情報処理学会論文誌　Vol. 50　No. 9　1984–1996 (Sep. 2009)

プライバシアウェアOS *Salvia* における共有メモリアクセス制御手法

廣　　真文[†1,*1]　鈴来和久[†1,*2]　毛利公一[†2]

近年，プライバシ情報が漏洩する事件が頻発している．データ漏洩の要因の多くは，誤操作のような人為的なミスや，正当なアクセス権限を持つユーザの故意による操作である．これらに起因するデータ漏洩を防止するために，我々は，コンテキストに適応したプライバシ保護を実現するオペレーティングシステム *Salvia* を開発している．*Salvia* は，データの伝搬範囲を制限することを目的としており，それを実現するために，データの保護方法を記述したデータ保護ポリシとコンテキストに基づき，データ漏洩を引き起こす可能性のある計算機資源へのアクセスを制御する．データ漏洩を発生させる可能性のある計算機資源には，ファイル，ソケット，プロセス間通信のための資源などがある．本論文では，特に，共有メモリを利用したプロセス間通信に起因するデータ漏洩を防止する手法を提案する．提案手法は，ページフォルト例外を利用して共有メモリへのアクセスの可否を制御する．また，*Salvia* の特徴であるコンテキストに適応したアクセス制御方式を適用することにより，データ漏洩が発生する可能性のある共有メモリアクセスを選択的に制限することができる．本論文では，*Salvia* のメモリアクセス制御機構の設計について述べるとともに，Intel x86 アーキテクチャにおける実装手法と性能評価について述べる．

An Access Control Method for Shared Memory in Privacy-aware Operating System *Salvia*

Masafumi Hiro,[†1,*1] Kazuhisa Suzuki[†1,*2] and Koichi Mouri[†2]

Today, incidents of leakage of sensitive data such as privacy information have occurred frequently. In many cases, the factors of data leakage are as follows: malicious operations by a user who has an authority, and user's misoperations. In order to solve this problem, we have developed a privacy-aware operating system named *Salvia*. According to data protection policies associated with sensitive data, *Salvia* limits accesses to computation resources such as files, sockets, pipes, and resources that are used for interprocess communications. Thus, in *Salvia*, it is possible to restrict accesses to the resources that could serve as output channels for data leakage. Especially, we describe a method to prevent data leakage via shared memory access in interprocess communications. To restrict accesses to shared memory, *Salvia* determines permissions for each access to shared memory in the page fault exception handler. Moreover, *Salvia* employs a context-aware access control mechanism that is suitable for privacy protection. Therefore, *Salvia* can restrict accesses to shared memory that may cause to data leakage, selectively. In this paper, the design and implementation (on Intel x86 architecture) of *Salvia*'s access control mechanism for shared memory are described, and also we show the results of the performance evaluation using a micro-benchmark program.

1. はじめに

近年，プライバシ情報が電子化され，計算機で管理されている．たとえば，作業の効率化やネットワークを介したサービスの提供を目的として，多くの企業が顧客の個人情報を電子化し，計算機で管理している．このような顧客情報や機密情報といったプライバシ情報が，記憶媒体やネットワークを介して計算機から漏洩する事件が多発している．個人情報の漏洩は，ユーザのプライバシ侵害を引き起こすだけでなく，企業に対してもイメージダウンや補償問題による金銭的な負担を与える．2005 年 4 月 1 日から施行された個人情報保護法[1] は，経済協力開発機構（OECD）のプライバシ保護の原則[2] に基づき，企業に対して個人情報の利用と取扱いに関する義務や制限を定めており，個人情報の取扱いに対する意識が高まっている．しかし，依然としてデータ漏洩事故は発生している．

文献 3) は，個人情報の漏洩を引き起こす要因として，(1) 従業員などの不注意による流出・紛失，(2) 従業員などによる外部への持出し，(3) 盗難による流出，(4) システム上の問題による流出などをあげている．さらに，以上の要因の中で，(1) と (2) が発生要因の多くを占めることを示している．暗号化や認証といったセキュリティ技術は，外部からの攻撃に

†1 立命館大学大学院理工学研究科
Graduate School of Science and Engineering, Ritsumeikan University
†2 立命館大学情報理工学部
College of Information Science and Engineering, Ritsumeikan University
∗1 現在，株式会社日本総合研究所
Presently with The Japan Research Institute, Limited
∗2 現在，三菱電機株式会社
Presently with Mitsubishi Electric Corporation

対するデータ保護を目的としている．そのため，(1) や (2) のような人為的なミスや内部犯によるデータ漏洩を防止する技術を開発する必要がある．

以上の背景より，我々は，これらに起因するデータ漏洩を防止するための手法として，ユーザや計算機の状態や環境（以下，コンテキストと記す）に適応したプライバシ保護を実現するオペレーティングシステム *Salvia* [4] を開発している．*Salvia* では，データ提供者やデータ管理者の意図する保護方針をデータ保護ポリシとして定義し，保護を必要とするファイル（以下，保護ファイルと記す）と組にして管理する．*Salvia* は，保護ファイルにアクセスしたプロセスに対して，データ保護ポリシに基づきアクセス制御を課することにより，保護すべきデータの伝搬範囲を制限する．具体的には，ファイル，ソケット，パイプ，共有メモリなどの計算機資源に対するアクセスのうち，データ漏洩の危険性があるものを制御することにより，従来の OS より柔軟なプライバシ保護を実現している．

本論文では，特に，共有メモリに着目し，共有メモリを利用したプロセス間通信に起因するデータ漏洩を防止する手法を提案する．本手法は，ページフォルト例外を利用することにより，共有メモリへのアクセスを制御する．また，データ保護ポリシとコンテキストに基づき，データ漏洩が発生する可能性のある共有メモリアクセスを選択的に制限する．これにより，共有メモリに起因するデータ漏洩を防止する．本論文では，実用性の観点から，一般に広く利用されている Intel x86 アーキテクチャ上での実装例を示す．また，本手法を適用した際に生じるオーバヘッドを計測し，本手法の性能を評価する．

以下，2 章で解決すべき課題である共有メモリを介したデータ漏洩について述べ，3 章で共有メモリの制御手法について述べる．4 章で Intel x86 アーキテクチャにおける実装例を示し，5 章で提案手法の性能を評価する．また，6 章で関連研究について述べる．

2. 共有メモリを介したデータ漏洩

プロセスは，System V IPC で提供されている共有メモリ資源を利用することにより，複数のプロセス間でデータを共有することができる．共有メモリによるプロセス間通信は，データベースシステムをはじめとして，プロセス間でデータを共有するアプリケーションにおいて頻繁に利用される．これらのアプリケーションのように，複数のプロセスが共有メモリによりデータを共有する状況を想定する．たとえば，2 つのプロセス（プロセス A とプロセス B）が，同一の共有メモリをアタッチしている状況において，プロセス A が読み出しアクセスを許可された保護ファイルをオープンする．プロセス A は，このファイルに対する読み出し操作を許可されているため，保護ファイルの内容（以下，保護データと記す）を自らのプロセスアドレス空間に読み出し，共有メモリに書き込むことが可能である．一方，プロセス B は，プロセス A が書き込んだ保護データを共有メモリから読み出すことが可能である．また，プロセス B は，共有メモリから読み出した保護データを新たなファイルに書き込むことにより，保護ファイルを複製することも可能である．このように，複数のプロセス間で保護データが共有されることにより，データ漏洩が発生する危険性がある．

