Vol.2016-CSEC-75 No.1
2016/12/1

動的テイント解析機能を利用したOSによる細粒度データ出力制御手法

松本 隆志[1,a)]　明田 修平[1]　瀧本 栄二[1]　齋藤 彰一[2]　毛利 公一[1]

概要：情報漏洩インシデントの主な原因は，人為的なミスによるものであると報告されている．この問題に対して，我々は人為的なミスによる情報漏洩を防止するセキュアシステム TA-Salvia の開発を行ってきた．TA-Salvia は，動的テイント解析機能を有するハードウェアエミュレータと OS が連携することで，ユーザプロセスが扱うデータフローを追跡し，漏洩を検出・防止することが可能である．これまでは，データの使用許可範囲を定義する保護ポリシをファイル毎に設定してアクセス制御を行ってきた．本論文では，TA-Salvia におけるテイント解析の範囲をディスクまで拡張することで，ディスク上のデータも 1 バイト単位で識別・追跡可能とした．これにより，さらに粒度の細かいバイト毎のアクセル制御が可能となったので，それについて報告する．また，インターネット上で公開されている実アプリケーションを用いた TA-Salvia の活用例についても述べ，TA-Salvia が実環境において利用可能であることを確認した．

キーワード：情報漏洩防止，ファイルアクセス制御，動的テイント解析，オペレーティングシステム

Fine-Grained Access Control Method by OS Using Dynamic Taint Analysis

Takashi Matsumoto[1,a)]　Shuhei Aketa[1]　Eiji Takimoto[1]　Shoichi Saito[2]　Koichi Mouri[1]

Abstract:
Most data loss incidents are caused by human error. We have been developing data loss prevention system for such incidents, called TA-Salvia. TA-Salvia is the system which tracks a data flow in user processes by dynamic taint analysis in hardware emulator and prevents data loss by operating system. To date, TA-Salvia has set the policy which specifies the scope of permission to a file and has controlled access in one file units. In this paper, we report the method which tracks a data on the disk by extending the range of taint analysis to a disk and controls access in 1-byte units. We evaluated TA-Salvia by real applications published on the Internet and confirmed data loss prevention.

Keywords: Data Loss Prevention, File Access Control, Dynamic Taint Analysis, Operating System

1. はじめに

情報システムの発展によって個人情報が電子化されるようになり，電子化された個人情報の漏洩事件が増加している．JNSA 2015 年情報セキュリティインシデントに関する調査報告書 [1] によると情報漏洩の発生件数の約 70 %は，「紛失・置忘れ」，「誤操作」および「管理ミス」が原因となっている．「紛失・置忘れ」は，持出し許可を得た情報を持出し先や移動中に忘れたり，紛失したりすることである．「誤操作」は，宛先の書き間違いや操作ボタンの押し間違いなどのことである．「管理ミス」は，業務上において，作業手順の誤りや情報の公開・管理のルールが明確化されていなかったことが原因のミスである．このように，

[1] 立命館大学
Ritsumeikan University, Kusatsu, Shiga, 525-8577, Japan
[2] 名古屋工業大学
Nagoya Insitute of Technology, Nagoya, Aichi 466–8555, Japan
[a)] tmatsumoto@asl.cs.ritsumei.ac.jp

情報漏洩のほとんどは，コンピュータウイルスやハッキングによる不正アクセスではなく，正当なアクセス権限を持つユーザの人為的なミスによるものである．

情報漏洩を防止するための既存のセキュリティ技術として，ユーザ認証によるアクセス制御やファイルの暗号化などが存在する．しかし，これらの技術を利用した場合において，正当なアクセス権限を持ったユーザが認証・復号した後は，自由にデータが扱えるため，人為的ミスによる情報漏洩の防止が困難である．また，既存のセキュリティ技術を拡張したシステムとして，SELinux[2] や TOMOYO Linux[3] などの強制アクセス制御がある．強制アクセス制御は，プロセスのリソースに対する操作毎にアクセス権限を詳細に設定できるため，より強力なアクセス制御を行うことができる．事前にファイルの機密度が固定的に決まっている場合には，それに適したセキュリティポリシを記述することで，人為的ミスによる情報漏洩の防止が可能である．しかし，多数のファイルの機密度が変化する場合などについては，それらに適応したセキュリティポリシを記述することは難しい．また，従来のアクセス制御で使用されていた ACL やパーミッションなどのファイルに対して設定するポリシの他に，多種類のポリシを必要とするため，データの保護を目的とする設定の記述が複雑化しやすいといった問題がある．特に，一般ユーザが個々に決定するのは困難である．

