Vol.2013-OS-124 No.9
2013/3/1

直感的ポリシー設定を可能にする動的な資源隔離機構

忠鉢 洋輔[1]　品川 高廣[2]　加藤 和彦[3]

概要：インターネットを経由した情報漏洩やマルウェア感染の被害が後を絶たない．その被害者は，PCやモバイルプラットフォームを使ってインターネットを利用している一般のユーザーである．彼らに対してアクセス制御技術の学習を求めたり，インターネット経由で入手したプログラムの動作を想定させることは現実的ではない．本研究では，ユーザーが直感的にプログラムとデータを隔離できるセキュリティモデルと，動的にそれを実施する動的な資源隔離機構を提案する．提案手法により，ユーザーが直感的にアクセス制御が連想できるプライバシーレベルを用いたポリシー設定をおこなうことが可能となる．さらに，動的な資源隔離機構がセキュリティモデルに沿った適切な隔離を実施し，情報流出やシステムの改ざんを防ぐ．提案手法に基づき，我々は OS による動的な資源隔離機構 SURIM（Simple and Usable Resource Isolation Mode）を設計し，実装をおこなった．

SURIM:Simple and Usable Resource Isolation Mode

YOSUKE CHUBACHI[1]　TAKAHIRO SHINAGAWA[2]　KAZUHIKO KATO[3]

Abstract: Desktop computer and modern mobile platforms are offen invaded by malware or attackers. These Devices widely spread around the world. Also many consumer are confronting various incident. Information leakages, system hijacking and more.You must attend lecture in access control and malware analysis to use the internet - its an impractical idea. In this paper, we propose usable security policy model against threat of potential of information leakage and system invasion over come the internet with dynamic resource isolation mechanism. Our approach be able to user-intended security configuration using privacy-level and domain name. We developed Simple and Usable Resource Isolation Mode(SURIM) based on this policy model and mechanism.

1. はじめに

インターネットを経由した情報漏洩やマルウェア感染の被害が後を絶たない．その被害者は，PC やモバイルプラットフォームを使ってインターネットを利用している一般のユーザーである．これらの脅威への対抗策として，一般的なクライアント環境においてはオペレーティングシステム（OS）のアクセス制御やアンチウィルスソフトウェアが広く利用されている．しかし，クライアント環境においては，ほとんどの場合はコンピューターの所有者が管理者を兼ねるため，アクセス制御ポリシーを管理者たるユーザーが決めなければならない．だが，アクセス制御ポリシーは OS の動作や OS 特有の表現，例えばプロセスやパーミッションについての知識が必要となる．このため，専門家ではない一般ユーザーが OS のアクセス制御を使いこなすことは難しいとされている [1].

マルウェアについても，アンチウィルスソフトウェアですべてのマルウェアを防ぐことは難しくなっている．例え

[1] 筑波大学大学院 システム情報工学研究科 コンピュータサイエンス専攻
Department of Computer Science, Graduate School of Systems and Information Engineering, University of Tsukuba
[2] 東京大学 情報基盤センター
Information Technology Center, The University of Tokyo
[3] 筑波大学 システム情報系 情報工学域
Division of Information Engineering, Faculty of Engineering, Information and Systems, University of Tsukuba

ば，無害なプログラムを装ってユーザー自身にインストールさせるマルウェアによって，システム侵害が簡単におこなわれてしまう．そもそも専門家ではない一般のユーザーにとっては，インターネット経由で入手したプログラムの動作を想定することは困難である．実際に，この手法を用いてシステムを掌握し，遠隔操作をおこなうような悪意のあるプログラムを用いた事件も発生している．