3. コンテキストに適応した共有メモリの制御

3.1 *Salvia* の概要

Salvia では，データ提供者やデータ管理者の意思を反映したデータ保護を実現するために，保護ファイルごとにデータ保護ポリシを定義できる．データ保護ポリシは，ext3 などのファイルシステムの inode に拡張属性として保存されており，保護ファイルと組にして管理される．*Salvia* は，プロセスがファイルをオープンすることによりファイルへのアクセスを開始する点に着目し，保護ファイルをオープンしたプロセスをアクセス制御の対象として認識する．ファイルやソケットといった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* はこれらのシステムコールの実行をデータ保護ポリシとコンテキストに基づいて制限する．*Salvia* で利用可能なコンテキストには，ユーザ ID，時刻，計算機の位置，送信先計算機の IP アドレス，プロセスの動作履歴などがある．

データ保護ポリシは，XML で記述され，コンテキストを用いて記述された制御条件に一致した場合のアクセス権限と，デフォルトのアクセス権限により構成される．データ保護ポリシの記述を容易とするために，*Salvia* では，データ漏洩を防止する観点からシステムコールを以下の 4 つのグループに分類し，それぞれのグループについてアクセス権限を記述する．

- 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.

2 章で述べたデータ漏洩を防止するためには，共有メモリによる保護データの共有の可否

```
<data_protection_policy>

  <default_access>
    <send_local>deny</send_local>
  </default_access>

  <data_protection_domain type="none">

    <ACL>
      <context>
        <location>
          <device id="net_radio">
            <value type="character">roomA</value>
            <value>30</value>
          </device>
        </location>
      </context>
      <access>
        <send_local>allow</send_local>
      </access>
    </ACL>

  </data_protection_domain>

</data_protection_policy>
```

図 1　データ保護ポリシの記述例
Fig. 1　Example of data protection policy.

を，send_local グループに対するポリシとして，データ保護ポリシに記述する．共有メモリによるデータの共有を制御するポリシの例を**図 1** に示す．このポリシは，基本的にはプロセス間通信を禁止するが，ESSID が roomA のアクセスポイントからの電波強度が 30 以上の場合は部屋 roomA 内にいると見なしてプロセス間通信を許可する例である．

Salvia のポリシ記述は，ファイルとシステムコールの関係を記述しているという点では Security-Enhanced Linux（以下，SELinux と記す）[5] で用いられている Type Enforcement（以下，TE と記す）[6] に似ている．TE では，ファイルなどの資源を示すオブジェクト，プログラムなどの主体を表すドメイン，どのような操作が可能であるかを示すアクセスベクタを細かく指定できる．しかし，一般的にはこれが逆に設定の困難さにつながっており，設定を容易にするための研究が行われている [7]．*Salvia* は，データの重要性や許される伝搬範囲はデータ自体の特性であり，それを制御することを目的としている．よって，ポリシの記述は下記に示すように比較的容易となるように設計されている．

- プロセスとの関連を記述する必要がない．
- データに対する操作の種類が上記の 4 種類と少ない．
- システムコールの種類と引数の内容からどの操作であるかを *Salvia* が自動判定してポリシを適用するため，利用者が細かいパラメータを記述しなくてもよい．

なお，inode の拡張属性として保存されているデータ保護ポリシは，拡張属性をアクセスするための既存のシステムコールである `getxattr` や `setxattr` ではアクセスできないようにしている．データ保護ポリシを取得・変更するためのシステムコールとして `get_salvia_policy` と `set_salvia_policy` を用意している．`set_salvia_policy` は，データ保護ポリシを変更する際にパスワード認証を行う機能を有しており，データ保護ポリシ自体も保護されている．実際には，データ保護ポリシを変更する場合は専用のアプリケーションを用いる．アプリケーションは，ユーザからのパスワード入力を受け付けた後，新しいデータ保護ポリシとパスワードを引数として `set_salvia_policy` を呼び出す．`set_salvia_policy` では，渡されたパスワードと，置き換える前のデータ保護ポリシ中に記されたパスワードとを比較し，一致した場合に新しいデータ保護ポリシを保存するという処理を行う．

3.2　共有メモリのアクセス制御モデル

3.2.1　書き込みの禁止によるデータ共有の制限

共有メモリを介したデータ漏洩を防止するためには，保護ファイルをオープンしたプロセスによる共有メモリへの書き込みや共有メモリの使用を禁止するか，他のプロセスによる共有メモリからの読み出しを禁止することにより，保護データの共有を制限する必要がある．他のプロセスによる共有メモリからの読み出しを制御する場合，プロセスに課される制限が厳しくなる．具体的には，保護ファイルにアクセスしているプロセスと同一の共有メモリをアタッチしているすべてのプロセスに対し，共有メモリからの読み出しを禁止する必要がある．そのため，データを読み出すことができるプロセスは，保護ファイルをオープンしたプロセスのみとなり，利便性が低下する．

一方，書き込みを制御する場合，保護ファイルをオープンしたプロセスに対し，共有メモリへの保護データの書き込みを制御する．保護データの共有を許可する場合は，共有メモリへの書き込みを許可し，禁止する場合は共有メモリへの書き込みを禁止する．これにより，共有メモリ上のデータが保護データではないことを保証することができる．そのため，当該共有メモリをアタッチしている他のプロセスは，共有メモリに対して自由にアクセスすることが可能である．以上より，*Salvia* では，後者の書き込みによる制御方式を採用した．

共有メモリに対する書き込みは，共有メモリのアタッチの後，当該メモリリージョンにメモリアクセスを行うことにより実現される．そのため，共有メモリのアタッチを禁止することにより，当該共有メモリへの書き込みを禁止することが可能である．しかし，この場合，

書き込みアクセスのみでなく，読み出しアクセスも同様に禁止される．また，保護データの共有の可否が共有メモリのアタッチ時に決定されるため，共有メモリによる保護データの共有が禁止される状況下であっても，共有メモリのアタッチがすでに許可されている場合，そのアクセスは許可されるという問題がある．したがって，コンテキストに基づいて共有メモリを制御するためには，メモリアクセス時にコンテキストを取得し，データ共有の可否を判断する必要がある．そこで，書き込みアクセスを制御対象とし，保護ファイルにアクセスしたプロセスが共有メモリに対して書き込みアクセスを行った際に，保護データの共有の可否を決定する．ただし，OS は，プロセスのローカルメモリ空間において，データが保護データであるか区別することができない．このため，データ共有を禁止する場合には，共有メモリに対する書き込みをすべて禁止する．

3.2.2　ページフォルト例外による書き込みの制御

共有メモリへのアクセスは，システムコールを介さずに CPU のインストラクションにより実行されるため，カーネルはこれを把握することができない．そのため，共有メモリへの書き込みアクセスを制御するためには，メモリアクセスをフックし，カーネルに処理を遷移させる必要がある．そこで，カーネルに処理を遷移させるために，ページフォルト例外を利用する．ページングによるメモリ管理の場合，ページフォルト例外を発生させることによりハンドラに処理が遷移するため，これをトリガとしてアクセス制御を実現する．本手法では，書き込みアクセスのみをフックするために，書き込み違反により，ページフォルト例外を発生させる．