以上の背景から，我々はこれまで，人為的なミスによる情報漏洩を防止するセキュアシステム TA-Salvia[4] の開発を行ってきた．TA-Salvia は，Argos[5] と呼ばれる動的テイント解析機能を有するハードウェアエミュレータと OS が連携することで情報漏洩を防止するシステムである．具体的には，ユーザプロセスが扱うデータフローを Argos の動的テイント解析機能を用いて追跡し，その追跡結果に基づいて OS がデータの出力などを判定することで，情報漏洩を防止する．これまでは，データ提供者の意図する方針（機密度に相当する）をデータ保護ポリシ（以下，ポリシ）として定義し，ファイル毎に設定してアクセス制御に利用していた．本論文では，TA-Salvia におけるテイント解析の範囲をディスクまで拡張することで，メモリだけでなくファイルにおいても，1 バイト単位でデータを識別・追跡可能とした．これにより，ファイル毎のアクセス制御より，さらに粒度の細かいバイト毎のアクセル制御が可能となったので，それについて報告する．

以下，本論文では，2 章で先行研究について述べ，3 章で提案手法について述べる．4 章で実装について述べ，5 章で提案手法の評価について述べる．6 章で関連研究について述べ，7 章でまとめる．

2. 先行研究

人為的なミスによる情報漏洩を防止する先行研究として，我々は過去に，Salvia Linux[6]，DF-Salvia[7]，User-mode DF-Salvia[8] を提案した．これらは，ポリシをファイル毎に設定でき，ポリシが設定されたファイルは，ポリシの記述に従って制御される．Salvia Linux は，ポリシが設定されたファイルを読み出したプロセスに対し，ポリシに基づいた強制アクセス制御を課すことで，プライバシを考慮したデータ保護を実現している．具体的には，プロセスの挙動を監視し，データ漏洩が発生する可能性のある計算機資源へアクセスしたとき，当該プロセスが過去に読み出した全保護ファイルのポリシに違反しないアクセスのみを許可する．Salvia Linux は，このようなデータ保護機構を OS 内に有しているため，すべてのアプリケーションに透過的に適用可能である．ただし，全保護ファイルのポリシに違反しないかを確認するため，過剰なアクセス制御が発生する場合がある．DF-Salvia と User-mode DF-Salvia は，データフローを単位として制御することで Salvia Linux の過剰なアクセス制御の問題を解決している．DF-Salvia は，コンパイラと協調し，プロセス内の個々のデータフローを主体として管理・制御することでアクセス制御を行う．User-mode DF-Salvia は，プログラムへアクセス制御用のコードを追加することで，プログラム単体でアクセス制御が可能となり，プラットフォームに依存しないアクセス制御を実現する．また，User-mode DF-Salvia は，プログラムのデータフローに基づいてデータを区別している．DF-Salvia と User-mode DF-Salvia は，いずれも事前にコンパイラを用いてプログラムのデータフローを解析またはコード変換をする必要がある．そのため，限られた数のアプリケーションにのみ適用させる場合には適しているが，システム全体のアプリケーションに共通に適用させる場合には向かない．

上述の 3 つの Salvia は，共通してファイル毎にポリシを設定してアクセス制御を行っている．しかし，ファイル毎のポリシ管理では，まだなお過剰なアクセス制御が発生する場合がある．例えば，Salvia.txt と Lavender.txt の 2 つのファイルを用意したとする．Salvia.txt は，「root と Salvia グループは，読出しを許可する（1）」というポリシを設定している．Lavender.txt は，「root と Lavender グループは，読出しを許可する（2）」というポリシを設定している．このとき，root がこれらのファイルを読み出して，Salvia_Lavender.txt を作成した．このファイルは，（1）と（2）のポリシを持つことになる．これまでの Salvia では，すべてのポリシに違反しないかを確認してしまうため，root 以外のユーザが Salvia_Lavender.txt を読み出すことができない．本来，Salvia.txt が源となるデータは，Salvia グループであれば読出しを許可されるべきである．しかし，（2）のポリシが適用されてしまうため，読み出すことができない．このように複数のファイルを 1 つのファイルに統合した場合，過剰なアクセス制御が発生してしま

