システムソフトウェアと
オペレーティング・システム 88－10
（２００１．７．２７）

ヘルパアプリケーションの安全な実行環境

品川 高廣 †　　河野 健二 ††,†††　　益田 隆司 ††

† 東京大学大学院 理学系研究科 情報科学専攻
†† 電気通信大学 情報工学科，††† 科学技術振興事業団さきがけ研究 21
電子メール：shina@is.s.u-tokyo.ac.jp，{kono,masuda}@cs.uec.ac.jp

要旨

様々なファイル形式の Web コンテンツを閲覧するために，ヘルパアプリケーションが利用されている．しかしヘルパアプリケーションはインターネットから入手した信頼できないデータを扱うため，たとえデータの内容を閲覧するだけであってもユーザのコンピュータが不正アクセスされる危険性がある．本論文では，ヘルパアプリケーションのアクセス権限を必要最小限に制限するための機構を提案する．この手法では，我々が提案・開発中の SeeMoc オペレーティングシステムで提供する細粒度保護ドメインを利用している．実験によって，この手法による保護のオーバーヘッドは 0.1〜1% 程度に抑えられることを確認した．

A Secure Execution Environment for Helper Applications

Takahiro Shinagawa†　　Kenji Kono††,†††　　Takashi Masuda††

†Department of Information Science, Graduate School of Science, University of Tokyo
††Department of Computer Science, University of Electro-Communications,
†††PREST, Japan Science Technology Corporation
E-mail: shina@is.s.u-tokyo.ac.jp, {kono,masuda}@cs.uec.ac.jp

Abstract

Many users utilize helper applications to browse Web contents in various formats. However, the users are in danger of unauthorized access even if they only browse the contents, because the helper applications may process malicious data which is downloaded from Internet. This paper proposes the mechanism to restrict the access rights of the helper applications so that they have the least privilege. This mechanism exploits fine-grained protection domains, which is supported by the SeeMoc operating system we have developed. Experimental results show that the average overhead for protection is about 0.1 to 1%.

1 はじめに

インターネット上の Web コンテンツは様々なファイル形式で提供されており，それらを閲覧するためにヘルパアプリケーションが利用されている．ヘルパアプリケーションとは，Web ブラウザでは解釈することができないファイル形式の Web コンテンツを閲覧するために，Web ブラウザに代わって解釈・表示などを行なうプログラムである．例えば PDF 形式のファイルを表示する Acrobat Reader や，Postscript 形式のファイルを解釈する ghostscript などがヘルパアプリケーションとして使われている．

しかしヘルパアプリケーションが扱う Web コンテンツはインターネットから入手した信頼できないデータであるため，たとえデータの内容を閲覧するだけであってもユーザのコンピュータが危険にさらされる可能性がある．悪意を持った Web コンテンツの作者は，アプリケーションが予期していないような内容のデータを与えることによって，アプリケーションを誤動作させてその制御を奪うなどの攻撃が可能である．実際に Acrobat Reader には，バッファオーバーフロー攻撃に対する脆弱性 [2]，ghostscript にはファイルが不正にアクセスされる問題が報告されている [1]．これらは LAN のような閉じた環境では単なるプログラムのバグとして扱うこともできるが，インターネットに接続された環境では重大なセキュリティ上の問題となる．アプリケーションはますます複雑になってきているので，プログラムを正しく作成することは困難である．

ヘルパアプリケーションを利用した攻撃によって被害を受ける可能性を減らすためには，ヘルパアプリケーションに必要以上の権限を与えない機構が必要である．ヘルパアプリケーションは通常のユーザの権限で動作するため，従来のオペレーティングシステムでは，攻撃を受けた場合にはファイルなどユーザがアクセス可能な全ての資源が危険にさらされてしまう．**最小特権の原則（*Principle of Least Privilege*）**に従って，ヘルパアプリケーションが動作するために必要最小限の資源以外へのアクセスを禁止することによって，たとえヘルパアプリケーションの制御が奪われたとしても，不正アクセスによる被害を最小限に抑えることができる．