共有メモリを介したデータ漏洩は，共有メモリのアタッチと保護ファイルのオープンの実行順序に関係なく発生する可能性があるため，両者が実行された時点でこれらの制御を開始する．プロセスが保護ファイルをオープンした後，共有メモリをアタッチした場合は，共有メモリのアタッチのための `ipc` システムコールが呼び出された際に，共有メモリに対応するページテーブルエントリ（以下，PTE と記す）を変更する．一方，共有メモリをアタッチした後，保護ファイルをオープンした場合は，保護ファイルのオープンをトリガとして，すでにアタッチしている共有メモリに対応する PTE を変更する．このために，共有メモリのアタッチが発行された場合に，その履歴を記録しておく．これにより，保護ファイルのオープンと共有メモリのアタッチの実行順序にかかわらず，共有メモリへの書き込みアクセスに対してページフォルト例外を発生させることができるため，ハンドラ内でアクセス制御の判定ができる．なお，デタッチされた共有メモリは制御対象とする必要がないため，共有メモリがデタッチされた場合には，当該共有メモリのアタッチに関する履歴を削除する．

3.2.3　コンテキストの変化にともなう可否の再判定

前項の処理を行った場合，1 度共有メモリへの書き込みを許可されたプロセスは，以後，当該共有メモリに対して自由に書き込みを行うことができる．しかし，コンテキストが変化した場合，保護データが漏洩する危険性も変化するため，再度書き込みの可否を判定する必要がある．そこで，着目すべきコンテキストの変化にともない，当該 PTE の書き込みビットをクリアする．これにより，コンテキストに適応した共有メモリへの書き込みアクセスの制御を実現する．

4. 実　　装

4.1 全 体 構 成

現在，*Salvia* は，Linux カーネルを基に Intel x86 アーキテクチャ上に実装している（**図 2** 参照）．*Salvia* のデータ保護機構は，システムコールの履歴を取得する 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たシステムコールの制御を行う Action Controller，データ保護ポリシの内容を格納する Policy List から構成される．

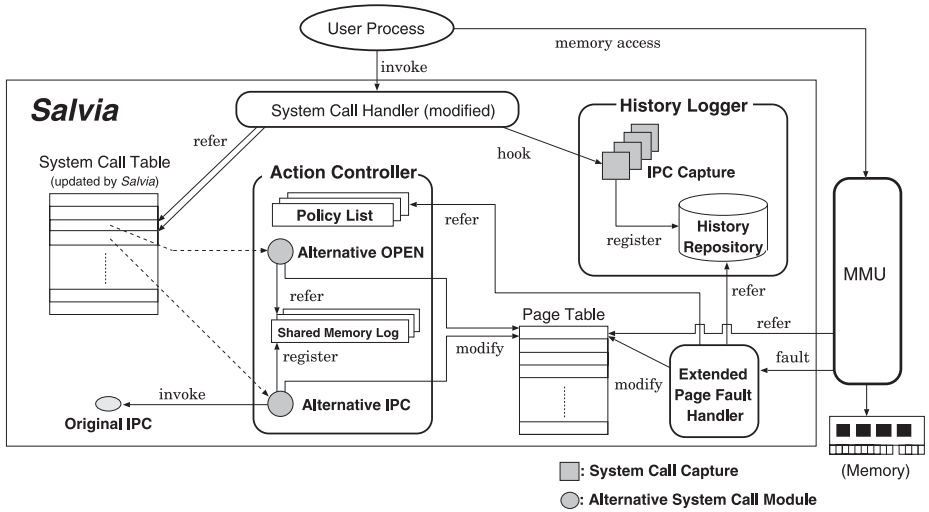


図 2　*Salvia* の全体構成
Fig. 2　Structure of data protection mechanism of *Salvia*.

History Logger は，`ioctl`，`mremap`，`setuid` などの履歴を取得する必要があるシステムコール（文献 4) 参照）が発行された際に，システムコールが発行された絶対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や返り値は，システムコールにより数やデータ構造が異なるため，履歴を取得すべきシステムコールごとに履歴取得用のモジュール（System Call Capture）を用意する．System Call Capture は，LKM（Loadable Kernel Module）で実現され，モジュールのロード時に History Logger に登録される．また，History Repository は，History Logger により取得されたシステムコールの履歴を時系列データとして保持する．システムコールの履歴は，保護ファイルのオープンをトリガとして取得を開始し，プロセスごとに管理する．

Action Controller は，保護ポリシに基づき，`read`，`write`，`socketcall` などの制御すべきシステムコール（文献 4) 参照）の実行を制御する．システムコールの制御には，システムコールの引数や返り値を利用する．しかし，引数や返り値のデータ構造はシステムコールにより異なるため，システムコールごとに実行を制御する機構が必要となる．そこで，*Salvia* では，システムコールごとに制御モジュール（Alternative System Call Module）を LKM で実装している．Action Controller は，制御モジュールがロードされた際に，システムコールテーブルの当該エントリのポインタを制御モジュールに変更することにより，制御モジュールに処理を移行させる．Policy List は，読み出したデータ保護ポリシを格納するリストである．*Salvia* では，保護ファイルのオープン時にデータ保護ポリシの内容を読み出し，Policy List に格納する．

これらの保護機構により，*Salvia* では，保護ファイルにアクセスしたプロセスに対してアクセス制御を課し，そのプロセスから各計算機資源へのアクセスを制限する．具体的には，以下の手順により，計算機資源へのアクセスを行うシステムコールの実行を制御する．

(1) プロセスが保護ファイルをオープンすると，`open` システムコールに対応する Alternative System Call Module に処理が移行する．このモジュールは，保護ファイルに対応するデータ保護ポリシを Policy List に読み出し，そのプロセスをアクセス制御の対象とする．
(2) アクセス制御の対象であるプロセスが，履歴を取得する必要があるシステムコールを発行すると，History Logger と System Call Capture がその履歴を取得し，History Repository に格納する．
(3) アクセス制御の対象であるプロセスが制御すべきシステムコールを発行すると，対応する Alternative System Call Module に処理が移行する．
(4) Alternative System Call Module は，位置，時刻，システムコールの履歴といったコンテキストを参照し，Policy List に格納されているデータ保護ポリシと比較することにより，システムコールを制御する．具体的にどのような制御を行うかはシステムコールにより異なり，たとえば `read` や `write` といったデータ漏洩の要因となるシステムコールについては，システムコールの実行の可否を決定する．

4.2　共有メモリ制御機構

Salvia において，共有メモリを制御するために，以下の機構を実装した（図 2 参照）．

- 共有メモリに関する履歴の取得機構
- 共有メモリに関するシステムコールの制御機構
- 共有メモリアクセスの制御機構

以下，それぞれの機構について述べる．

4.2.1　共有メモリに関する履歴の取得

共有メモリに関する履歴の取得機構として，`ipc` システムコールに対する System Call Capture（以下，IPC Capture と記す）を実装した．IPC Capture は，`ipc` システムコールの引数と返り値を取得する．これらの値は，History Logger が取得したコンテキスト（時刻やプロセスの属性値）とともに History Repository に格納される．

