2005－OS－100（1）
2005／8／3

オープン処理に着目した情報拡散追跡法

箱守　聰[†‡]　横山　和俊[†‡]　乃村　能成[‡]　谷口　秀夫[‡]

機密性の高い情報が漏えいすることを防止するため、オペレーティングシステムが提供するファイルのオープン処理に着目し、機密情報が計算機内に拡散する経路を追跡する手法を提案する。情報が拡散する経路を明らかにするとともに、拡散状況を追跡する機構について述べる。また、Linux に提案手法を実装し、基本的な処理オーバーヘッドの評価結果を報告する。

Classified Information Diffusion Tracing Triggered by Open System Call

Satoshi HAKOMORI[†‡], Kazutoshi YOKOYAMA[†‡], Yoshinari NOMURA[‡] and Hideo TANIGUCHI[‡]

Abstract

Recently, leakage of classified information such as personal information has become serious problem. To prevent the leakage, it is important to know where the information diffuses in the computer environment. In this paper, we propose the tracing method of the classified information diffusion. The tracing starts when a process invokes the open system call for the file, and checks if whole or a part of the classified information may be copied to the opening file. We describe the algorithm of the method, Implementation of the method on the Linux kernel, and the overhead to the system calls.

1. はじめに

情報システムは、今や社会の重要インフラとなり、いつでも誰もが安心して利用できるために高い信頼性を確保することが求められている。近年、これらの情報システムが広域ネットワークに接続され様々な情報が流通するようになり、個人情報や企業情報などの秘密性の高い情報（機密情報）を取り扱う機会が増加している。これらの情報には高い機密性が求められており、その一部でも外部へ漏えいすると情報を管理する企業にとって大きな損失になる。特に、個人情報の場合は対象となる個人のプライバシーが侵害されることになり、社会的にも影響が大きい。このため、これらの情報の機密性を保持することが情報システムの信頼性を確保する上で重要である。

最近の情報漏えいの事例についてその原因を分類すると、計算機の操作ミスや設定ミスによる不注意から起こる場合が最も多く、次に計算機の管理者や利用者が故意に情報を持ち出す場合である。また、外部からネットワークを介し計算機へ不正にアクセスしたり、ウイルスやワームが計算機に侵入したりすることで情報が持ち出される事例もある。このような情報の漏えいを防止するには、情報システム上に情報が容易に漏えいし

†（株）NTT データ　技術開発本部
‡岡山大学大学院自然科学研究科

ないための仕組みを実現するとともに、情報システムにおける情報取扱規程を制定し管理者や利用者に遵守させる運用を徹底することが求められる。

情報の漏えいは、プログラムが機密情報にアクセスしさまざまな経路で外部へ伝達することによって起こる。つまり、プログラムを実行するプロセスが情報を伝達する経路には、ソケット、メッセージ、共有メモリ等のプロセス間通信やファイル等があり、プロセスがこれらの資源にアクセスし他のプロセスや資源に機密情報を伝達すると、機密情報は次第に拡散していく。したがって、情報漏えいの事実を認識しその経路を特定したり、漏えいを防止するには、プロセスが機密情報や資源にアクセスする状況を常に管理していくことが必要である。ここで、情報や資源へのアクセスや伝達には、計算機を制御するオペレーティングシステム（OS）が必ず関与している。このため、OS がプロセスの情報や資源へのアクセス状況を監視し、その履歴を取得したりアクセスを制御することで、機密情報の管理が可能である。たとえば、Security Enhanced Linux(SE-Linux)[1]に代表されるセキュア OS は、計算機上の資源に対するアクセス制御の粒度を細かく設定することにより、不正アクセス等における資源の不正な利用を防止する。また、プロセスによる機密情報へのアクセスをシステムコールライブラリやシステムコール処理の入口で監視し、あらかじめ定めた制御ポリシーに基づいて制御する方式も検討されている[2][3]。

ここでは、OS が提供するファイルのオープン処理に着目し、プロセスが機密情報のファイルをオープンすることを契機として計算機内に機密情報が拡散する経路を追跡する手法を提案する。具体的には、機密情報が拡散する基本的な経路を明らかにするとともに、拡散状況を追跡するアルゴリズムを提案する。また、提案手法を計算機上に実装し、基本的な処理オーバーヘッドの評価結果を報告する。