本論文では，ヘルパアプリケーションのアクセス権限を制限するための機構を提案する．この機構では，アプリケーションの実行に必要ではない資源へのアクセスを制限するために，ポリシーモジュールと呼ぶコードをアプリケーションのプロセス内に挿入することでアクセス制御を実現する．ポリシーモジュールは**参照モニタ** *(reference monitor)* の役割を果たすコードで，アプリケーションが発行するシステムコールを横取りして，不正なアクセスをチェックする作業を行なう．ポリシーモジュールをアプリケーションから保護するために，プロセス内に保護ドメインを作成して，アプリケーションと参照モニタに異なる保護ドメインに割り当てる．アプリケーションを改変せずに保護を実現するために，ELF ローダを改造してアプリケーションの起動時にポリシーモジュールの挿入と保護ドメインの作成，割り当てなどの処理を行う．

この保護機構では，我々が提案・開発を行っている SeeMoc オペレーティングシステムの機能を利用している [16, 13]．SeeMoc では，１つのプロセス内に複数の保護ドメインを持つことが可能になっており，このプロセス内の保護ドメインを我々は細粒度保護ドメインと呼んでいる．細粒度保護ドメインのアクセス制御は，ユーザレベルで実装されるポリシーモジュールによって，柔軟に行なうことができる．また，細粒度保護ドメインの切り替えはプロセスよりも高速に行うことが可能であり，保護のオーバーヘッドを低く抑えられる．

以下2章では，SeeMoc でサポートする細粒度保護ドメインの保護モデルについて説明する．3章では，アクセス制御を実現するための機構を説明する．4章では，細粒度保護ドメインによるオーバーヘッドを見積もる実験を行なう．5章で関連研究に触れ，6章 で本論文をまとめる．

2 保護モデル

SeeMoc オペレーティングシステムでサポートする細粒度保護ドメインは，細粒度のアクセス制御，効率の良い資源共有，高速な保護ドメイン切り替えといった特徴を備えている．本章では，この細粒度保護ドメインによる保護モデルについて説明する．

2.1 細粒度保護ドメイン

細粒度保護ドメインは，一つのプロセスの中に複数個持つことができる保護ドメインである．保護ド

メインとはアクセス可能な資源の集合を表す概念である．従来のオペレーティングシステムでは，プロセスが保護ドメインの役目を果たしている．細粒度保護ドメインはプロセスによる保護ドメインの部分集合として定義される．つまり，細粒度保護ドメインでアクセス可能な資源は，それが定義されたプロセスでアクセス可能な資源の中の一部になっている．

細粒度保護ドメインでは，アクセス制御を細かい単位で行なうことができる．例えば，メモリに対するアクセスは，ページ単位で細粒度保護ドメイン毎に異なる保護モードを設定することができる．また，ファイルやネットワークなどのシステム資源へのアクセスも任意の単位でアクセス権限を設定することができる．システム資源に対するアクセス制御の具体的な方式はオペレーティングシステムでは規定しておらず，プロセス毎にユーザレベルで実装されるポリシーモジュールと呼ぶコードにアクセス制御を委任する．従って，アプリケーションに合わせて様々な保護ポリシーを実現できる．

細粒度保護ドメインはプロセスの一部なので，細粒度保護ドメイン間の資源共有を効率よく行なうことができる．例えば，仮想アドレス空間を共有しているので，ポインタを含む複雑なデータ構造などでもメモリ上で容易に共有することができる．

細粒度保護ドメインはプロセスに割り当てられた資源を共有するので，保護ドメインの切り替えを高速に行なうことができる．例えば，ページテーブルを共有するので，保護ドメイン切り替えの際に TLB フラッシュに伴うコストを避けることができる．また，タイムスライスを共有するので，保護ドメイン切り替えの際のスケジューリングが不要である．

2.2 保護の実現

本説では，SeeMoc カーネルにおいて細粒度保護ドメインを実現するための機構について説明する．なお，詳細な実装の方法については文献 [13, 16] を参照されたい．