`ipc` システムコールは，共有メモリのほかに，セマフォやメッセージキューに対する処理も提供している．そのため，`ipc` システムコールの引数や返り値は，要求された処理ごとに数や意味が異なる．そこで，拡張性の観点から，IPC Capture では，`ipc` システムコールにより提供される処理別に引数と返り値を取得する関数を定義可能としている．`ipc` システムコールが発行されると，History Logger に処理が移行する．History Logger は，システムコール番号から ipc システムコールが呼び出されたことを特定し，IPC Capture を呼び出す．IPC Capture は，`ipc` システムコールの引数から要求された処理の種類を特定し，処理ごとに履歴取得関数を呼び出す．履歴取得関数は，`ipc` システムコールの引数を取得する．たとえば，共有メモリのアタッチの場合，要求された処理を示すパラメータ値（SHMAT），IPC 識別子，ユーザが指定した共有メモリリージョンの開始アドレス，共有メモリに関するフラグ値の 4 つのデータを引数として取得する．History Logger は，これらのデータに加え，時刻やプロセスの属性値を履歴として History Repository に格納する．

また，共有メモリのアタッチの場合，制御に利用するために，共有メモリリージョンのアドレスを履歴として取得する必要がある．この値はシステムコールの実行後に決定されるため，後述の Alternative IPC 内で，アドレス値を履歴に追記する．

4.2.2　共有メモリに関するシステムコールの制御

共有メモリに関するシステムコールの制御機構として，ipc システムコールに対応する Alternative System Call Module（以下，Alternative IPC と記す）を実装した．Alternative IPC は，ipc システムコールが発行されたときに呼び出され，共有メモリのアタッチやデタッチを制御する．

共有メモリへの書き込みアクセスに対してページフォルトを発生させるために，Alternative IPC は，以下に示す手順により，PTE を変更する．

(1)　従来のシステムコール関数を呼び出すことにより，従来と同様の処理を行う．

(2)　要求された処理が共有メモリのアタッチの場合は，保護ファイルがオープンされているか検査する．オープンされている場合は，この時点から制御を開始する必要があるため，(3) 以降で示すように，ページフォルトを発生させるための処理を行う．オープンされていない場合には，Shared Memory Log に履歴として共有メモリリージョンの先頭リニアアドレスを格納する．要求された処理が共有メモリのデタッチの場合には，対応する共有メモリに関する履歴を削除し，処理を終了する．

(3)　History Repository の当該履歴に共有メモリリージョンの先頭リニアアドレスを追記し，共有メモリリージョンに対して，制御すべき共有メモリであることを示すフラグを設定する．

(4)　共有メモリリージョンに対してページフレームを割り当て，対応する PTE の値を設定した後，それらすべての PTE の書き込みビットをクリアする．共有メモリのサイズがページサイズよりも大きい場合，対応する PTE は複数存在する．これにより，これらの PTE が示すページは書き込み禁止となり，書き込みアクセスを行うとページフォルト例外が発生する．

保護ファイルにアクセス済みのプロセスが共有メモリをアタッチした場合，アタッチの際に発行される ipc システムコールをトリガとして，上記手順によりページフォルトを発生させるための処理を行う．一方，保護ファイルにアクセスしていないプロセスが共有メモリをアタッチし，後に保護ファイルにアクセスした場合には，(2) の処理により Shared Memory Log に蓄積された履歴を利用して，保護ファイルのオープンをトリガとした制御を行う．プロセスが保護ファイルをオープンすると，open システムコールの制御モジュールは，Shared Memory Log を参照することにより，共有メモリがアタッチされているか検査する．共有メモリをアタッチしている場合には，上記と同様の手順により，PTE の書き込みビットをクリアする．

4.2.3　共有メモリへの書き込みアクセスの制御

共有メモリアクセスの制御機構として，Extended Page Fault Handler を実装した．Extended Page Fault Handler は，従来のページフォルト例外ハンドラに，データ保護ポリシに基づき共有メモリへの書き込みの可否を決定する機能を追加したものである．

ページングによるメモリの管理の場合，保護ファイルをオープンしたプロセスが共有メモリへ書き込みアクセスを行うと，MMU がページテーブルを参照し，論理アドレスから物理アドレスへの変換を行う．4.2.2 項で述べた処理により，共有メモリに対応する PTE の書き込みビットがクリアされているため，アドレス変換過程においてページフォルト例外が発生し，Extended Page Fault Handler に処理が移行する．Extended Page Fault Handler は，ページフォルト例外の原因を特定し，原因が制御すべき共有メモリへの書き込みである場合，当該共有メモリに対する書き込みの可否を判定する．具体的には，Policy List を参照し，プロセスがアクセスしたすべての保護ファイルに対応するデータ保護ポリシと，書き込み時のコンテキストを比較する．書き込みを許可する場合，Extended Page Fault Handler は，当該 PTE を書き込み許可に変更し，処理を終了する．ハンドラの処理が終了すると，再度同じ書き込みアクセスが実行され，共有メモリへデータを書き込むことが可能となる．一方，書き込みを拒否する場合は，メモリアクセス違反を示すシグナルをプロセスに通知する．このシグナルが発行された際の処理は，コールバック関数として定義可能である．

ただし，3.2.3 項で述べたように，1 度書き込みを許可した共有メモリに対しても継続的にコンテキストに基づく制御を行うために，再度 PTE の書き込みビットをクリアする必要がある．本機構では，プロトタイプとして，コンテキストスイッチが行われた際に，共有メモリに対応する PTE の書き込みビットをクリアする機能を CPU スケジューラに追加した．Linux カーネル 2.6 では，x86 アーキテクチャにおけるタイムスライスは 1 ミリ秒であるため，少なくとも 1 ミリ秒ごとに書き込みビットがクリアされる．

5. 評　　価

5.1 機 能 評 価

5.1.1　テストプログラムによる評価

本手法により，コンテキストに適応して共有メモリを制御可能であることを示すために，テストプログラムを作成し，移動端末を用いて実験を行った．テストプログラムは，ファイルをオープンした後，共有メモリを確保，アタッチし，30 秒周期で読み出しと書き込みを交互に行うものである．

(1)　ファイルに対し，図 1 に示したデータ保護ポリシを定義する．このデータ保護ポリシは，roomA にいる場合のみ，共有メモリを含むプロセス間通信を許可することを示している．
(2)　roomA 内にいる状況において，テストプログラムを実行する．
(3)　読み出しと書き込みが 1 度ずつ行われるまで待機する．
(4)　roomA 外に移動し，(3) と同様の操作を行う．

上記の結果，(2) で共有メモリがアタッチされ，(3) で共有メモリの内容が正常に読み出された．また，データ保護ポリシの send_local に対するアクセス権限を満たすため，共有メモリへの書き込みが正常に実行された．一方，(4) において，roomA 外に移動した場合，読み出しアクセスは正常に実行されたが，書き込みアクセスに対してはエラーを示すシグナルが発行され，実行は失敗した．以上より，コンテキストを用いてデータ保護ポリシに記述された send_local の許可条件に基づき，共有メモリに対する書き込みアクセスを制御可能であることが確認できた．