クライアント環境において情報漏洩を防ぐ手法や，マルウェアによるシステム侵害を防ぐ手法については多くの研究がなされている [2][3][4][5]．強制アクセス制御（MAC：Mandatory Access Control）[6] やアプリケーションサンドボックス [7] の研究では，プロセス単位でセキュリティポリシーを記述してプロセスの動作を制限する．だが，MAC のセキュリティポリシー記述は，OS に詳しいサーバ管理者でも難しいタスクであることもまた知られている [1]．近年は設定ツールの充実が図られているものの，一般のユーザーが使いこなすことは到底困難である．また，強制アクセス制御よりも細かい粒度で情報の制御をおこなう手法として，Information Flow Control（IFC）[8][9] がある．IFC では，プログラムが読み出した情報をどのように扱うかを細かく設定し，OS やプログラミング言語のランタイムがこの情報の流れを細かく制御することが可能になる．しかし，IFC をクライアント環境に適用するためには，既存の実行環境に対する大幅な変更と，厳密なポリシーを記述しなければならない．これらの手法では，アクセス制御のプリミティブがユーザーにとってわかりにくいという点と，既存の実行環境に対して大幅な変更が必要である点が課題となっている．

本研究では，ユーザーが直感的に理解できるセキュリティポリシーとセキュリティモデル，これらに基づいて動的な資源隔離を実施する機構を提案する．動的な資源隔離機構は，ファイルに対して設定されたプライバシーレベル (Privacy-level）と出自属性（Origin）を参照し，プロセスがアクセスできるファイルを動的に制限する．プライバシーレベルはユーザーによって設定され，プライベート（private）か公開（public）の２つのプリミティブからなる．この２つは，読み取りに関するプリミティブを提供し，プライベートの場合は，そのファイルがネットワークに流出することを拒否し，公開の場合は，ネットワークやそのファイルの出自とは異なる出自のプログラムからも読み取り可能であることを表している．

これと同時に，ユーザーではなくシステムが自動的に割り当てる出自属性を用いたアクセス制御を実行する．出自属性はプログラムやファイルの入手元，生成元の DNS ドメイン名を用いる．異なる出自属性を持つ場合，お互いの完全性を侵害するような振る舞いを資源隔離機構が阻止する．このように，システムやアプリケーションの完全性を，ユーザーに対して意識させることなく保護する．

本研究で提案する OS による動的な資源隔離機構 SURIM（Simple and Usable Resource Isolation Mode）は，プロセスがアクセスしたファイルやネットワークアクセス状況からプロセスの隔離をおこなう．SURIM はプロセスがアクセスしたファイルのセキュリティ属性を逐一追跡しプロセスの動作を制限する．例えば，プライバシーレベルが private なファイルを開いているプロセスが，ネットワークに接続するような動作をしたとき，その接続を拒否する．このように，ファイルの公開範囲を連想させる属性と，SURIM による動的な資源隔離により，ユーザーにとって直感的なポリシー設定を可能とする．

2. 脅威モデルとセキュリティポリシー

本章では，本研究で想定する脅威モデルと，ユーザーに対して提供するセキュリティポリシー，動的な資源隔離機構がおこなうアクセス制御のモデルについて説明する．

2.1 脅威モデル

本研究では次の二つの脅威を想定する．

(1) インターネットへの情報流出（read-sensitive case）
(2) 信頼できない出自のプログラムがシステムに対しておこなう改ざん（write-sensitive case）

インターネットへの情報流出が発生するモデルケースとして，**ユーザーの過失による流出**と**信頼できないアプリケーションによる流出**を仮定する．

ユーザーの過失による流出の例としては，アプリケーションの出自や通信状況に関わらず，E メールの添付ミス，ブラウザやファイラーでのファイル選択ミスなどが挙げられる．既存のアクセス制御では，ユーザーが，個々のファイルの生成，あるいは編集段階においては流出が好ましくないと判断できていても，ネットワーク接続をおこなうアプリケーションで意図せずに送信してしまうことがある．

信頼できないアプリケーションによる流出の例としては，出所の不確かなプログラムや信頼できないアプリケーションを実行したときに，システム内のデータをネットワーク越しに送信してしまう事例が挙げられる．ユーザーには，インターネットからダウンロードしたプログラムがどのように振る舞うか知ることができないため，このような脅威が考えられる．