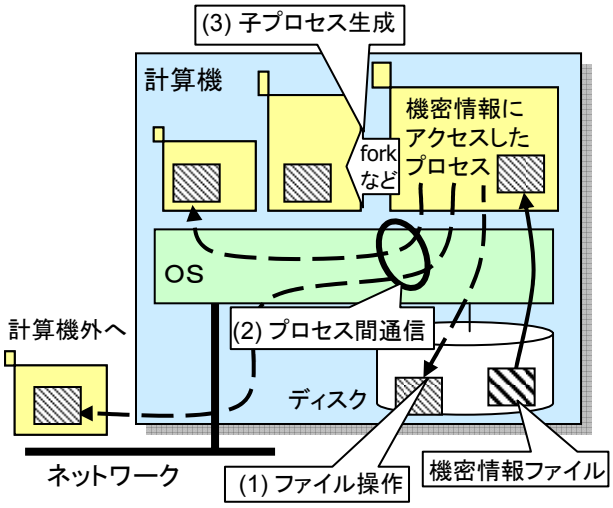

図1． 情報の拡散経路

2. 基本的な情報拡散の経路

以降では、機密情報はファイルの形式で存在し、どれが機密情報に相当するかは指定されていると仮定する。ここで、情報漏えいとは、取り扱いを許可されていない者（部外者）に機密情報が伝達されることである。機密情報の伝達は、ファイル形式で存在する機密情報をプロセスがオープンしてその内容を読み込み、さらに他のプロセスやファイルなどへその内容を伝えることで行われる。このとき、機密情報のファイルを複数のプロセスがオープンしたり、その内容を複数のプロセスやファイルなどへ伝えたりすると、機密情報は計算機やシステム内に次第に拡散していく。プロセスが情報を伝達する経路を、図１に示す。具体的には、以下の経路がある。

（１）ファイル操作

機密情報にアクセスしたプロセスが、さらに他のファイルにアクセスすることにより情報が伝達される。たとえば、あるプロセスＰが機密情報ファイルＦをオープンすると、そのプロセスＰはファイルＦ内の機密情報を読み出して利用することが可能となる。ここで、このプロセスＰが他のファイルをオープンすると、読み出した機密

情報をそのファイルに書き出される可能性がある。したがって、このファイルも機密情報ファイルとみなす必要がある。また、このファイルをさらに他のプロセスがオープンした場合は、そのプロセスも機密情報にアクセスする可能性があるため、そのプロセスがオープンする他のファイルも機密情報ファイルとみなさなければならない。

（2）プロセス間通信

機密情報にアクセスしたプロセスが、プロセス間通信により他のプロセスにその内容を伝達する場合がある。一般的なプロセス間通信の手段には、共有メモリ、メッセージ、ソケットがある。通信により、通信先のプロセスも機密情報を持っていると見なし、情報の拡散状況を把握するために管理していく必要がある。さらに、これらのプロセスがオープンするファイルも機密情報が伝達される先として管理することが求められる。

（3）子プロセスの生成

子プロセスの生成時に、子プロセスが親プロセスの資源を引き継ぐ機能がある場合（UNIX における fork 処理）には、この機能を通してプロセス間で情報が伝達される。したがって、親プロセスが機密情報ファイルにアクセスしているときは、子プロセスにも機密情報が伝達されたと判断する必要がある。

このように、機密情報ファイルをオープンしたことを契機として、機密情報を一部でも保有する可能性のあるファイルやプロセスを追跡することにより、機密情報が広範囲に拡散する可能性を把握できる。つまり、情報漏えいを防止するには、機密情報が伝達される経路を明らかにし、情報が拡散する状況を把握し制御することが必要である。