2.2.1 細粒度のアクセス制御

システムの資源に対して細かい単位でアクセス制御を行なうために，SeeMoc カーネルではシステムコールを横取りする機構を用意している．細粒度保護ドメインが割り当てられたモバイルコードがシステムコールを発行すると，予め登録されたユーザレベルで動作するポリシーモジュールが呼び出される．ポリシーモジュール自体も細粒度保護ドメインによって保護されており，通常は Web ブラウザ本体の細粒度保護ドメインが割り当てられる．

ポリシーモジュールは自身の保護ポリシーに基づいて，横取りしたシステムコールの扱いを判断する．SeeMoc カーネルはポリシーモジュールを呼び出す際に，呼び出し元のコードに割り当てられた細粒度保護ドメインを識別する ID と，システムコールの種類，及びその引数が渡される．ポリシーモジュールは，この ID とシステムコールの内容をチェックして，そのシステムコールの発行を許可，不許可を決定する．

ポリシーモジュールの呼び出しは，細粒度保護ドメイン間呼び出しで行なわれるので，横取りに伴うオーバーヘッドは低く抑えられる．また，ポリシーモジュールの呼び出し頻度を減らすために，`getpid()` などのシステムコールは無条件で許可するといった設定をすることもできる．

2.2.2 細粒度のメモリ保護

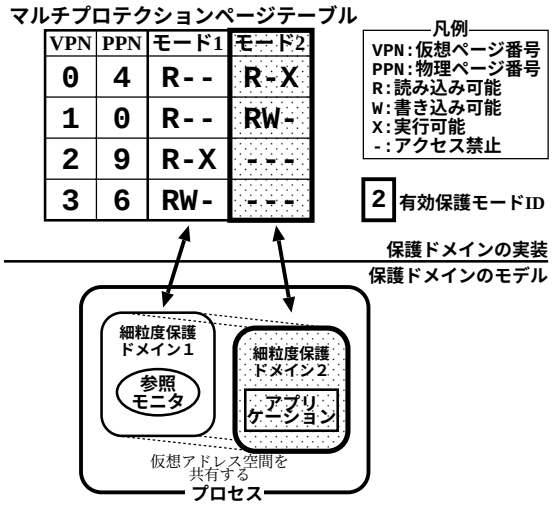

図 1: マルチプロテクションページテーブル

プロセス内でページ単位のメモリ保護を実現するために，SeeMoc カーネルではマルチプロテクションページテーブルという抽象概念を導入している [13, 16]．マルチプロテクションページテーブルとは，従来のページテーブルを拡張して，全てのペー

ジの保護モードを一度に切り替えられるようにしたものである．図1 上部に示すように，ページテーブルの各エントリには，仮想アドレスから物理アドレスへのマッピングに加えて，ページ保護モードを複数個設定できるようになっている．ページ保護モードの列がそれぞれ一つの細粒度保護ドメインに対応している．ある時点で有効な保護モードの列は一つだけであり，これを変更することによって保護ドメインを切り替えることができる．

マルチプロテクションページテーブルを用いると，細粒度のメモリ保護と同時に，効率の良い資源共有や高速な保護ドメイン切り替えも容易に実現できる．例えば細粒度保護ドメイン毎に異なるページ保護モードを設定しつつもアドレス空間を共有できるので，ポインタを含む複雑なデータ構造などの共有が容易に行なえる．また保護ドメイン切り替えの際にページテーブルを切り替える必要がないので，TLB フラッシュに伴うオーバーヘッドを避けて，高速な保護ドメイン切り替えが実現できる．

2.2.3 高速な保護ドメイン切り替え

高速な保護ドメイン切り替えを実現するために，SeeMoc カーネルは専用のシステムコールを用意している．細粒度保護ドメイン間呼び出しを行なうときには，呼び出し先の細粒度保護ドメインを識別する ID を引数として，このシステムコールを発行する．このシステムコールの中では，システムコールを呼び出したスレッドに割り当てられている細粒度保護ドメインの ID を，引数で指定した細粒度保護ドメインの ID に書き換える．そして，呼び出し元の細粒度保護ドメインの ID を引数として，予め細粒度保護ドメイン毎に登録されているエントリポイントに制御を渡す．