信頼できない出自のプログラムに対する改ざんの例として， システムやインストール済みのアプリケーションのデータを改ざんするマルウェアがある．これらはインターネット掲示板やインターネットストレージサービスなどから自動的に，あるいは偽アンチウィルスソフトウェアなどの形でダウンロードされ，クライアント環境上で実行される．既存の対策でこの実行を阻止することが出来ない場合，再起動後でも自身を起動するようにシステムを改ざんしたり，アンチウィルスソフトウェアの設定を変更すると

いったシステムの完全性を脅かす攻撃が成立してしまう．

2.2 セキュリティポリシー

ユーザーに対して直感的な 2 つのポリシー記述プリミティブを提供する．一つはファイルに対して設定する**プライバシーレベル**，もう一つはシステム全体で共有する**信頼できるドメインリスト**である．

プライバシーレベルにはあらかじめ用意した 2 つのプリミティブを提供する．

プライベート（private） インターネットに送信されることはない

公開（public） インターネットに対して公開する

例えば，教員であるユーザーが試験問題を文書ファイルとしてファイラーから作成し，このファイルを過失により流出することを防ぎたい場合を考える．ユーザーは直感的に，このファイルがプライベートであると考え，ファイルに対してプライバシーレベルを設定する．この文書ファイルには localhost という出自属性が付き，もし，ブラウザから過失により信頼できないドメインのファイル共有サービスに対してアップロードを試行してしまったとき，この時点で，ブラウザは private なファイルにアクセスできないため，流出は起こりえない．このように情報流出を直感的な設定で防ぐことができる．

信頼できる出自のドメイン名リストには，信頼できるドメインを設定する．現在はワイルドカードなどは想定せず，完全な DNS ドメイン名を仮定している．

3. 直感的なポリシー設定を可能にするセキュリティモデル

提案するセキュリティモデルは，プライバシーレベルと出自属性からなるセキュリティドメインを基礎とする．セキュリティドメイン内はプライバシーレベルに基づく機密性を保護するモデルに沿ったアクセス制御をおこなう．また，セキュリティドメイン間は出自に基づく完全性を保護するモデルに沿っている．またアクセス制御におけるサブジェクトのセキュリティドメイン間の遷移についてもここで説明する．

3.1 セキュリティドメイン

本研究では，直感的なアクセス制御を実現するため，セキュリティドメインという概念を導入する．セキュリティドメインは，プライバシーレベルと出自属性の 2 つの属性で決定される，サブジェクトやオブジェクトのセキュリティ上の隔離単位である．プライバシーレベルはユーザーが決定し，出自属性は OS がオブジェクト（主にファイル）に対して自動的に付与するものとする．この出自属性は基本的にユーザーが介在する余地がない．このセキュリティドメインが同一の場合は，読み込み書き込み，実行，及び出自属性へのネットワークアクセスのすべてが許可される．このアクセス制御は Web ブラウザ上で実行されるスクリプト言語のアクセス制御の一つである Same Origin Policy を参考としている [10].

プライバシーレベルは 3 つのレベルを用いる．

Private このセキュリティドメイン以外から読み込むことを拒否

Public すべてのセキュリティドメインから読み込むことが可能である

Neutral 特定のセキュリティドメインは読み込むことができる

特に指定しない限り，オブジェクトはすべて Neutral のプライバシー属性と見なす．

3.2 セキュリティドメインの遷移

セキュリティドメインが異なるアクセスについては，属性によるアクセスの制限，またセキュリティドメインの遷移がおこなわれる．また，サブジェクトはオブジェクトへのアクセスによってセキュリティドメインを遷移する．これは，Chinese wall ポリシー [11] に沿った振る舞いで，異なるセキュリティドメインに一度遷移したサブジェクトは，それ以上遷移することができない．これによって，サブジェクトの振る舞いを大きく制限する．