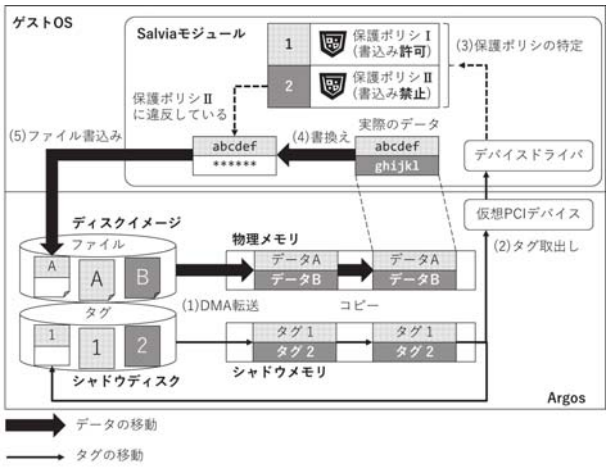

図 1 TA-Salvia の構成

う場合がある．本論文では，TA-Salvia を改良することで上記の問題を解決する．

3. 提案手法

本論文では，TA-Salvia を改良することで先行研究で発生する過剰なアクセス制御の問題を解決する．改良した TA-Salvia の全体構成を**図 1** に示す．本章では，TA-Salvia における Argos の動的テイント解析機能，シャドウ領域制御用のインタフェース，ポリシの管理方法，バイト単位のアクセス制御について述べ，最後に全体の動作の流れについて述べる．

3.1 Argos の動的テイント解析機能

動的テイント解析は，解析したいデータに対してテイントタグ（以下，タグ）と呼ばれる識別子を割り当てることで，データの伝播を追跡する技術である．動的テイント解析は，タグを割り当てたデータが別の領域に伝播されるとき，タグも同時に伝播させる．データに設定されたタグを確認することで，データフローの識別が可能である．

TA-Salvia が利用している Argos は，QEMU[9] と呼ばれるハードウェアエミュレータに動的テイント解析機能を追加したシステムである．動的テイント解析を実現するためには，データにタグを設定できるような仕組みが必要である．Argos は，データにタグを設定するために，物理メモリとレジスタに 1 対 1 に対応するタグ保持用のシャドウメモリとシャドウレジスタを用意している．Argos は，仮想 CPU が命令をフェッチするたびにタグを確認し，Argos のタグ伝播ルールに基づいてタグを伝播させることでデータの追跡を可能にしている．TA-Salvia では，この Argos をさらに拡張し，ディスク上のデータに対してもタグを保持できるようにした．具体的には，エミュレータで使用するディスクイメージと同じサイズのタグ保持用のシャドウディスクイメージ（以下，シャドウディスク）を用意し，シャドウディスクとシャドウメモリ間でタグを伝播できるように拡張した．これにより，メモリだけでなくディスク上のファイルのデータも追跡することが可能となる．

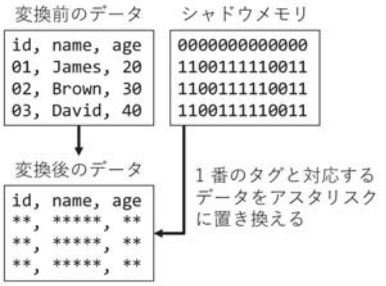

図 2 アクセス制御イメージ

3.2 シャドウ領域制御用のインタフェース

TA-Salvia は，Argos の動的テイント解析機能を OS が利用することでアクセス制御を行う．しかし，Argos には，OS にタグを設定・取得させるための機能を持たない．そのため，TA-Salvia は，OS からシャドウ領域を制御するためのインタフェースとして，Argos に仮想 PCI デバイスを追加した．また，OS には，そのデバイスを利用するためのデバイスドライバを用意した．OS は，デバイスドライバを通して Argos の仮想 PCI デバイスにコマンドを送信し，Argos は，仮想 PCI デバイスを通して OS のデバイスドライバにコマンドの実行結果を返す．このような仕組みを用いて OS は，シャドウメモリにタグ付け・取出しなどを行うことができる．

3.3 ポリシの管理方法

TA-Salvia は，ポリシとタグを紐付けて管理するために，ポリシ管理テーブルを用意している．あるファイルのデータにポリシを設定したい場合，まず，ポリシファイルを作成する．ポリシファイルの保存先は，/etc/salvia ディレクトリ以下である．作成したポリシファイルをポリシ管理テーブルに登録し，ポリシと対応するタグを保護したいファイルのデータに設定する．タグが設定されたデータは，ポリシに基づいたアクセス制御を受けることができる．

