ユーザレベルでの情報漏洩防止機構 *User-Mode Salvia* の構築

河島 裕亮† 岩永 真幸†† 毛利 公一†
† 立命館大学 情報理工学部 †† 立命館大学大学院 理工学研究科

1 はじめに

近年, 計算機やネットワークの発展に伴い, プライバシ情報を電子データとする情報の電子化が進んでいる. それに伴い, 電子化されたプライバシ情報が, データ保有者の意図に反して漏洩する事件が多発している. 文献 [1] では, 情報漏洩を引き起こす要因として, 情報の正規利用者の管理ミスによる流出や紛失, 外部への持出し, アプリケーションの誤操作, 盗難が挙げられている. 特に, 情報の正規利用者による情報漏洩が, 発生原因の多くを占めることを示している. しかし, 暗号化や認証といった既存のセキュリティ技術は, 外部からの攻撃を防止することを目的とするため, 人為的なミスや内部犯による情報漏洩を防止することは困難である.

以上の背景により, 我々は, 人為的なミスや内部犯による情報漏洩を防止する手法として, Privacy-aware OS *Salvia*[2](以下, Salvia と記す) の開発を行っている. 上述の情報漏洩は, プロセスが発行するシステムコールを契機として発生する. そのため, *Salvia* では, システムコールを通じて重要なデータが漏洩しているか否かを検査する仕組を実現した. ただし, *Salvia* は, OS レベルで実装されているため, 保護機構の移植作業や機能拡張を容易にすることを目的として, *Salvia* の保護機構をユーザレベルで実現する *User-Mode Salvia*(以下, UMS と記す) を構築している.

UMS は, 現在 Linux の ptrace システムコールを用いて *Salvia* の保護機構をユーザレベルで実現している. すなわち, UMS の動作に最低限必要な ptrace 相当のインタフェースが存在すれば, 僅かな修正で種々の OS へも適応可能である. また, OS において, バージョンアップなどの仕様変更がある場合でも, そのインタフェースが変更されることは少ないため, UMS の保護機構を大きく修正する必要がない, といった利点がある.

以下, ユーザレベルで情報漏洩を防止する *User-Mode Salvia* の概要, 構成, 評価について述べる.

Construction of information leak prevention mechanism *User-Mode Salvia*
Yusuke Kawashima†, Masayuki Iwanaga††, and Koichi Mouri†
†College of Information Science and Engineering, Ritsumeikan University
††Graduate School of Science and Engineering, Ritsumeikan University

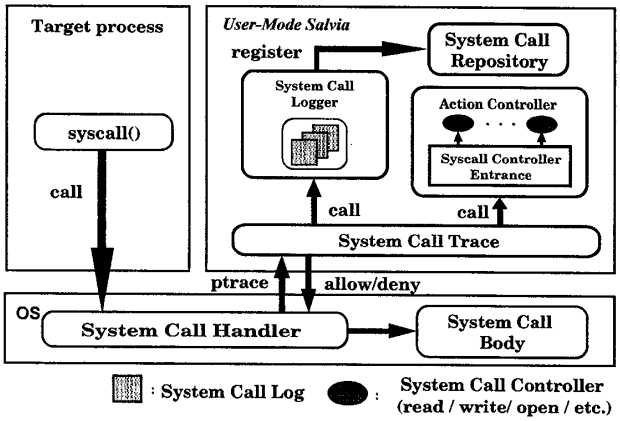

図 1 *User-Mode Salvia* の構成

2 ユーザレベルでの情報漏洩防止機構

2.1 概要

計算機上で管理される情報が漏洩する際, プロセスで発行されたシステムコールを必ず経由することになる. そのため, *Salvia* は, 情報漏洩の原因となるシステムコールを制御することで, 情報漏洩を防止する. *Salvia* では, ファイルごとにデータ保護ポリシを設定可能とし, 漏洩の要因となるシステムコールが発行されると, 保護ポリシやコンテキストの内容に基き, そのシステムコールの実行可否を判定する. UMS は, このようなプロセスの監視, 保護ポリシ・コンテキストの管理, システムコールの実行制御といった機能をユーザレベルで実現する.

2.2 全体構成

UMS は, Linux で提供される ptrace システムコールを基に Linux 上に実装している. UMS の情報漏洩防止機構 (図 1 参照) は, プロセスが発行するシステムコールの検知を行う System Call Trace, プロセスが発行するシステムコールを履歴として収集する System Call Logger, その履歴を時系列データとして管理する System Call Repository, データ保護ポリシ, コンテキストの取得, システムコールの実行制御を行う Action Controller から成る.