3.3 異なる出自間のアクセス制御

異なる出自間においては，LOMAC[12] モデルを参考にした，完全性に関するアクセス制御をおこなう．異なる出自間において，他の出自に対する書き込みは禁止される．また，セキュリティドメインを遷移を厳密におこなうと，アプリケーションのライブラリ読み込みや異なるドメインに属する設定ファイルなどの読み込みができなくなってしまう．このため，異なる出自間で，ドメイン遷移をおこなわずにアクセスを許可する**信頼された出自**属性を追加する．異なる出自間においても，信頼された出自属性が同一であれば遷移をおこなわずに読み込みがおこなえる．また，信頼された出自属性から信頼されていない出自属性に対するドメイン遷移は一方的に許可されるが，その逆は不可能とする．これは LOMAC モデルに対応させると，信頼された出自は高位のインテグリティレベルであり，そうでない出自属性を持つ場合は低位のインテグリティレベルと位置付けることができる．

4. SURIM:Simple and Usable Resource Isolation Mode の設計

SURIM（Simple and Usable Resource Isolation Mode）は，本研究で提案する直感的なポリシー設定を可能にするセキュリティモデルに基づき，動的な資源隔離機構を実現するシステムである．

			File					
			Public		neutral		Private	
			Untrusted	Trusted	Untrusted	Trusted	Untrusted	Trusted
Process	Public	Untrusted	**rwx**	**r**	**r**	**r**	-	-
		Trusted	**rw**	**rwx**	**r**	**rw**	-	-
	Neutral	Untrusted	**T**	-	**rwx**	**r**	**T**	-
		Trusted	**T**	**T**	**T**	**rwx**	**T**	**T**
	Private	Untrusted	-	-	**r**	**r**	**rwx**	-
		Trusted	-	-	**r**	**r**	**r**	**rwx**

r:read, w:write, x:execute, T:domain transition

図 1 アクセス制御マトリックス
Fig. 1 Access Control Matrix

本章では，この SURIM の設計について述べる．

4.1 SURIM の概要

本研究で提案する動的な資源隔離機構である SURIM は，ファイルへのアクセスすべて漏れなくチェックするため，OS カーネルレベルで隔離をおこなう．アクセス制御は強制アクセス制御における標準的なサブジェクトの単位であるプロセス単位でおこなう．また，アクセス対象となるオブジェクトの粒度はファイル単位であり，本研究では，ファイルアクセスのみを監視している．IPC（Inter Process Communication）や共有メモリについては今後，設計と実装をおこなう予定である．アクセス権限は読み込み，書き込み，実行の 3 つを基本とする．なお，本研究では既存のプログラムを対象にプロセスレベルの隔離をおこなうが，ソースコードレベルで自己区画化（self-compartmentalization）[13] されたプログラムでは，管理プロセスと外部からの情報を扱う処理プロセスに分割されているため，本研究による隔離が非常に有効である．

4.2 セキュリティドメイン

提案する資源隔離機構では，プロセスやファイルに対して，資源隔離機構が管理するプライバシーレベルと出自属性を割り当てる．この 2 つの属性によって決定するセキュリティドメインが，隔離の基本単位である．以後，このセキュリティドメインは {domain name}#{privacy level} と記述する．例えば，www.osss.cs.tsukuba.ac.jp#public とは，http://www.osss.cs.tsukuba.ac.jp/で公開されているファイルのことを指す．プロセスとファイルが同一のセキュリティドメインにある場合，読み書き実行のすべてが自由におこなえる．また，その出自に対してネットワーク接続することも自由にである．

4.3 プロセスのセキュリティドメイン遷移

セキュリティドメインの遷移は，基本的にプライバシーレベルが neutral であるプロセスしかおこなうことができない．もしプロセスが private, public なセキュリティドメインに遷移した場合，ネットワーク接続が制限され，そのプロセスのセキュリティドメインが固定される．これにより，プライバシーレベルが高いファイルを読み込んだプロセスは，情報を送信する先を自由に選ぶことができない．

例として，プロセス 1 がファイル A，B，C にアクセスするときのことを考える．なお，この例では localhost, と www.osss.cs.tsukuba.ac.jp を信頼できるドメインと設定したとする．