3. 追跡法への要求

機密情報の拡散を追跡する方法には、以下の要求がある。

（要求１）追跡に漏れがないこと

（要求２）リアルタイムな追跡が可能であること

（要求３）追跡処理のオーバーヘッドが小さいこと

（要求４）追跡の結果が的確であること

（要求５）追跡処理が利用しやすいこと

確実な追跡（要求１）、および漏えい防止のための警告などの制御をリアルタイムに行うための追跡法のリアルタイム性（要求２）は、機密情報の漏えいを確実に防止するために必須な事項である。一方、（要求３）（要求４）（要求５）は必須ではないが要望される事項であり、これらはトレードオフの関係にある。提供しているサービスへの影響を抑制するには、追跡処理オーバーヘッドの最小化（要求３）が必要である。また、（要求１）を満足するために追跡の結果が膨大なものになってしまっては、「疑わしきは漏えい」のような判断となり、システムの誤動作とも判断しかねない。したがって、追跡の結果が漏えいを高い確率で示す（要求４）ことが必要である。さらに、（要求１）や（要求４）を満足するために漏えいか否かを判断する処理を複雑化させると、追跡処理の設定処理が複雑化してしまう。例えば、SE-Linux は、詳細なセキュリティ確保が可能であるが、そのポリシーやパラメータ設定は非常に煩雑であり、精通することが要求される。したがって、追跡処理の設定が容易であり使いやすいこと（要求５）も必要である。

4. 情報拡散追跡法

4.1. 情報拡散のモデル

機密情報の漏えいを確実に把握するために、機密情報を持つ可能性のあるプロセスやファイルをリアルタイムに把握する仕組みについて検討する。ここでは、情報の伝達経路としてファイ

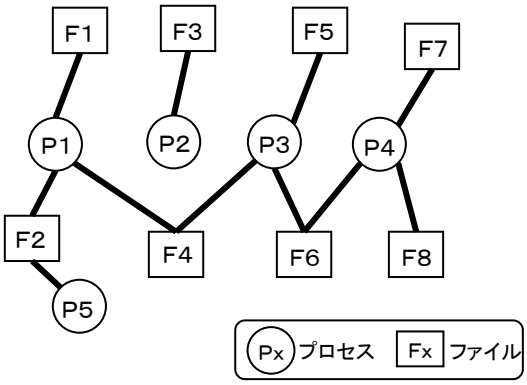

図2. プロセスとファイルの状態例
（機密情報にアクセスしていないとき）

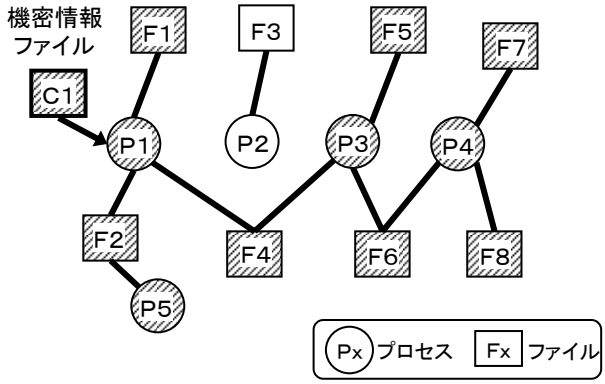

図3. プロセスとファイルの状態例
（機密情報にアクセスしたとき）

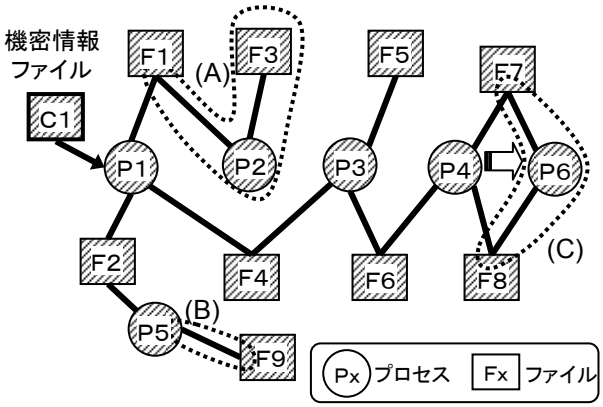

図4. プロセスとファイルの状態例
（機密情報にアクセスしたとき）

ル操作および子プロセスの生成を対象とし、機密情報が計算機内に拡散する様子をモデル化する。モデル化において、各プロセスの内部処理は自由なものと仮定する。すなわち、プロセスがファイルや他のプロセスへのアクセス経路を確立した時点で情報が伝達されたと考える。

情報の伝達経路をファイルとするとき、情報の拡散状況を追跡するには、計算機上の各プロセスと、それらのプロセスがオープンしているファイルの関係がわかればよい。これらの関係を元にして、機密情報が伝達された、あるいは伝達された可能性のあるプロセスやファイルを探索し管理対象に加えていくことにより、機密情報の拡散状況を把握することができる。