保護ドメイン切り替えに必要な処理は，本質的にはスレッドに割り当てられている細粒度保護ドメインの ID を書き換えるだけであり，高速な保護ドメイン切り替えが実現できる．

3 安全な実行環境の実現

本章では，SeeMoc オペレーティングシステムでサポートする細粒度保護ドメインを利用して，ヘルパアプリケーションの安全な実行環境を実現する手法について説明する．まず SeeMoc で細粒度保護ドメインを利用するためのシステムコールについて説明する．次に既存のアプリケーションを改変せずにポリシーモジュールを挿入して，アクセス制御を行なう手法について説明する．

3.1 細粒度保護ドメインのシステムコール

SeeMoc オペレーティングシステムでは，細粒度保護ドメインをサポートするために，以下のようなシステムコールを提供している．

`fpd_open();` 細粒度保護ドメインをオープンする．メモリ上の領域を指定して細粒度保護ドメインを作成し，その ID を返す．

`fpd_ctl();` 細粒度保護ドメインの保護ポリシーを設定する．システムコール発行の許可・不許可や，横取りした際に呼び出されるポリシーモジュール内のルーチンの設定などを行なう．

`fpd_call();` 呼び出し先の保護ドメインの ID を指定して，保護ドメイン間呼び出しを行なう．引数はレジスタ上で渡される．

`fpd_close();` 細粒度保護ドメインをクローズする．

なお，現在我々が開発中の SeeMoc カーネルは Intel の IA-32 と SPARC 上で動作している．カーネル内の実装の詳細については，文献 [13, 16, 17] を参照されたい．

3.2 アクセス制御の実現

本論文で述べる手法では，ヘルパアプリケーションのアクセス権限を制限するために，アプリケーションが発行するシステムコールをチェックするポリシーモジュールを配置する．このポリシーモジュール自身は，SeeMoc でサポートする細粒度保護ドメインを利用して，アプリケーション側から保護する．また，アプリケーションを改変せずに保護を実現するために，ELF ローダを改造してポリシーモジュールの挿入や細粒度保護ドメインの作成を行なっている．以下では，ELF ローダ内で行なう処理の詳細を説明する．

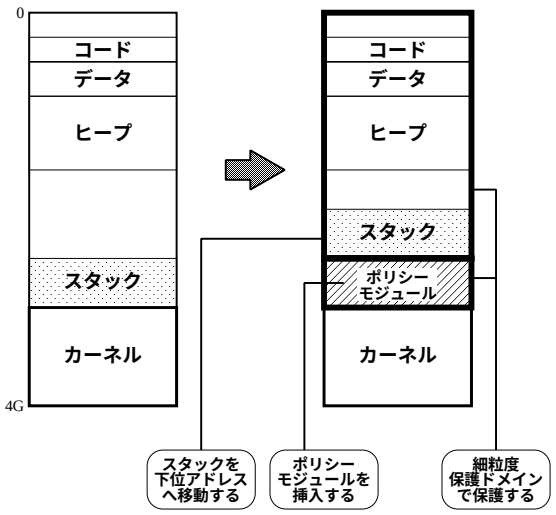

図 2: アドレス空間の配置

3.2.1 スタック領域の移動

まずアプリケーションのアドレス空間内にポリシーモジュールを挿入する領域を確保するために，スタック領域を下位アドレスに移動させる（図2参照）．これによって，アプリケーション本体を移動することなく，アドレス空間内に必要な領域を確保することができる．

スタック領域を移動させるためには，以下のような作業が必要である．まずスタック領域には環境変数や引数などのデータが格納されているので，これをコピーする必要がある．また，環境変数や引数は「文字列に対するポインタの配列」というデータ構造になっているので，ポインタに相当する部分のアドレスの値を調整する必要がある．最後にスタックポインタを新しいスタック領域の先頭に設定して，初期化の処理を続行する．