プロセス 1 のセキュリティドメイン
www.osss.cs.tsukuba.ac.jp#neutral
ファイル A のセキュリティドメイン localhost#private
ファイル B のセキュリティドメイン www.ecc.u-tokyo.ac.jp#public

プロセス 1 はファイル A に対してアクセスするとき，そのドメインが．localhost#private に遷移する．この際，プロセス 1 はネットワーク接続を localhost に限定されるため，ファイル A の情報がインターネット上に流出することはない．ファイル B に対してアクセスをおこなったとき，プロセスのセキュリティドメインが www.ecc.u-tokyo.ac.jp#public に遷移し，アクセス制御がおこなわれる．ファイル B に対しては，読み込み，と書き込みと実行をおこなうことができる．3 章で定義したセキュリティモデルに基づき，セキュリティドメイン間のアクセス制御についてまとめたのが図 1 である．

5. SURIM の実装

本章では，Linux を対象とした SURIM（Simple and Usable Resource Isolation Mode）の LSM 実装について説明する．SURIM は特定のシステムコールをフックし，アクセス制御を実施する LSM である．SURIM はアクセス制御のためのフックに関する surim_lsm モジュールと，アクセス制御と属性の管理をおこなう surim_access モジュール，

信頼できるドメインを設定及び管理するための surim_fs からなる．なお，SURIM は SMACK[14] の実装を参考にしている．開発環境は OS に Fedora Core17，ファイルシステムは ext4 を用いた．また，Linux カーネルはバージョン 3.6.10-2 をベースに実装している．

5.1 ファイルのセキュリティ属性

SURIM では，ファイルのセキュリティ属性は xattr に格納する．サポートするサイズは 256 バイトであり，xattr の名前空間は security.surim_privacy がプライバシーレベル，security.surim_origin が出自属性を保持する．なお，プライバシーレベルはユーザーが自由に変更できることが望ましいが，現在はプログラムから変更されることを防ぐため，CAP_MAC_ADMIN を持つプロセスからのみ変更できるように制約を設けている．

5.2 LSM を用いたシステムコールのフック

SURIM では現在，アクセス制御のために open，execve，connect システムコールをフックし，プロセスのファイルオープンと実行を隔離する．ネットワークアクセスの制限については検討中である．出自属性に設定されているドメイン名に限定したアクセスを実現するためには，カーネル内で DNS 名前解決を行う必要があるが，こちらは検討中である．ファイルアクセスや execve によるセキュリティドメイン遷移は，プロセスごとにセキュリティドメインを管理する構造体を持ち，その中でプライバシーレベルと出自属性を管理することで実現している．プライバシーレベルと出自属性はすべてのファイルとプロセスに付くため，メモリ空間の節約のためにカーネル内部で文字列として確保し，同一の属性の場合は，その文字列へのポインタを保持する．また，ファイルシステムの xattr 領域にセキュリティ属性を格納している．このため，setxattr システムコールをセキュリティ属性の保護のためにフックし，特権を持たないユーザーがセキュリティ属性を変更してしまうことを防いでいる．

5.3 audit フレームワークを用いた監査

Linux カーネルには，audit フレームワークという，アクセス制御など監査を統一的に扱うフレームワークが存在する．SURIM もこれに対応し，open, execve, connect 時にアクセス制御に違反するアクセスがおこなわれた場合，audit フレームワークを通じて監査ログを出力する．

5.4 surimfs

surimfs は，linux の securityfs インターフェースを実装した，セキュリティモジュールとユーザー空間を繋ぐ疑似ファイルシステムである．surimfs は信頼できるドメイン名を/surim/load に対して入力することで，カーネル内に信頼できるドメイン名を送り込む．現在の実装では，アプリケーションからの変更を防ぐため，CAP_MAC_ADMIN を持つプロセスからのみ，変更が可能である．