3.4 バイト単位のアクセス制御

TA-Salvia は，情報漏洩の要因となるシステムコールをフックし，アクセス制御機能を追加している．バイト単位のアクセス制御は，制御したいデータを伏せ字（アスタリスク）に置き換えることで実現する．**図 2** にアクセス制御イメージを示す．1 番のタグに関連付けられたポリシに「ファイルへの書込みを禁止する」と記述されていた場合，TA-Salvia は，1 番のタグと対応するデータをすべてアスタリスクに変換することで，出力を禁止する．

3.5 動作の流れ

ポリシファイルの作成やファイルへのタグ付けは，実際のファイルアクセスよりも前に設定しておく．ファイルアクセス時の TA-Salvia の動作の流れは，以下のようになる．

(1) ディスク上のファイルは，OS の先読み処理により，ディスクキャッシュとしてメモリに転送される．この転送と同時に Argos は，シャドウディスク上の対応するタグをシャドウメモリに伝播させる．
(2) ファイル書込み時に OS は，書き込むデータと対応するタグを Argos のシャドウメモリから取得する．タグ取出しは，仮想 PCI デバイスとそれを制御するデバイスドライバを通して行われる．
(3) 取得したタグとポリシ管理テーブルを用いて，適用すべきポリシを特定する．
(4) ポリシに違反したデータがあれば，そのデータをアスタリスクに置き換える．アスタリスクに置き換えたデータのタグは，0（タグなし）に変更される．
(5) ファイル書込みにより，ファイルのデータがメモリからディスクに転送される．この転送と同時に Argos は，シャドウメモリ上の対応するタグをからシャドウディスクに伝播させる．

4. 実装

本章では，提案手法の実装について述べる．特に，TA-Salvia を改良した点であるシャドウディスク，ポリシとタグの紐付け，タグの設定と削除，バイト単位のアクセス制御について述べる．

4.1 シャドウディスク

TA-Salvia には，エミュレータで使用するディスクイメージと同じサイズのシャドウディスクを用意している．これにより，ディスク上のファイルのデータに対してもタグを保持させることができる．また，シャドウディスクは，ディスクイメージとして用意するため，シャットダウン後もタグを保持させることが可能である．

TA-Salvia は，ディスク上のファイルデータがメモリに書き込まれるとき，タグも同時にシャドウディスクからシャドウメモリに伝播させる．そのために，Argos の IDE コントローラの DMA 転送処理にシャドウディスクとシャドウメモリ間のタグ伝播処理を追加した．具体的には，データの転送要求が来たときに，セクタ番号とデータサイズから，伝播させるべきシャドウディスクの領域を特定し，メモリにマップする．ディスクからメモリへの転送要求の場合は，シャドウディスクからシャドウメモリにタグを伝播させる．メモリからディスクへの転送要求の場合は，シャドウメモリからシャドウディスクにタグを伝播させる．これにより，シャドウディスクとシャドウメモリ間のタグ伝播を実現している．

4.2 ポリシとタグの紐付け

TA-Salvia は，ポリシ管理テーブルを OS 側に用意している．このテーブルは，タグ番号とポリシの 2 つの要素を持ち，タグとポリシを紐付けるために使用される．TA-Salvia は，ポリシを/etc/salvia ディレクトリ以下に，policy.001 から policy.255 というファイル名で管理している．ファイル名の最後の番号は，タグ番号と対応している．policy.002 という名前のポリシファイルがあった場合，TA-Salvia は，2 番のタグと紐付けて管理する．

TA-Salvia は，ルートファイルシステムマウント時もしくは salvia_load_policy システムコール発行時にポリシをポリシ管理テーブルに登録する．salvia_load_policy システムコールは，TA-Salvia に新たに追加したシステムコールである．ルートファイルシステムマウント後にポリシを追加・修正したい場合は，このシステムコールを使用する．このシステムコールは，ファイルディスクリプタで参照されるポリシファイルを読み出し，ポリシ管理テーブルに登録する．

4.3 タグの設定と削除

TA-Salvia は，ファイルにタグを設定・削除するためのシステムコールを用意している．本節では，これら 2 つのシステムコールについて述べる．

4.3.1 salvia_set_tag システムコール

ユーザは，salvia_set_tag システムコールを用いてファイル内のデータにタグを設定する．このシステムコールの引数は，以下の 4 つである．

- int fd
- off_t offset
- size_t length
- unsigned char tag

このシステムコールは，ファイルディスクリプタ（fd）で参照されるファイルのオフセット（offset）から length バイトにタグ（tag）を設定する．このシステムコールの動作手順を以下に示す．