たとえば、ある時点における計算機上のプロセスの状態を図２に示す。図２において、P1, P2,... はプロセス、F1, F2, ...はファイルを示しており、プロセス P1 はファイル F1, F2, F4 をオープンしている。ここではどのプロセスもまだ機密情報ファイルをオープンしておらず、したがって図中のどのプロセスやファイルも情報拡散を把握するための管理対象となっていない。

ここで、プロセス P1 が機密情報ファイル C1 にアクセスしたときの情報拡散の様子を図３に示す。図３において、斜線で記されたプロセスやファイルは機密情報が伝達される可能性があるため管理対象となるものであり、①、②、...の数字はその探索順番を示す。まず P1 が機密情報ファイル C1 にアクセスすると、P1 は機密情報を保持する可能性があるため、管理対象となる（①）。次に、P1 がすでにオープンしているファイル F1, F2, F4 は P1 から機密情報を書き込まれる可能性があるため、これらも管理対象となる（②）。さらに、F2, F4 をオープンしているプロセス P3, P5 は F2, F4 に書き込まれた機密情報を読み出す可能性があるため、管理対象とする（③）。以下、同様に機密情報が伝達される可能性のあるファイルとプロセスが次々と管理対象となり、図３に斜線で示すファイルとプロセスが機密情報の拡散状況を把握するための管理の対象となる。

図４は、図３の状態からさらに３つの場合を示したものである。(A)は、管理対象となってい

ないプロセス P2 が、すでに管理対象となっているファイル F1 をオープンした場合である。このとき、P2 は機密情報を F1 から読み出す可能性があるため管理対象とするとともに、P2 がすでにオープンしているファイル F3 も管理対象とする。(B)は、管理対象となっているプロセス P5 が管理対象でないファイル F9 をオープンした場合であり、このとき F9 は新たに管理対象となる。(C)はプロセス P4 が fork により子プロセス P6 を生成した場合である。このとき、P6 は P4 の状態を引き継いで生成されるため、P4 と同様に管理対象になる。

このように、プロセスとそれらがオープンするファイルとの関係を把握しておくことにより、ファイルアクセスを情報伝達の経路としたときの機密情報の拡散状況を把握することができる。

4.2. 拡散状況を把握するための情報追跡処理

4.1 節で機密情報の拡散状況について考察した。これらを具体的な機密情報の追跡処理手順として明らかにする。まず、追跡処理の開始点は、ファイルをオープンしたときと子プロセスを生成した時点である。それぞれの時点における処理手順を以下に示す。

（1）ファイルオープン時の追跡処理アルゴリズム

オープンするファイルとプロセスの状態による処理手順を表 1 に示し、以下に解説する。

（ケース 1）ファイルとプロセスがともに管理対象である

プロセスとファイルがすでに機密情報を持つ可能性がある場合であり、双方ともすでに管理対象となっていることからあらたに管理対象を加える処理は行わない。

（ケース 2）プロセスがすでに管理対象であり、ファイルが管理対象でない

表1. ファイルのオープン要求時の追跡処理

		追跡対象ファイルであるか	
		である	でない
追跡対象プロセスであるか	である	（ケース1） ・新たに管理対象とするものはない	（ケース2） ・ファイルを新たに管理対象にする ・ファイルをオープンしているプロセス全てを管理対象にする
	でない	（ケース3） ・プロセスを新たに管理対象にする ・プロセスがオープンしているファイルを全て管理対象にする	（ケース4） ・新たに管理対象とするものはない

プロセスが持つ機密情報ファイルを書き込まれる可能性があるため、ファイルを新たに管理対象とする。また、このファイルをすでに他のプロセスがオープンしているときは、そのプロセス全てを新たに管理対象とする（ケース 3 の処理）。

（ケース 3）プロセスが管理対象でなく、ファイルがすでに管理対象である

ファイルが持つ機密情報をプロセスが読み込む可能性があるため、プロセスを新たに管理対象とする。また、このプロセスがすでにオープンしているファイル全てを新たに管理対象とする（ケース 2 の処理）。

（ケース 4）ファイルとプロセスがともに管理対象でない

プロセスとファイルがともに機密情報を持っていないとみなし、新たに管理対象を加えることはしない。

このように、ファイルオープン時の処理はファイルとプロセスのいずれかが管理対象となることにより、そのファイルをオープンしている他のプロセスや、そのプロセスがオープンしている他のファイルも全て管理対象となる。