6. 関連研究

一般ユーザーの過失による情報流出を防ぐ研究として，TightLip[2] がある．TightLip では，Doppelganger プロセスという仕組みを用いて漏洩を防止する．Doppelganger プロセスは，コピー元となるプロセスがセンシティブデータの読み込みがおこなわれた後にネットワークアクセスをおこなうとき，コピー元プロセスとすり替わり，センシティブデータの代わりに個人情報をマスクされた代替データをネットワークに送信する．TightLip のメカニズムは，プロセスとファイルを二重化し，Doppelganger プロセスが親プロセスの状態に追従していく点が大きな特徴である．このメカニズムにより，TightLip は既存のアプリケーションへの変更をせずに，Information Flow Control）[8][9] を実現している．ただ，TightLip では機密性のみを守る仕組みであり，ネットワーク外から来たデータの振る舞いについて特に制限されない．

UMIP[3] はネットワーク外からシステムやアプリケーションを改ざんする攻撃を OS で防ぐ仕組みである．UMIP はシンプルな概念で完全性の保護を目指す．プロセスには High と Low の 2 つの Integrity-Level があり，Low の場合は振る舞いを制限される．プロセスは自身が生成される際の親プロセスからこの Integrity-Level を受け継ぎ，また，ネットワークアクセスの状態によって Integrity-Level が変化する．ただし，UMIP はアプリケーションやシステムの完全性を保護することが目的であり，情報流出を防止する仕組みにはなっていない．

また，近年のモバイルプラットフォームにおけるユーザーに対してアクセスの許可を求める仕組みは，直感的なポリシー設定という観点から本研究と比較する必要がある．Android[15] におけるパーミッションは，ユーザーに対してアプリケーションが使う機能を提示する．しかし，Android のパーミッションは，大半のユーザーに取って求める機能の意図が分からないため，直感的とはいい難い [16]．また，iOS[17] では，位置情報やコンタクトリストに対してアプリケーションがアクセスする際に prompt を表示し，許可を求める．これはくり返し許可を求めることで，ユーザーに対して慣れを感じさせ，結果としてユーザーの意図と反する許可がおこなわれることが考えられる．これらの問題に対して，User-driven Access Control（UDAC）[4] という手法が提案されている．UDAC はユーザーの意図を明確に反映するために，ユーザーインターフェースのデザインと機能を結びつけること，クリックジャッキングなどに対応するためにカーネルレベルの入力監視をおこなうことを提案している．本研究における直感的なポリシー設定を可能

とする動的な資源隔離機構は，これらのプラットフォームに対しても適用可能である．かつ，セキュリティポリシーはユーザーに対して正しく意図が伝わるような分かりやすさを提供している．ただし本研究では，直感的であるという点について適切な評価をおこなえていない．今後，直感的であるという点について適切な実験をおこない，その妥当性を検証することも検討している．

また，強制アクセス制御の発展系として Information Flow Control（IFC）[8][9] がある．IFC では，プログラムが読み込んだデータの流れをセキュリティポリシーに沿って逐一チェックする．例えばネットワーク経由で入力されたデータについて，そのデータをクライアントで実行することを防ぐことができる，言語レベル [18]，ランタイムレベル [19]，OS レベル [5][20] など，数々のプラットフォームで提案がなされている．これらは多くの場合，情報の流れを扱うという点においてセキュリティポリシーが複雑であり，また，既存のプログラムついて改編が必要などの問題点がある．

OS のアクセス制御レベルでの IFC の発展系として，Decentralized Information Flow Control[21] に基づく，プロセスの自発的隔離機構が提案されている． 例えば，Capsicum[13] は FreeBSD においてプログラムの特権分割を実現するメカニズムである．Capsicum では，自身をサンドボックス化するシステムコールの追加や，名前空間の隔離するという仕組みによってこれを実現する．本研究では，プロセスのセキュリティ属性を切り替えるというシンプルな実装によって動的な資源隔離機構を実現しているが，プロセス間通信や共有メモリなどについてはまだ考慮されていない．Capsicum などの OS による隔離機構の手法についてよく検討し，本研究で提案するセキュリティモデルに合わせて導入することを考えている．