ヒープ領域の後方など他の領域にポリシーモジュールを挿入することが出来ないのは，Intel 版の SeeMoc の制約による．Intel 版 SeeMoc では，細粒度保護ドメインを作成するためにアドレス空間内に一定の領域を必要とするが，この領域はお互いに重なっていてはいけないという制約がある．従って，アプリケーションとポリシーモジュールの細粒度保護ドメインのための領域をアドレス空間内で重ならないように確保するために，スタック領域を移動してアプリケーションが利用するアドレス領域を小さくしている．

3.2.2 ポリシーモジュールの挿入

前説で確保した領域にポリシーモジュールのコードを配置する．ポリシーモジュールは独立したプログラムとして動作するので，この領域にポリシーモジュールのコード，データ，ヒープ，及びスタック領域をそれぞれ確保する．ポリシーモジュールはオブジェクトファイルとして ELF ローダとは別ファイルで提供されており，アプリケーション毎に異なる保護ポリシーを実現するポリシーモジュールを挿入することが可能である．

アプリケーションが発行したシステムコールは，このポリシーモジュールによって横取りされて，アクセス制御のための処理が行なわれるように設定される．ポリシーモジュールはアクセス制御リストやケーバビリティなどの機構を実装して様々な保護ポリシーを実現することができる．

3.2.3 細粒度保護ドメインによる保護

ポリシーモジュールをアプリケーションから保護するための細粒度保護ドメインを作成する．ポリシーモジュールはオブジェクトファイルとして ELF 形式で提供されているので，まず ELF ローダ内のコードを利用してポリシーモジュールをメモリ領域に読み込み，`fpd_open();` を呼び出して，ポリシーモジュールのための細粒度保護ドメインを作成する．

次に，アプリケーションのアクセス権限を制限するために，アプリケーションが読み込まれる領域に対して細粒度保護ドメインを作成する．アプリケーション自身の読み込みは既存の ELF ローダのコードをそのまま利用している．

細粒度保護ドメインを作成した後は，ページ保護モードを設定する．まずポリシーモジュールの細粒度保護ドメインでは，システムコールの引数として渡されたポインタをポリシーモジュールがチェック出来るようにするために，アプリケーションのメモリページ全体がアクセスできるように設定する．アプリケーションの保護ドメインでは，ポリシーモジュールを保護するために，ポリシーモジュールのメモリページにはアクセスできないようにする．

アプリケーションのアクセス制御を行なうために，ポリシーモジュールはアプリケーションがシステムコールを発行したときに呼び出されるコールバック

ルーチンを登録する．コールバックルーチンの呼び出しは，細粒度保護ドメイン間呼び出しになっているので，高速に行なうことができる．

最後に，細粒度保護ドメイン間呼び出しを利用して，アプリケーションの保護ドメインに切り替えて `main()` 関数を呼び出す．

4 性能実験

3章で示した手法で保護を行なった場合のオーバーヘッドを測定する実験を行なった．まず保護ドメイン間呼び出し自体にかかる時間を計測する実験を行なって，細粒度保護ドメインの基本性能を測定した．次にアプリケーションが発行するシステムコールを横取りすることによるオーバーヘッドを見積もる実験を行なった．

実験に使用したマシンは，CPU は PentiumIII 1GHz，メモリ 128M バイト，ハードディスク 75GB (DTLA-307030)，グラフィックカードは Matrox Millennium G450 を搭載した PC である．使用したオペレーティングシステムは，我々が Linux 2.2.18 を改造して細粒度保護ドメインの機構を組み込んだ SeeMoc カーネルバージョン 0.4 である．

4.1 保護ドメイン間呼び出し

SeeMoc の細粒度保護ドメインにおいて，保護ドメイン間呼び出しにかかるサイクル数を測定する実験を行なった．測定には PentiumIII に搭載されているサイクルカウンタを利用した．実験には保護ドメイン間呼び出しで何もしない手続きを呼び出して，その前後のサイクルカウンタの値の差を求めた．比較実験として，別プロセスの手続きを IPC (パイプ) で呼び出す場合にかかるサイクル数も計測した．表1に実験結果を示す．