5.1.2　アプリケーションを用いた評価

本手法を既存のアプリケーションにも適用可能であることを示すために，ImageMagick [8] を用いて実験を行った．ImageMagick は画像の表示，フォーマットの変換，合成などの機能を提供するソフトウェアである．ImageMagick は，画像を表示するときに display 機能を利用する．display 機能は，X サーバと通信を行い，画像ファイルの内容をディスプレイ上に表示する．このとき，X サーバと X クライアント間の通信は，MIT（Massachusetts Institute of Technology）による X Window System の共有メモリ拡張サポート機能（以下，MIT-SHM と記す）を用いて実現される．MIT-SHM は，X サーバと X クライアントがアタッチした共有メモリ上に，画像に関する情報を格納した XImage と画像データを配置することにより，X サーバと X クライアント間の高速な通信を実現する仕組みである．

実験では，この MIT-SHM によるデータ漏洩を防止するために，機密情報を格納した画像ファイルに対して図 1 に示したデータ保護ポリシを定義し，roomA の内外で display 関数を実行した．その結果，roomA 内では画像ファイルの内容が表示されたが，roomA 外では display 関数がエラーとなり，画像は表示されなかった．以上の結果より，ImageMagick のような既存のアプリケーションに対して，本手法による制御を適用可能であることを確認した．

5.2　性 能 評 価

本手法を用いて共有メモリを制御した際のオーバヘッドを計測するために，CPU が Celeron 2.4 GHz，メモリ 512 MB の PC/AT 互換機を用いて実験を行った．実験では，以下の 3 種類のプロセスにおいて，共有メモリのアタッチと書き込みアクセスの処理時間を計測した．なお，実験で使用した *Salvia* は，Linux カーネル 2.6.8 を基に実装している．

(1)　標準の Linux カーネル 2.6.8 におけるプロセス
(2)　*Salvia* における，保護ファイルにアクセスしていないプロセス
(3)　*Salvia* における，保護ファイルにアクセスしたプロセス

5.2.1　共有メモリのアタッチ

共有メモリのアタッチの処理時間を計測するために，`shmat` 関数を 1,000 回実行するテストプログラムを作成し，テストプログラム内で処理時間を計測した．処理時間は，`shmat` 関数の前後で RDTSC（read-time stamp counter）命令を用いて計測したクロックサイクル数を基に算出した．共有メモリのアタッチの際に行う処理のうち，PTE の設定や変更に要する時間は，アタッチする共有メモリのサイズにより変化する．そこで，実験では，共有メモリのサイズを 4 KB，40 KB，400 KB の 3 種類に変更し，(1)〜(3) のそれぞれにおいて実験を行った．

`shmat` 関数の平均処理時間を**表 1** に示す．共有メモリのサイズが 4 KB の場合，(1) に対して，(2) では約 1.08 倍，(3) では約 2.13 倍に処理時間が増加した．(2) は，保護ファイルにアクセスしているかを検査し，Shared Memory Log へ履歴を格納するオーバヘッドで，(3) はページを割り付け PTE を設定するオーバヘッドである．また，共有メモリのサイズを増加させた場合，(1) と (2) ではほとんど変化がなかったのに対して，(3) では処理時間がやや増加した．これは，共有メモリのサイズの増加にともない，変更する PTE の数が増加したためであると考えられる．

5.2.2　共有メモリへの書き込み

共有メモリへの書き込みアクセスの処理時間を計測するために，共有メモリに対して 1 バイトのデータを書き込んだ際の処理時間を計測した．ただし，(2) では (1) と同様の処理

表 1　共有メモリのアタッチの処理時間
Table 1　Processing time of `shmat` function.

	processing time [μs]		
	4 KB	40 KB	400 KB
(1) Linux	2.56	2.59	2.61
(2) *Salvia* (uncontrolled)	2.76	2.84	2.79
(3) *Salvia* (controlled)	5.45	5.51	5.73

表 2　共有メモリへの書き込みアクセスの処理時間（4 KB）
Table 2　Processing time of write access to shared memory (4 KB).

		processing time [μs]			
		average	max	min	SD
(1) Linux	(a)	12.65	14.27	11.12	0.87
	(b)	0.28	0.59	0.19	0.08
(3) *Salvia*	(a)	18.63	27.06	16.01	1.16
	(b)	0.28	0.56	0.23	0.08

SD: standard deviation

が行われるため，(1) と (3) について実験を行った．また，(3) では，図 1 に示したデータ保護ポリシを付与し，send_local に対するアクセス権限を満たす状況下で実験を行った．実験では，配列として用意した共有メモリに対してシーケンシャルにアクセスを行うテストプログラムを作成し，各アクセスに要する時間をテストプログラム内で計測した．処理時間の計測には，共有メモリのアタッチの場合と同様に RDTSC 命令を用いた．

4 KB の共有メモリに対し，1,000 回の試行を行った結果を**表 2** に示す．表 2 の (a) は，共有メモリに対して初めてアクセスした際の処理時間であり，(b) は 2 回目以降の各アクセスに要した時間である．(1) と (3) のいずれにおいても，メモリアクセスの処理時間は，初回のアクセスと 2 回目以降のアクセスで大きく変化した．標準の Linux の場合，デマンドページング方式により初回アクセス時にページを割り付けるため，(1) における初回アクセスの平均処理時間は，(a) に示すように約 12.65 μs となった．処理時間が (a) となるのは初めてアクセスした場合のみであり，それ以外のアクセスについては (b) に示す処理時間となる．一方，*Salvia* では，書き込みビットをクリアする際にページを割り付けるため，初回アクセス時にページの割付けは行われない．しかし，当該共有メモリに対する書き込みの可否を判定するために，ページフォルト例外が発生する．そのため，(3) における初回アクセスの処理時間 (a) は約 18.63 μs となった．(a) のような処理時間となるのは，これ以降，着目すべきコンテキストが変化し，再度書き込みの可否を判定する必要がある場合のみである．それ以外のアクセスの処理時間は (b) であり，(1) の (b) の場合と同等となる．よって，*Salvia* 特有のオーバヘッドは，メモリアクセスの処理時間 (b) と比較すると約 66.54 倍程度の大きさであることが分かる．また，(1) の (a) と (3) の (a) の差から，*Salvia* では 1 ページあたり約 2 μs の処理時間の増加となることが分かる．

また，書き込み時に行う処理のうち，書き込みビットのセット処理については共有メモリのサイズにより処理時間が変化する．そこで，共有メモリのサイズを 40 KB と 400 KB に

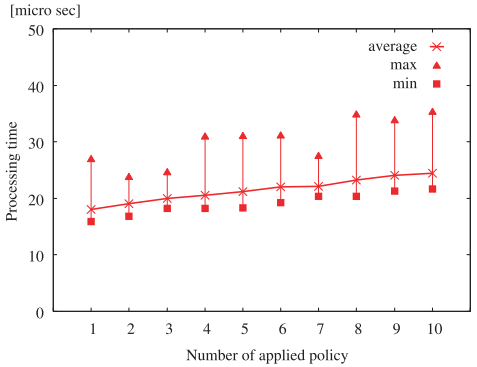


図 3　データ保護ポリシの規模と書き込みアクセスの処理時間の関係
Fig. 3　Relation between size of data protection policy and processing time of write access.