情報漏洩やシステムの完全性を侵害する攻撃から OS を守る手法として，強制アクセス制御メカニズムが多く提案されている．しかし，SELinux [6] や AppArmor[7] に代表される強制アクセス制御メカニズムは，そのセキュリティポリシー記述が難しいことが知られている [22][1]．特に，クライアント環境においては，ほとんどの場合コンピューターの所有者が管理者を兼ねるため，セキュリティポリシーを管理者たるユーザーが決めなければならない．だがこれらのメカニズムにおいて，セキュリティポリシーを記述するにはプロセス生成の仕組みやファイルシステム，IPC など OS について詳しい知識が必要である．このため，一般ユーザーが自身のプライベートなファイルなどに対して適切なラベルを設定し，それについてポリシーを記述することは更に困難であるといえる．

7. まとめと今後の課題

ユーザーが直感的に理解できるセキュリティポリシーとセキュリティモデルを提案し，それを実現する SURIM の設計と実装について述べた．動的な資源隔離機構である SURIM（Simple and Usable Resource Isolation Mode）は，プライバシーに関してプライベート（private）か公開（public）の 2 つのポリシーによって保護を決定する．プライベートの場合は，そのファイルがネットワークに流出することを拒否し，公開の場合は，ネットワークやそのファイルの出自とは異なる出自のプログラムからも読み取り可能であることを表しており，インターネット経由での情報流出を防ぐ．また，システムの完全性については，システムが自動的に割り当てる出自属性を用いた保護モデルを提案した．出自属性はプログラムやファイルの入手元，生成元の DNS ドメイン名を用いる．異なる出自属性を持つ場合，お互いの完全性を侵害するような振る舞いを SURIM が阻止する．これによりユーザーに対して意識させることなく，システムや個々のプログラムの完全性を保護する．これに基づいた OS レベルのアクセス制御として SURIM の設計を示した．

SURIM の実装は LSM を用いておこなった．プロセスがアクセスしたファイルの属性をプロセスと対応付けて逐一追跡し，動的にプロセスの振る舞いを制限する．このように，ファイルの読み取り権限を連想させる属性と，資源隔離機構による動的な資源隔離によってユーザーに取って直感的なアクセス制御ポリシーが可能とした．

今後の課題として，SURIM の実装の完成度を上げることが挙げられる．現状はごく最低限の実装となっており，アプリケーションの動作テストを十分におこなえていない．また，脅威モデルに従ったセキュリティ分析をおこない，本研究で提案する動的な資源隔離機構の制約を明らかにする必要がある．また，セキュリティモデルについては，形式的証明などを用いてその矛盾がないか検証をおこなうことも視野に入れている．

謝辞

本研究は JSPS 特別研究員奨励費 201202549 の助成を受けたものです。

参考文献

[1] Das, T., Bhagwan, R. and Naldurg, P.: Baaz: a system for detecting access control misconfigurations, *USENIX Security'10: Proceedings of the 19th USENIX conference on Security*, USENIX Association (2010).

[2] Yumerefendi, A. R., Mickle, B. and Cox, L. P.: Tightlip: keeping applications from spilling the beans, *NSDI'07: Proceedings of the 4th USENIX conference on Networked systems design & implementation*, USENIX Association (2007).

[3] Li, N., Mao, Z., Security, H. C. and Privacy, . S. . I. S. o.: Usable Mandatory Integrity Protection for Operating Systems, *Security and Privacy, 2007. SP '07. IEEE Symposium on*, pp. 164–178 (2007).

[4] Roesner, F., Kohno, T., Moshchuk, A., Parno, B., Wang,

H. J. and Cowan, C.: User-Driven Access Control: Rethinking Permission Granting in Modern Operating Systems, *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, Washington, DC, USA, IEEE Computer Society, pp. 224–238 (online), DOI: 10.1109/SP.2012.24 (2012).