表 1: 保護ドメイン間呼び出しにかかるサイクル数

方式	サイクル数	時間
SeeMoc	378	0.38μs
IPC (パイプ)	4,046	4.05μs

SeeMoc の保護ドメイン間呼び出しは IPC に比べると約 21 倍速く，高速な保護ドメイン間呼び出しが実現されていることがわかる．

4.2 ヘルパアプリケーション

一般的にヘルパアプリケーションとして使われているプログラムを使って，オーバーヘッドを見積もる実験を行なった．使用したプログラムは，Acrobat Reader, Ghostscript の 2 つである．

我々の方式では，アプリケーションが発行するシステムコールをチェックするため，システムコールの発行頻度に比例してオーバーヘッドがかかる．そこでアプリケーションの実行時間と，その間に発行されるシステムコールの回数を計測し，計算によりオーバーヘッドの大きさを見積もった．

オーバーヘッドの値は，以下のようにして計算した．アプリケーションがシステムコールを発行すると，そのたびにポリシーモジュールを呼び出すための細粒度保護ドメイン切り替えが発生する．従って，アプリケーションの実行時間を T[秒]，アプリケーションが発行するシステムコールの回数を n[回]，細粒度保護ドメイン切り替えにかかる時間を t_{sw}[秒] とすると，オーバーヘッドは以下の式で表せる．

$$overhead = \frac{n \times t_{sw}}{T} \quad (1)$$

実験の方法は以下の通りである．まず，Acrobat Reader では，PDF 形式のファイル（PDF ファイル）を Postscript 形式に変換するのにかかる時間と，そのあいだに発行するシステムコールの回数を計測した．誤差を減らすために，PDF から Postscript への変換はコマンドラインから非対話的に行なった．また，ディスクアクセスの影響を減らすために，書き込み先のファイルは /dev/null とした．使用したファイルは，大きさが 1.6KByte から 8.1MByte までのランダムに選択した PDF ファイル 193 個である．使用した Acrobat Reader のバージョンは 4.05 である．

実験結果を図3に示す．PDF ファイルの内容によってオーバーヘッドの大きさにばらつきがあるが，およそ 0.15 % から 2.5 % 程度の範囲に抑えられていることが分かる．オーバーヘッドの平均はおよそ 1.0% であった．

次に，Ghostscript での実験を行なった．Ghostscript では Postscript 形式のファイル（PS ファイル）を解釈して表示するのにかかる時間と，そのあいだに発行するシステムコールの回数を計測した．誤差を減らすために，実行はコマンドラインから非対話的に行ない，ページの最初から最後まで一気に表示させた．使用したファイルは，大きさが 112 Byte

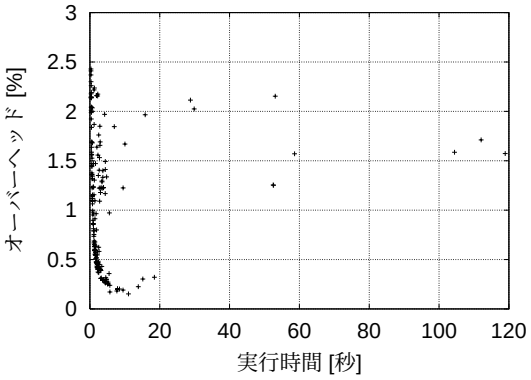

図 3: オーバーヘッド (Acrobat Reader)

から 5MB までのランダムに選択した PS ファイル 181 個と，175MB のファイル 1 個である．使用した Ghostscript のバージョンは 5.50 である．