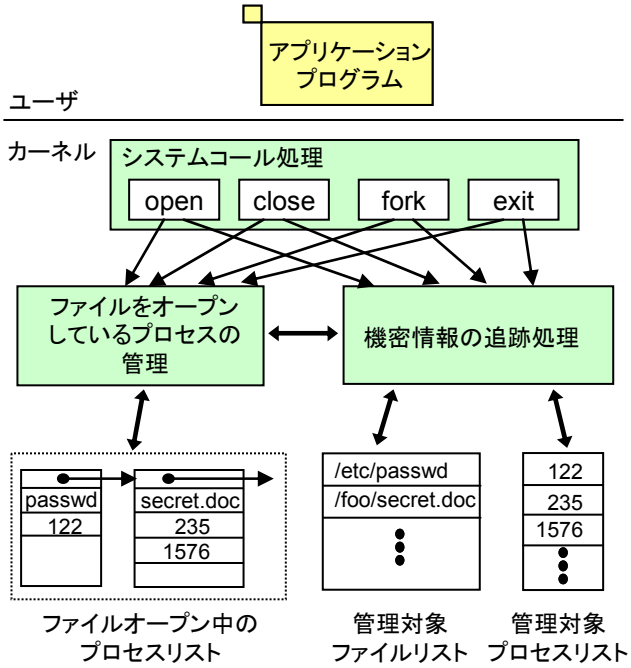

図5.　モジュール構成と管理テーブル

（2）プロセス生成時の追跡処理アルゴリズム

新たに子プロセスを生成するときは、親プロセスが管理対象となっているかどうかにより処理が分かれる。

親プロセスがすでに管理対象となっている場合は、子プロセスにも機密情報が継承されるため管理対象とする必要がある。このとき、親プロセスがオープンしているファイルは全てがすでに管理対象となっている。

一方、親プロセスが管理対象となっていない場合は、子プロセスも管理対象とはならない。

5. 実装と評価

5.1. 実装

提案した方式を、Linux2.6.0 上に実装した。実装したモジュール構成とテーブル構造を図５に示す。それぞれのモジュールにおける処理概要は以下の通りである。

（1）システムコール処理から実装機能の呼び出し

表2.　システムコール処理時間の測定結果

	open	close
追跡処理あり	9,600	2,900
追跡処理なし	9.000	2,800

(nsec)

表3.　fork処理実行時のオーバーヘッド

	機密情報ファイルをオープンしていた場合	機密情報でないファイルをオープンしていた場合
追跡処理の実行時間	570	460

(nsec)

open, close, fork, exit の各システムコールの入り口において、追跡機能を呼び出す。

（2）各ファイルをオープンしているプロセスの管理

4.2 節の処理手順を実現するには、あるファイルをオープンしているプロセス全てを探索する必要がある。Linux では、各プロセスがどのファイルをオープンしているかの情報はタスク構造体から得ることができるが、各ファイルがどのプロセスからオープンされているかを知ることは難しい。このため、ファイル単位にテーブルを新たに用意し、ファイルをオープンするたびにそのプロセスを登録する処理を実現する。この処理は機密情報ファイルだけでなく、全てのファイルを対象として実行される。これにより、あるファイルが新たに管理対象となったときに、そのファイルをオープンしているプロセスを管理対象とする処理を高速に行うことができる。

（3）機密情報の追跡処理

３．２節で述べたアルゴリズムを実行する。管理対象のファイルとプロセスを識別するために、それぞれの情報を格納する管理テーブルを持

つ。

5.2. 基本機能のオーバーヘッド評価

実装した OS を Intel Pentium 4 2.40GHz プロセッサを搭載する計算機で動作させ、本機能を実現したことによる基本的なオーバーヘッドを測定した。なお、以降の評価でファイルアクセスは/tmp 下にあるファイルを対象とし、測定前に open/close を実行してＯＳのバッファキャッシュにヒットする状態とした。また、測定はシングルユーザモードで行った。