変更し，それぞれに対して前述と同様の実験を行った．その結果，(1) の処理時間は変化しなかったのに対し，(3) においては，初回アクセスの平均処理時間が，4 KB の場合に対して 40 KB の場合は約 1.01 倍，400 KB の場合は約 1.03 倍となった．

さらに，コンテキストと保護ポリシの比較時間は，比較対象となるデータ保護ポリシの規模により変化する．そこで，アクセスする保護ファイルの数を 1 から 10 まで変更し，4 KB の共有メモリに書き込みアクセスを行った際の初回アクセスの処理時間を計測した．1,000 回の試行を行った際の平均値，最大値，最小値を**図 3** に示す．実験結果より，データ保護ポリシの規模が増加すると，書き込みアクセスに要する時間も線形に増加することが分かる．したがって，本実験では，保護ファイルの数を 1 から 10 まで変化させたが，この数が増加した場合の処理時間は式 (1) で予想できる．ただし n は保護ファイルの数とし，$time$ はコンテキストと保護ポリシの比較時間（μs）である．

$$time = 18.0 + 0.64n \quad (1)$$

5.2.3　書き込みビットのクリアの処理時間

コンテキストが変化した場合，*Salvia* は，PTE の書き込みビットをクリアすることによって，プロセスが共有メモリをアクセスしたことを再度検出し，アクセスの可否の再判定を行う．このときの書き込みビットのクリアにかかる時間を計測した．この処理時間は，共有メモリのアタッチの履歴の数や，アタッチしている共有メモリのサイズにより変化する．そこで，(3) のプロセスにおいて，アタッチしている共有メモリのサイズを 4 KB，40 KB，400 KB に変化させ，アタッチしている共有メモリの数が 1，10，100 のそれぞれの場合に

表 3　書き込みビットのクリアに要する時間
Table 3　Processing time to clear write bit.

number of shared memory	processing time [μs]		
	4 KB	40 KB	400 KB
1	0.85	1.05	1.17
10	2.16	2.20	6.47
100	13.71	18.72	65.61

ついて計測した．

計測した結果を**表 3** に示す．実験結果より，4 KB の共有メモリを 1 個アタッチしている場合，処理時間は約 0.85 μs であった．この状況において，共有メモリのサイズを 4 KB から 40 KB にした場合，処理時間は約 1.24 倍に増加し，400 KB にした場合には約 1.38 倍に増加した．また，共有メモリの数を 1 から 10 に変更した場合，処理時間は約 1.37 倍に増加し，100 に変更した場合は約 16.13 倍に増加した．以上より，書き込みビットのクリアの処理時間は，共有メモリのサイズよりも数に大きく影響され，共有メモリの数が増加するにともない，線形に増加することが分かる．

5.3 考　　察

前節では，共有メモリのアタッチに関して，提案手法により制御を行った場合に `shmat` 関数のオーバヘッドが約 3 μs となることを示した．ただし，共有メモリを利用してプロセス間通信を行う場合，初めに 1 つの共有メモリを確保し，その共有メモリに対して読み書きアクセスを繰り返すことが多い．この場合，共有メモリをアタッチする必要があるのは 1 回のみであるため，上記のオーバヘッドがプロセスの実行に及ぼす影響は少ないと考えられる．

共有メモリへの書き込みに関しては，ページフォルト例外により書き込みアクセスを制御した場合の処理時間は，制御を行わない場合の約 66.54 倍となることを示した．これは，標準 Linux と比較すると，*Salvia* では 1 ページあたり約 2 μs の処理時間の増加となることを示した．また，データ保護ポリシの規模や共有メモリのサイズの増加にともない，書き込みに要する時間はさらに増加した．しかし，データ保護ポリシの規模が 10 倍になった場合の処理時間の増加は，約 1.25 倍程度に収まる．また，共有メモリのサイズを 100 倍にした場合，処理時間は約 1.03 倍とわずかに増加したのみであった．したがって，書き込みの際に生じるオーバヘッドは，数十 μs オーダとなることが予想できる．さらに，このオーバヘッドは，着目すべきコンテキストが変化し，再度書き込みの可否を判定する必要がある際にのみ発生するため，共有メモリに対するアクセスの大半は，標準の Linux と同程度の処理時間となる．以上より，共有メモリへの書き込みアクセスの制御にともなうオーバヘッドがプロセスの実行時間に占める割合は少ないと考えられる．

書き込みビットのクリアの処理時間については，プロセスが 4 KB の共有メモリを 1 つアタッチしている場合に約 0.85 μs であることを示した．この値は，メモリのサイズや数の増加にともない増加した．ただし，この値は，プロセスがアタッチしているサイズよりも数に大きく影響される．一般に，プロセスが確保する共有メモリは少なく，ImageMagick の場合でも 3 つ程度である．またクリアの処理は *Salvia* のコンテキストが変化したときのみに発生するため，この処理がプロセスの実行に影響を及ぼすことは少ないと考えられる．

一般的なアプリケーションへの適用を考えた場合，オーバヘッドは，`shmat` 時のオーバヘッド（約 3 μs），共有メモリへの初回アクセスの際のオーバヘッド（約 2 μs），コンテキストが変化した際の書き込みビットのクリアの処理時間（4 KB の共有メモリ 1 つであれば約 0.85 μs）の和となる．ただし，高速化を目的として用いる共有メモリにおいて，`shmat` を頻繁に呼び出すような場合は稀であると考えられる．また，コンテキストの変化の頻度は最も頻繁に変化する時刻であっても 1 秒に 1 回であるため，オーバヘッドは非常に小さいといえる．

6. 関 連 研 究

情報漏洩事故の主な発生要因として，正当なアクセス権限を持つユーザによる不正アクセスや誤操作があげられる．このため，不正アクセスを防止するためのセキュリティ技術が提案されている．それらの技術には，情報フローを制御する目的として，Bell-LaPadula モデル[9] のように秘匿性を重視した方式と，完全性を重視した方式がある．また，制御手法として，静的に情報フローを検出する手法と，動的にアクセスを制御する手法がある．さらに，動的なアクセス制御手法には，アクセス制御の粒度として，システムコールを制御する手法，メモリアクセスを制御する手法，変数間の情報フローを制御する手法などがある．

静的に情報フローを検出する方法として，分散ラベルモデル（Decentralized Label Model）[10] がある．分散ラベルモデルは，Java を拡張したプログラミング言語である Jif を導入し，コンパイル時に情報フローを解析する．プログラム中の変数に対してアクセス制御ポリシを示すラベルを付与し，変数間におけるデータコピーや通信チャネルへのデータの送信を，ラベルに基づき制御する．これに対し，*Salvia* は，計算機資源へのアクセス要求時のコンテキストに適応してデータの伝搬範囲を制限するために，プログラム実行時に動的にアクセス制御を行う．また，言語処理系による情報フローの検出を行わないため，プログラ

ミング言語に依存しないという特徴を持つ．

実行時にアクセス制御を課する方式には，プログラムの動作やアクセス可能な計算機資源を制限するサンドボックス環境を利用する方法や，アクセス制御方式を実現したセキュアOS を利用する方法がある．

SoftwarePot [11] は，サンドボックス環境を実現するミドルウェアである．SoftwarePot では，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や計算機資源へのアクセスをシステムコールの引数の検査や書き換えによって実現している．また，細粒度保護ドメインを利用することにより，サンドボックス環境を実現することも可能である[12]．細粒度保護ドメインでは，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によって決定された保護ポリシに基づき，カーネルがシステムコールの実行を制御する．*Salvia* は，システムコールを制御することにより，動的なアクセス制御を実現する点でこれらと類似している．しかし，*Salvia* はプライバシ保護を目的としているため，データ提供者やデータ管理者の意思を反映したデータ保護ポリシをファイルごとに定義し，それらのデータ保護ポリシに基づくアクセス制御を実現する．SoftwarePot では，アクセス制御のためのセキュリティポリシは，利用者により書き換えが可能であるため，データ提供者の意思と異なるポリシが定義される可能性がある．また，細粒度保護ドメインでは，保護ポリシはプロセスに対して定義されるため，データ提供者の意思を反映したプライバシ保護を実現できない．