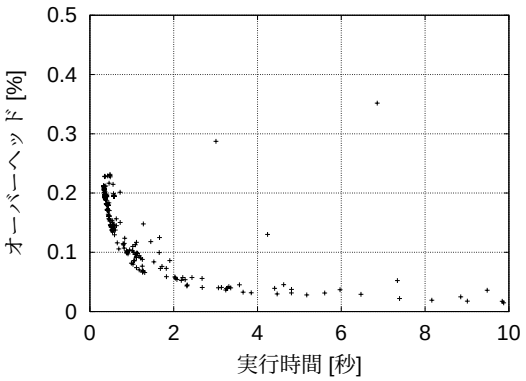

図 4: オーバーヘッド (Ghostscript)

実験結果を図3に示す．オーバーヘッドは 0.015 % から 0.3 % 程度の範囲に抑えられていることが分かる．大きさが 175MB のファイルは，オーバーヘッドが最も大きくなっている（実行時間 6.9 秒，オーバーヘッド 0.35%）．これはページのほとんどがビットマップイメージで構成されているファイルで，サイズが大きいので他の PS ファイルと比べて read システムコールが頻繁に発行されるためである．

以上の実験結果により，この手法による保護のオーバーヘッドは平均で 0.1〜1% 程度，最大でも 0.35〜2.5%程度に抑えられることが分かった．

5 関連研究

ヘルパアプリケーションを安全に実行する仕組みとしては，Janus [7] がある．Janus では，Solaris のシステムコールトレース機能を利用して，監視プロセスがヘルパアプリケーションを子プロセスとして実行し，ヘルパアプリケーションが発行するシステムコールをチェックしている．Janus はシステムコールの監視を別プロセスで行なっており，システムコールを発行するたびにプロセス切り替えが発生するので，保護のオーバーヘッドが大きくなる．

アプリケーションのアクセス権限を細かく制御できる仕組みとしては，Java アプリケーション [14] がある．Java は安全性を確保するために，型安全な言語の使用や，バイトコードベリファイアによるコードの検証，Java 仮想マシンによる実行時チェックなど，言語処理系としての機能を利用している．しかし Java による方式は，バイトコードと呼ばれる中間コードを Java 仮想マシンで解釈しながら実行するため，ネイティブコードに比べると実行性能が劣る．JIT コンパイラなどの高速化技術を用いても，ネイティブコードに匹敵するまでにはいたっていない．

オペレーティングシステムに依存しない方式でプロセス内に保護ドメインを形成する仕組みとしては，SFI(Software Fault Isolation) [15] や PCC(Proof Carrying Code) [12, 11] などがある．SFI ではバイナリコードを直接改変してアクセスチェックのための命令を挿入する．また，PCC は，バイナリコードに添付された証明を静的に検証することにより，実行時のチェックを行なわずに安全性を保証する．

オペレーティングシステムの機能としてプロセス内に保護ドメインを形成する仕組みとしては，Palladium [5] や PSL (Protected Shared Libraries) [3] などがある．Palladium は Intel CPU のリング保護機構を利用して，プロセス内に 2 段階の保護ドメインを形成する．PSL は POWER プロセッサ上でアドレス空間を部分的に切り替えて，共有ライブラリのデータを安全に共有する．

これらのプロセス内の保護ドメインは，主にメモリ保護のみを目的としたものであり，ファイルなどの資源に対するアクセス制限は想定されていない

従来の保護ドメインであるプロセスの切り替えを高速化する仕組みとしては，さまざまな試みがなされている (LRPC [4], Spring [8], Mach [6], L4 [9, 10]). しかしプロセスによる保護ドメインでは，オペレー

ティングシステムの資源に対して細かなアクセス制限を行なうことはできない.

6 まとめ

本論文では，ヘルパアプリケーションを安全に実行できる環境を実現する手法を提案した．ヘルパアプリケーションを利用した攻撃によって被害を受ける可能性を減らすために，ユーザのファイルなどの資源に対するアクセス制御を行なって，ヘルパアプリケーションが動作するために必要最小限の資源以外へのアクセスを禁止できるようにした．アクセス制御を実現するためには，我々が提案・開発中のSeeMoc オペレーティングシステムで提供する細粒度保護ドメインを利用した．既存のアプリケーションを改変せずに保護を実現するために，ELF ローダを改造して細粒度保護ドメインの作成・割り当てや，参照モニタの機能を果たすポリシーモジュールの挿入などの処理を行なうようにした．実際のアプリケーションを用いて保護によるオーバーヘッドを見積もる実験を行なった結果，そのオーバーヘッドは 0.1%〜1% 程度に抑えられることを確認した.