（１）ファイルオープン処理時のオーバーヘッド

まず、機密情報でないファイルを open/close したときにかかる処理時間を測定した。時間の測定には gettimeofday()システムコールを用いた。結果を表２に示す。まず、open 処理では追跡処理を実装したことにより数百 nsec のオーバーヘッドが見られる。これは、ファイルをオープンしているプロセスをファイルごとに管理する処理を行っているためである。ここで、open 処理における処理の増加分をハードウェアクロックにより詳しく測定すると、570nsec であった。追跡処理を実装することによって通常のファイルオープン処理にかかるオーバーヘッドは、バッファキャッシュにヒットする状態でも全体の数％であり、大きな負荷にはならないといえる。一方、close 処理では追跡処理のあるなしにかかわらず処理時間はほとんど同じであり、open 処理と同様に負荷にならないといえる。

次に、複数のファイルをオープンしているプロセスが新たに機密情報ファイルをオープンしたときに、追跡処理の実行にかかる処理時間を評価した。結果を図６に示す。ファイル数が０，すなわち事前にファイルをオープンしていないときに機密情報ファイルをオープンした場合に、追跡処理にかかる時間は 1600nsec 程度であり、open 処理全体の１５％ほどである。そして、オープンしているファイル数に対してほぼ正比例で増加している。

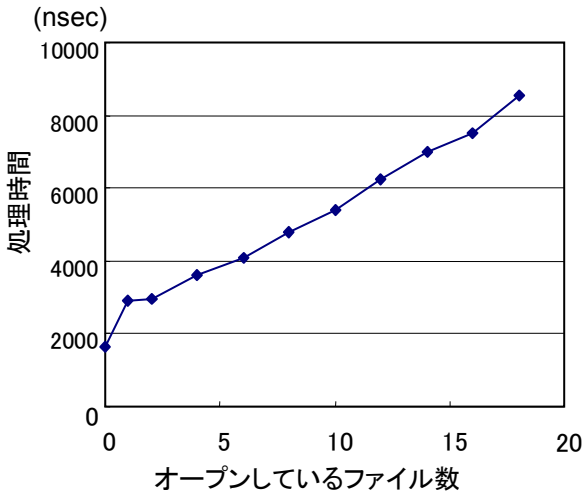

図６．　機密情報ファイルオープン時のオーバーヘッド

（２）fork 処理時のオーバーヘッド

機密情報ファイルをオープンしているプロセスが fork を実行した場合と、機密情報でないファイルをオープンしているプロセスが fork を実行した場合とで、追跡処理にかかる処理時間を評価した。機密情報ファイルでないファイルをオープンしていた場合でも、fork によりそのファイルをオープンしているプロセスが増えることになり、ファイルごとにオープンしているプロセスを管理するための処理が必要である。機密情報ファイルをオープンしている場合は、それに加えて新たに生成されたプロセスを管理対象とする処理が実行される。いずれの場合でも増加する処理時間は 400～500nsec であり、fork 処理全体に比べると大きくない。

６．おわりに

機密性を要する情報の漏えいを防止することを目的として、計算機内に機密情報が拡散する経路を追跡する手法を提案した。機密情報がファイル形式で存在するとき、その内容が拡散する経路

には、ファイル操作、プロセス間通信、および子プロセスの生成がある。機密情報が拡散する経路を追跡する際の要求を示した。ファイル操作と子プロセス生成とを対象にしたときに機密情報が計算機内に拡散する様子をモデル化し、具体的な追跡処理の手順を明らかにした。Linux 上に追跡処理を実装し、処理にかかる時間を測定し、従来のシステムコール処理に対して大きな処理時間の増加にならないことを示した。

今後の課題は、まず拡散経路としてプロセス間通信を考慮することが考えられる。プロセス間通信にはソケット、メッセージ、共有メモリなどがあり、これらを対象としたときの情報拡散のモデル化が必要である。また、情報が拡散するときの条件を精査することにより、真に情報が伝達された場合のみを追跡の対象とする検討も必要である。

参考文献

[1] Loscocco and Smalley, Integrating Flexible Support for Security Policies into the Linux Operating System, Proc. of the FREENIX Track: 2001 USENIX Annual Technical Conference (FREENIX '01), June 2001.

[2] 一柳, 鈴来, 毛利, 大久保："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", 情処研報 2005-OS-98, Vol.2004, No.16, pp.1-8(2005).

[3] 鷲尾, 除補, 大嶋, 金井："属性の伝搬を利用した電子文書の柔軟な利用制御方式の提案", 情処研報 2005-CSEC-28, Vol.2004, No.65, pp.375-380(2005).