SELinux [5] は，Role-Based Access Control [13] や TE [6] といったアクセス制御方式を適用したセキュア OS である．SELinux では，最小特権の原則に基づき，プロセスやユーザに対して，アプリケーションの実行に必要な最低限のアクセス権限のみを付与する．具体的には，セキュリティポリシに基づいて，計算機資源へのアクセスを行うシステムコールの実行の可否を制御する．しかし，Role-Based Access Control や TE には，アクセス要求時の状況に適応した制御を行う仕組みはない．*Salvia* では，アクセス要求時の状況をコンテキストとして定義し，コンテキストに適応したアクセス制御を実現している．また，SELinux は，共有メモリのようなシステムコールを介さない計算機資源へのアクセスを制御する機能は有していない．

細粒度保護ドメイン[14] は，悪意のある拡張コンポーネントから計算機資源を保護することを目的としている．特に，ページテーブルに対して複数の保護モードを同時に設定可能としたマルチプロテクションページテーブルと呼ばれる機構により，不正なメモリアクセスを防止している．マルチプロテクションページテーブルは，セグメントディスクリプタの存在フラグをソフトウェア割込みによって書き換えることにより実現されている．細粒度保護ドメインは，メモリアクセス時のアドレス変換過程でアクセス制御を行う点で，*Salvia* における共有メモリ制御手法と類似している．しかし，細粒度保護ドメインは，各計算機資源へのアクセスを直接的に制限するために，セグメントを単位として保護ドメインを切り替える．これに対し，*Salvia* における共有メモリ制御機構では，アクセスの可否を判定するために，ページ単位で書き込みビットを変更し，共有メモリアクセスを選択的にフックする．このように，細粒度保護ドメインは，例外を発生させる粒度と目的において提案手法と異なる．

GIFT [15] は，情報フローを検出するためのフレームワークであり，動的な情報フローの追跡を可能とする C 言語用のコンパイラ（GIFT コンパイラ）を提供している．GIFT では，GIFT コンパイラにより挿入された変数のチェック関数をプログラム実行時に呼び出すことにより，動的な情報フローの追跡を実現している．*Salvia* も，GIFT と同様に，プログラム実行時に情報フローを検出する．しかし，GIFT が外部からのデータの不正な実行を防止することを目的としているのに対し，*Salvia* はデータの伝搬範囲を制限することを目的としてアクセス制御を行う．

DIFT [16] は，専用のハードウェアを利用することにより，外部から注入されたデータによる不正アクセスを防止可能としている．DIFT では，メモリに入力されたデータに対してフラグを付与し，専用のプロセッサがこのフラグを利用して情報フローを追跡する．本論文で示した共有メモリ制御手法は，ページフォルト例外を利用するため，ハードウェアに依存する点で DIFT と類似している．ただし，ページフォルト例外は，現在一般に広く利用されているアーキテクチャの大半において実装されているため，提案手法の適用場面も広いと考えられる．

7. おわりに

本論文では，*Salvia* における共有メモリの制御手法，Intel x86 アーキテクチャにおける実装例，評価について述べた．提案手法は，ページフォルト例外を利用し，共有メモリへの書き込みアクセスを制御することにより，複数のプロセス間におけるデータの共有を制限する．また，コンテキストに適応したアクセス制御方式を適用することにより，データ漏洩の原因となりうる共有メモリアクセスを選択的に制限する．また，性能評価では，提案手法を適用した際の処理時間を計測し，共有メモリのアタッチの際に約 3 μs のオーバヘッドが生じることを示した．共有メモリへの書き込みアクセスについては，書き込みの可否を判定した場合に処理時間が約 66.54 倍となることを示し，これらのオーバヘッドがプロセスの実行

に与える影響について議論した．今後の課題として，本論文ではプロトタイプとして CPU スケジューラに実装を行った PTE の書き込みビットをクリアする機構を，3.2.3 項で述べたように，着目すべきコンテキストの変化をトリガとする方法に変更する必要がある．

参 考 文 献

1) 内閣府：個人情報の保護に関する法律. http://www5.cao.go.jp/seikatsu/kojin/index.html
2) 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
3) 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. http://www.kokusen.go.jp/cgi-bin/byteserver.pl/pdf/n-20050325_1.pdf
4) 鈴来和久，一柳淑美，毛利公一，大久保英嗣：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，情報処理学会論文誌：コンピューティングシステム，Vol.47, No.SIG 3, pp.1–15 (2006).
5) National Security Agency: Security-Enhanced Linux. http://www.nsa.gov/selinux/
6) Boebert, W. and Kain, R.: A Practical Alternative to Hierarchical Integrity Policies, *Proc. 8th National Computer Security Conference*, pp.18–27 (1985).
7) 原田季栄，保理江高志，田中一男：TOMOYO Linux — タスク構造体の拡張によるセキュリティ強化 Linux，*Linux Conference 2004* (2004).
8) ImageMagick Studio LLC: ImageMagick: Convert, Edit, and Compose Images. http://www.imagemagick.org/script/index.php
9) Bell, D. and LaPadula, L.: Secure Computer System: Unified Exposition and Multics Interpretation, Technical report, MITRE Corporation (1976).
10) Myers, A. 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).
11) 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).
12) 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，電子情報通信学会技術研究報告，CPSY2002-23, Vol.102, No.153, pp.87–94 (2002).
13) 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).
14) 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軽量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).
15) Lam, L.C. and cker Chiueh, T.: A General Dynamic Information Flow Tracking Framework for Security Applications, *ACSAC '06: Proc. 22nd Annual Computer Security Applications Conference*, pp.463–472 (2006).
16) Suh, G.E., Lee, J.W., Zhang, D. and Devadas, S.: Secure Program Execution Via Dynamic Information Flow Tracking, *ASPLOS XI: Proc. 11th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp.85–96 (2004).