参考文献

[1] CERT Advisory CA-1995-10: Ghostscript Vulnerability, August 1995.

[2] SPS Advisory #39: Adobe Acrobat Series PDF File Buffer Overflow, July 2000.

[3] Arindam Banerji, John Michael Tracey, and David L. Cohn. Protected Shared Libraries - A New Approach to Modularity and Sharing. In *Proc. of theUSENIX 1997 Annual Technical Conference*, pp. 59–75, October 1997.

[4] Brian N. Bershad, Thomas E. Anderson, Edward D. Lanzowska, and Henry M. Levy. Lightweight Remote Procedure Call. *ACM TOCS*, Vol. 8, No. 1, pp. 37–55, February 1990.

[5] Tzi-cker Chiueh, Ganesh Venkitachalam, and Prashant Pradhan. Integrating Segmentation and Paging Protection for Safe, Efficient and Transparent Software Extensions. In *Proc. of the 17th ACM Symposium on Operating System Principles (SOSP '99)*, pp. 140–153, December 1999.

[6] Bryan Ford and Jay Lepreau. Evolving Mach 3.0 to a Migrating Thread Model. In *Proc. of theUSENIX Winter 1994 Technical Conference*, January 1994.

[7] Ian Goldberg, David Wagner, Randi Thomas, and Eric A. Brewer. A Secure Enviroment for Unstrusted Helper Applications. In *Proc. of the 6th USENIX Security Symposium*, July 1996.

[8] Graham Hamilton, Michael L. Powell, and James G. Mitchell. Subcontract: A flexible base for distributed programming. In *Proc. of the 14th ACM Symposium on Operating System Principles (SOSP '93)*, pp. 69–79, December 1993.

[9] Hermann Härtig, Michael Hohmuth, Jochen Liedtke, Sebastian Schönberg, and Jean Wolter. The Performance of μ-Kernel-Based Systems. In *Proc. of the 16th ACM Symposium on Operating System Principles (SOSP '97)*, pp. 66–77, October 1997.

[10] Jochen Liedtke, Kevin Elphinstone, Sebastian Schönberg, Hermann Härtig, Gernot Heiser, Nayeem Islam, and Trent Jaeger. Achieved IPC Performance. In *Proc. of the 6th Workshop on Hot Topics in Operating Systems (HOTOS '97)*, pp. 28–31, May 1997.

[11] George C. Necula. Proof-Carrying Code. In *Proc. of the 24th ACM Symposium on Principles of Programing Languages (POPL '97)*, pp. 106–119, January 1997.

[12] George C. Necula and Peter Lee. Safe Kernel Extensions without Runtime Checking. In *Proc. of the 2nd Symposium on Operating System Design and Implementation (OSDI '96)*, pp. 229–243, October 1996.

[13] M. Takahashi, K. Kono, and T. Masuda. Efficient Kernel Support of Fine-Grained Protection Domains for Mobile Code. In *Proc. of the 19th IEEE International Conference on Distributed Computing Systems*, pp. 64–73, May 1999.

[14] Java Team, James Gosling, Bill Joy, and Guy Steele. *The Java[tm] Language Spectication*. Addison Wesley Longman, 1996. ISBN 0-201-6345-1.

[15] Robert Wahbe, Steven Lucco, Thomas E. Anderson, and Susan L. Graham. Efficient Software-Based Fault Isolation. In *Proc. of the 14th ACM Symposium on Operating System Principles (SOSP '93)*, pp. 203–216, December 1993.

[16] 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, June 1999.

[17] 品川高廣, 河野健二, 益田隆司. セグメント機構を用いた細粒度保護ドメインの性能分析. 情報処理学会研究会報告書,99-OS-82, pp. 17–24, August 1999.