．ラベルと値を区切るデリミタにはコロンを使用する．具体的には，以下のようなラベルと値の組で表現される．

- 実ユーザ ID，実効ユーザ ID
  実ユーザ ID のラベルは `uid`，実効ユーザ ID のラベルは `euid` で表現される．両者とも正の整数値を値に持つ（例 `euid:1000`）．
- 位置
  位置を示すラベルは `location` である．位置の場合，位置情報を取得するデバイスを指定するラベルとして `ESSID` と `GPS` を指定する．
  - 無線 LAN
    ラベル `ESSID` に続けて ESSID を表す文字列と，Link Quality 値を値に持つ．Link Quality 値は，接頭語`+link` に続けて正の整数値を記述する（例 `ESSID:solnet +link:75`）．
  - GPS
    ラベル `GPS` に続けて緯度・経度を示す値を記述する．緯度を示す値には接尾語に `N` または `S` を用いる．同様に，経度を示す値には接尾語に `E` または `W` を用いる．範囲を指定する場合はハイフンを用いる（例 `GPS: 34.47N, 135.45E - 35.42N, 136.27E`）．
- 時刻
  時刻を示すラベルは `time` である．このラベルに続けて時刻を記述する．絶対時刻は 24 時制で時

刻を記述し，相対時刻は 0 以上の整数値と相対時刻を表す接尾語+を記述する．また，相対時刻の単位は秒である（例 `time:300+`）．

また，コンテキストを用いた制御条件は，複数のコンテキストを論理積や論理和で評価したものを制御条件として記述することができる．論理和と論理積は，組となるコンテキストを `"&&"` または `"||"` で結合することにより評価される．

A.2 データ保護ポリシを設定するシステムコール

Salvia では，データ保護ポリシを管理するために，次に示すシステムコールを実装している．

```
int set_data_prot_policy(const char *path,
                         const void *dpp,
                         size_t size)
```

`set_data_prot_policy` は，*path* で指定されたファイルに対応するデータ保護ポリシを *dpp* で示されるメモリ領域から読み出す．その際，データ保護ポリシのデータサイズは *size* に指定されていなければならない．このシステムコールの実行は，root 権限を必要とする．さらに，正当なユーザによってこのシステムコールが呼び出されていることを確認するため，パスワードによるユーザ認証を行う．

現在の *Salvia* では，データ保護ポリシの読み出しが成功すると，カーネルはデータ保護ポリシをファイルとしてディスクに書き込む．ただし，*Salvia* は，このファイルに対する `open`，`read`，`write` などの通常のファイルアクセスを実現するシステムコールを失敗させる．これにより，テキストエディタなどの既存のアプリケーションによってデータ保護ポリシが変更されることを防ぐ．データ保護ポリシに対するアクセス制御は，他のファイルへのアクセスと同様に Action Controller で実現している．また，今後の課題として，Third Extended Filesystem（EXT3 FS）などで利用可能な拡張アトリビュートを利用したデータ保護ポリシの管理手法を検討している．

(平成 17 年 7 月 29 日受付)
(平成 17 年 11 月 11 日採録)

鈴来 和久（学生会員）

昭和 53 年生．平成 14 年立命館大学理工学部情報学科卒業．現在，同大学大学院理工学研究科一貫制博士課程フロンティア理工学専攻に在学中．オペレーティングシステム，動的 QoS 制御のためのミドルウェア，計算機ネットワーク，ユビキタスコンピューティングに関する研究に興味を持つ．

一柳 淑美（学生会員）

昭和 55 年生．平成 16 年立命館大学理工学部情報学科卒業．現在，同大学大学院理工学研究科博士課程前期課程情報システム学専攻に在学中．オペレーティングシステムとシステムソフトウェアに関する研究に興味を持つ．

毛利 公一（正会員）

昭和 47 年生．平成 11 年立命館大学大学院理工学研究科博士課程後期課程総合理工学専攻修了．同年東京農工大学工学部情報コミュニケーション工学科助手，平成 14 年立命館大学理工学部情報学科講師，平成 16 年同大学情報理工学部情報システム学科講師となり，現在に至る．工学博士．オペレーティングシステム，プライバシ保護技術，計算機ネットワーク等の研究に従事．電子情報通信学会，日本ソフトウェア科学会，ACM，IEEE-CS 各会員．

大久保英嗣（正会員）

昭和 26 年生．昭和 52 年北海道大学大学院工学研究科情報工学専攻修士課程修了．同年（株）日立製作所ソフトウェア工場に入所．主として FORTRAN コンパイラの開発に従事．昭和 54 年京都大学工学部情報工学科助手，昭和 60 年同講師，昭和 62 年同助教授，平成 3 年立命館大学理工学部情報学科教授，平成 16 年同大学情報理工学部情報システム学科教授となり，現在に至る．工学博士．オペレーティングシステム，データベースシステム，分散システム，実時間システム等の研究に従事．電子情報通信学会，日本ソフトウェア科学会，システム制御情報学会，ACM，IEEE-CS 各会員．