付　　録

A.1　データ保護ポリシの DTD

Salvia におけるデータ保護ポリシの DTD を**図 4** に示す．大きくは，特に条件が設定されていないときの挙動を定義する部分 (1)，条件によってアクセス制御の挙動を変更したい場合の条件定義の部分 (2) と挙動の定義 (4)，データ保護ポリシへのアクセスを保護するための認証情報の定義 (5) に分類される．

(1) デフォルトアクセス

4 行目　ファイルを読み出す許可の有無を定義する．

5〜8 行目　プロセスが保護ファイルを読み出した後の，ファイル全般へのデータ書き込みについての許可の有無を定義する．6〜7 行目の write_access タグの update 属性は，特に保護ファイルについて，更新を許可するかどうかを定義する．8 行目の filename は，特定のファイルへのアクセスを制限したいときに定義できる．

9 行目　プロセスが保護ファイルを読み出した後の，他のローカルプロセスにデータを書き込む許可を定義する．本論文で述べた共有メモリのアクセスはこれに属する．

10〜13 行目　プロセスが保護ファイルを読み出した後の，リモートプロセスにデータを書き込む許可を定義する．12〜13 行目の ip_address は，特定の送信先の計算機だけを制限したい場合に定義できる．

14 行目　システムコールを個別に制限する場合に，システムコール名と許可の有無を定義する．

(2) 条件定義

19〜21 行目　ユーザ ID を指定する．ユーザ ID は実ユーザ ID または実効ユーザ ID が指定できる．

22〜24 行目　グループ ID を指定する．グループ ID は実グループ ID または実効グルー

```
1  <!ELEMENT poricy(data_protection_policy,manager_list)>
2   <!ELEMENT data_protection_policy (default_access?, data_protection_domain+)>
3    <!ELEMENT default_access (read?, write?, send_local?, send_remote?, syscall*)>
4     <!ELEMENT read (#PCDATA)>
5     <!ELEMENT write (write_access, filename*)>
6      <!ELEMENT write_access (#PCDATA)>
7      <!ATTLIST write_access update (deny|allow) "deny">
8      <!ELEMENT filename (#PCDATA)>
9     <!ELEMENT send_local (#PCDATA)>
10    <!ELEMENT send_remote (send_remote_access, ip_address*)>
11     <!ELEMENT send_remote_access (#PCDATA)>
12     <!ELEMENT ip_address (#PCDATA)>
13     <!ATTLIST ip_address version (4|6) "4">
14    <!ELEMENT syscall (#PCDATA)>                                        (1)
15   <!ELEMENT data_protection_domain (ACL)>
16   <!ATTLIST data_protection_domain type (read|receive|both|none) "both">
17    <!ELEMENT ACL (context, (access?|ACL*))>
18    <!ELEMENT context (user*, group*, time?, location*, frequency?)>
19     <!ELEMENT user (user_id+)>
20      <!ELEMENT user_id (#PCDATA)>
21      <!ATTLIST user_id type (effective|real) "real">                   (2)
22     <!ELEMENT group (gourp_id+)>
23      <!ELEMENT group_id (#PCDATA)>
24      <!ATTLIST group_id type (effective|own) "own">
25     <!ELEMENT time (second*)>
26      <!ELEMENT second (#PCDATA)>
27      <!ATTLIST second mode (relative|absolute) "relative">
28     <!ELEMENT location (area)>                                         (3)
29      <!ELEMENT area (device)>
30      <!ELEMENT device (net_radio?, GPS?, RFID?)>
31      <!ELEMENT net_radio (essid,quality)>
32       <!ELEMENT essid (#PCDATA)>
33       <!ELEMENT quality (#PCDATA)>
34      <!ELEMENT GPS (latitude,longitude,range)>
35       <!ELEMENT latitude (#PCDATA)>
36       <!ELEMENT longitude (#PCDATA)>
37       <!ELEMENT range (#PCDATA)>
38      <!ELEMENT RFID (tag_id)>
39       <!ELEMENT tag_id (#PCDATA)>
40     <!ELEMENT frequency (read?, write?)>
41      <!ELEMENT read (#PCDATA)>
42      <!ELEMENT write (#PCDATA)>
43   <!ELEMENT access (read?, write?, send_local?, send_remote?, syscall*)>  (4)
44    <!ELEMENT read (#PCDATA)>
45    <!ELEMENT write (write_access, filename*)>
46     <!ELEMENT write_access (#PCDATA)>
47     <!ATTLIST write_access update (deny|allow) "deny">
48    <!ELEMENT filename (#PCDATA)>
49    <!ELEMENT send_local (#PCDATA)>
50    <!ELEMENT send_remote (send_remote_access, ip_address*)>
51     <!ELEMENT send_remote_access (#PCDATA)>
52     <!ELEMENT ip_address (#PCDATA)>
53    <!ELEMENT syscall (#PCDATA)>
54 <!ELEMENT manager_list(ACL+)>                                          (5)
55  <!ELEMENT ACL(context)>
56   <!ELEMENT context(user?,group*,password?,RFID?)>
57    <!ELEMENT user(user_id+)>
58     <!ELEMENT user_id(#PCDATA)>
59     <!ATTLIST user_id type (effective|real) "real">
60    <!ELEMENT group(group_id+)>
61     <!ELEMENT group_id(#PCDATA)>
62     <!ATTLIST group_id type (effective|own) "own">
63    <!ELEMENT password(password_str+)>
64     <!ELEMENT password_str(#PCDATA)>
65    <!ELEMENT RFID(tag_id+)>
66     <!ELEMENT tag_id(#PCDATA)>
```

図 4　データ保護ポリシの DTD
Fig. 4　DTD of data protection policy.

プ ID が指定できる．

25〜27 行目　時刻を指定する．現在時刻またはファイルをオープンしてからの経過時間を指定できる．

28〜39 行目　計算機の位置を指定する．詳細は次の (3) 位置定義で述べる．

40〜42 行目　ファイルにアクセスする回数を指定する．読み書きについてそれぞれ指定できる．

(3) 位置定義

31〜33 行目　無線 LAN を用いて位置を指定する．ESSID と電波強度によって範囲を定義する．

34〜37 行目　GPS を用いて位置を指定する．緯度と経度，およびその点からの距離を定義する．

38〜39 行目　RF タグを用いて位置を定義する．RF タグの ID を定義する．これは，特定の ID のタグを読める位置にいるかどうかを示す．

(4) アクセス動作定義

(1) デフォルトアクセスと同様の動作について定義できる．ここでは省略する．

(5) 認証定義

57〜59 行目　ユーザ ID による認証の定義．

60〜62 行目　グループ ID による認証の定義．

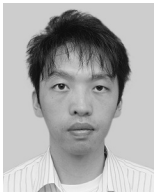
63〜64 行目　パスワードによる認証の定義．

65〜66 行目　RF タグによる認証の定義．

(平成 20 年 12 月 1 日受付)
(平成 21 年　6　月 4 日採録)

廣　　真文（正会員）

昭和 58 年生．平成 18 年立命館大学理工学部情報学科卒業，平成 20 年同大学大学院理工学研究科修士課程情報理工学専攻修了．同年株式会社日本総合研究所に入社．主として，CTI 基盤技術を中心とした CRM システムのインフラ構築やソフトウェア開発に従事．修士（工学）．また，大学院在学中はプライバシ保護とオペレーティングシステムに関する研究に従事．



鈴来　和久（正会員）

昭和 53 年生．平成 20 年立命館大学大学院理工学研究科一貫制博士課程単位取得満期退学．同年三菱電機株式会社に入社し，先端技術総合研究所へ配属．主として組み込みリアルタイムシステム構築技術，およびソフトウェアプラットフォーム構築技術の開発に従事．また，大学院在学中はオペレーティングシステム構成法に関する研究に従事．

毛利　公一（正会員）

昭和 47 年生．平成 6 年立命館大学理工学部情報工学科卒業，平成 8 年同大学大学院理工学研究科修士課程情報システム学専攻修了，平成 11 年同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，平成 20 年同准教授となり，現在に至る．博士（工学）．オペレーティングシステム，コンピュータセキュリティ，コンピュータネットワークの研究に従事．電子情報通信学会，日本ソフトウェア科学会，ACM，IEEE-CS，USENIX 各会員．