[5] Zeldovich, N., Boyd-Wickizer, S., Kohler, E. and Mazières, D.: Making information flow explicit in HiStar, *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, OSDI '06, Berkeley, CA, USA, USENIX Association, pp. 19–19 (online), available from ⟨http://dl.acm.org/citation.cfm?id=1267308.1267327⟩ (2006).

[6] Loscocco, P.: Integrating flexible support for security policies into the Linux operating system, *Proc. 2001 USENIX Annual Technical ...* (2001).

[7] Cowan, C., Beattie, S., Kroah-Hartman, G., Pu, C., Wagle, P. and Gligor, V.: SubDomain: Parsimonious Server Security, *LISA '00: Proceedings of the 14th USENIX conference on System administration*, USENIX Association (2000).

[8] Bell, E. D. and La Padula, J. L.: Secure computer system: Unified exposition and Multics interpretation (1976).

[9] Myers, A. C. and Liskov, B.: Protecting privacy using the decentralized label model, *ACM Trans. Softw. Eng. Methodol.*, Vol. 9, No. 4, pp. 410–442 (online), DOI: 10.1145/363516.363526 (2000).

[10] Moshchuk, A., Wang, H. J. and Liu, Y.: Content-Based Isolation: Rethinking Isolation Policy in Modern Client Systems (2012).

[11] D.F.C.Brewer and M.J.Nash: The chinese wall security policy, *In Proceedings of IEEE Symposium on Security and Privacy*, pp. 206–214 (1989).

[12] Fraser, T.: LOMAC: Low Water-Mark Integrity Protection for COTS Environments, *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, SP '00, Washington, DC, USA, IEEE Computer Society, pp. 230– (online), available from ⟨http://dl.acm.org/citation.cfm?id=882494.884406⟩ (2000).

[13] Watson, R. N. M., Anderson, J., Laurie, B. and Kennaway, K.: Capsicum: practical capabilities for UNIX, *USENIX Security'10: Proceedings of the 19th USENIX conference on Security*, USENIX Association (2010).

[14] Schaufler, C.: Smack in embedded computing (2008).

[15] Google: Google Android, (online), available from ⟨http://www.android.com/⟩.

[16] Felt, A. P., Ha, E., Egelman, S., Haney, A., Chin, E. and Wagner, D.: Android permissions: user attention, comprehension, and behavior, *Proceedings of the Eighth Symposium on Usable Privacy and Security*, SOUPS '12, New York, NY, USA, ACM, pp. 3:1–3:14 (online), DOI: 10.1145/2335356.2335360 (2012).

[17] APPLE: Apple iOS, (online), available from ⟨http://www.apple.com/ios/⟩.

[18] Seo, J. and Lam, M. S.: InvisiType: Object-Oriented Security Policies, *Proceedings of the Network and Distributed System Security Symposium, NDSS 2010, San Diego, California, USA, 28th February - 3rd March 2010*, The Internet Society, (online), DOI: http://www.isoc.org/isoc/conferences/ndss/10/pdf/16.pdf (2010).

[19] Welch, I. and Stroud, R. J.: Using reflection as a mechanism for enforcing security policies on compiled code, *J. Comput. Secur.*, Vol. 10, No. 4, pp. 399–432 (online), available from ⟨http://dl.acm.org/citation.cfm?id=773069.773074⟩ (2002).

[20] Efstathopoulos, P., Krohn, M., VanDeBogart, S., Frey, C., Ziegler, D., Kohler, E., Mazières, D., Kaashoek, F. and Morris, R.: Labels and event processes in the asbestos operating system, *SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles*, ACM Request Permissions (2005).

[21] Myers, A. C. and Liskov, B.: A decentralized model for information flow control, *Proceedings of the sixteenth ACM symposium on Operating systems principles*, SOSP '97, New York, NY, USA, ACM, pp. 129–142 (online), DOI: 10.1145/268998.266669 (1997).

[22] Nakamura, Y., Sameshima, Y. and Tabata, T.: SEEdit: SELinux security policy configuration system with higher level language (2009).