System Call Trace は, UMS によるプロセスの監視を行うために, ptrace を用いて監視対象プロセスと UMS 間の接続処理を行う. また, 監視対象のプロセスからシステムコールが発行されると, システムコールの入出口において, システムコール番号と返り値を取得する. ここで取得したシステムコール情報は, System Call Logger

や Action Controller に渡す.

System Call Logger は, プロセスが発行したシステムコールを履歴として収集する. システムコール履歴の収集には, システムコールごとに異なる引数や返り値の数やデータ構造に対応させるため, それぞれに履歴取得用関数を設ける. 取得した履歴は, System Call Repository において時系列データとして管理を行い, コンテキストの一種として利用する.

Action Controller は, 保護ポリシの取得や, 取得した保護ポリシの内容に応じて, 情報漏洩の要因となるシステムコールの制御を行う. 保護ポリシは, ファイルの拡張属性として管理する. そのため, open システムコールの発行時に, オープンするファイルにポリシが存在するか否かを確認することで, ポリシの有無を判定する. ポリシが存在する場合, その内容を登録する. その登録されたポリシの内容に応じてシステムコールの実行可否を制御する. システムコールを制御するためには, システムコールごとの性質や引数・返り値のデータ構造に対応させるため, それぞれに System Call Controller を用いる.

3 性能評価

3.1 性能評価環境

UMS を Linux(2.6.16-0vl60) を上に実装し, Intel Pentium III(933MHz), メモリ 256MB を搭載した PC で性能を評価した.

3.2 性能評価

ユーザレベルでの情報漏洩防止機構の性能として, 監視対象プロセスから発行された open システムコールの処理時間を測定した. 評価項目を以下に示す.

(1) 既存の open システムコール

(2) UMS 監視下の open システムコール

また, 処理時間の内訳を求めるために, open システムコールの制御用関数における処理時間を測定した. 具体的には, 以下の処理を行う関数の処理時間を計測する.

(a) ファイル名取得処理

(b) 保護ポリシの検索と内容の登録処理

(c) システムコールの発行と終了の検知

処理時間の計測には, RDTSC(read-time stamp counter) 命令を用いて計測したクロック数を基に算出した. 測定結果を表 1, 表 2 に示す.

表 1 open システムコール全体の処理時間 (μsec)

評価項目	最大値	最小値	最頻値
(1)	20.5	8.1	8.7
(2)	158.7	50.8	52.3

表 2 open システムコール制御用関数の処理時間 (μsec)

制御用関数	最大値	最小値	最頻値
(a)	93.8	15.8	16.9
(b)	41.6	7.3	11.3
(c)	33.6	1.3	6.7

4 考察

評価の結果, UMS の監視下に置かれたプロセスが発行する open システムコールに発生する遅延時間は, 約 52.3μsec となった. これは, 通常の open システムコールの処理時間に対して, 約 6 倍の遅延が発生していることを意味する. ここで, open システムコールの遅延時間の内訳は, (a) が 16.9μsec(32.3%), (b) が 11.3μsec(21.6%), (c) が 6.7μsec(12.8%) となった. しかし, (c) はシステムコールの発行直後と終了直前の 2 回呼出すため, 実際の処理時間は約 15 ～ 20μsec となり, 全体の約 29% を占める.

open システムコールの遅延時間の内, (c) の処理が最も影響することがわかった. しかし, 現在の実装では, システムコールが発行されたことを検知するために, ptrace システムコールと waitpid システムコールを用いる必要がある. すなわち, (c) の処理時間は, 不可欠な時間である. また, ファイルの拡張属性から保護ポリシを取得するためには, (a), (b) の処理を行う必要があるため, 上記の処理時間を必要とする. したがって, UMS を用いて open システムコールを制御する場合, (a), (b), (c) の処理時間を要する. また, UMS では,read や write など, 情報漏洩の要因となるシステムコールに対する制御も行うため, その制御における処理時間を計測する必要がある.

5 おわりに

本稿では, ユーザレベルでの情報漏洩機構 *User-Mode Salvia* について, その概要と構成, 性能評価について述べた. 今後の課題は, プロセス間通信における制御方法の考案と実装, システム全体の高速化を行う.

参考文献

[1] NPO 日本ネットワークセキュリティ協会: JNSA 2007 年情報セキュリティインシデントに関する調査報告書, http://www.jnsa.org/result/2007/pol/incident/2007incidentsurvey_v1.32.pdf, 2008.

[2] 鈴木和久, 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 (ACS 13), Vol. 47, No. SIG3, pp. 1–15, 2006.