(1) mmap システムコールを用いてファイルをメモリにマップする．
(2) 強制的にページフォルトを起こし，実際にメモリにマップさせる．
(3) マップされた領域にタグ付けを行う．
(4) タグ付けしたデータを更新する．
(5) msync システムコールを用いて，ディスク上のファイルと同期を取る．

まず，システムコールが発行されると，TA-Salvia は，mmap システムコールを用いて指定されたファイルをメモリにマップする．ただし，mmap システムコールは，メモリマップの宣言を行うだけである．そのため，実際にデータをメモリにマップするには，ページフォルトを発生させ

る必要がある．ページフォルトは，mmap システムコールによって宣言された領域にアクセスすることで発生する．TA-Salvia は，強制的にページフォルトを発生させるために，先頭のデータを別の変数にコピーする．これにより，TA-Salvia は，1 ページ（4096 バイト）分のデータをメモリにマップさせることができる．次に，TA-Salvia は，ファイルデータがマップされた領域にタグ付けを行う．最後に，TA-Salvia は，タグ付けしたデータを同じデータで上書きすることで，ページをダーティ状態に移行させる．OS は，ダーティ状態のページがあった場合，ディスク上のデータと同期を取る．OS がディスク上のデータと同期を取ると，タグも同時にシャドウディスクに伝播される．これにより，ファイル内のデータにタグを設定することができる．

4.3.2 salvia_delete_tag システムコール

ユーザは，salvia_delete_tag システムコールを用いてシャドウメモリ・シャドウディスクに伝播されたすべてのタグを削除することができる．このシステムコールは，引数に unsigned char 型の tag を指定する．このシステムコールは，シャドウメモリ・シャドウディスク上の tag 番のタグを削除する．このシステムコールのために，Argos に追加したシャドウ領域制御用のインタフェースである仮想 PCI デバイスを改良して，タグ削除用のコマンドを追加している．このシステムコールの動作手順を以下に示す．

(1) OS から仮想 PCI デバイスにタグ削除コマンドを送信する．
(2) 仮想 PCI デバイスは，指定されたタグと一致するシャドウメモリ上のタグを削除する．
(3) 仮想 PCI デバイスは，指定されたタグと一致するシャドウディスク上のタグを削除する．

このシステムコールは，すべてのタグを削除したい場合に使用する．ファイル内のデータに設定されたタグだけを削除したい場合は，salvia_set_tag システムコールの第 4 引数に 0（タグなし）を指定することで，対応するシャドウディスク上のタグを削除することができる．

4.4 バイト単位のアクセス制御

TA-Salvia は，情報漏洩の要因となるシステムコールをフックし，アクセス制御機能を追加している．現在実装が完了しているシステムコールは，write，writev，send，sendto，sendfile，sendfile64，mmap の 7 つである．他にも実装が必要なシステムコールとして，pwrite や sendmsg などがあるが，まだ実装できていない．本論文では，実装した 7 つのシステムコールのアクセス制御について述べる．

4.4.1 write，writev，send，sendto システムコール

まずは，write システムコールのアクセス制御の実装について述べる．write システムコールの出力を制御するためには，第 2 引数のバッファを書き換える必要がある．しかし，write システムコールの第 2 引数のバッファには，const 修飾子が指定されているため書き換えることができない．そのため，write システムコールのバッファの複製を作成し，複製したバッファ内のデータをアスタリスクに書き換えることでアクセス制御を実現する．write システムコールのアクセス制御は，以下の手順で行う．

(1) write システムコールのバッファの複製を作成する．
(2) write システムコールのバッファをページサイズ毎に分割し，そのデータ内にタグが設定されていないか確認する．
(3) タグが設定されていた場合，1 バイトずつタグを取り出し，そのタグと対応するポリシから出力を禁止するべきかを判定する．
(4) 出力を禁止する場合，禁止したいデータと対応する複製されたバッファ内のデータをアスタリスクに書き換える．
(5) 出力を禁止するデータが含まれていた場合は，kernel_write 関数で複製したバッファをファイルに書き込む．出力を禁止するデータが含まれていない場合は，通常通り write システムコールを実行する．

write システムコールをページサイズ毎に分割しているのは，バッファサイズがページサイズより大きい場合に，連続で物理メモリにマップされていない可能性があるためである．TA-Salvia がタグの存在を確認するとき，先頭の物理アドレスとサイズを指定し，その範囲内のタグを確認する．バッファが連続でマップされていない場合，TA-Salvia がバッファ外のデータを確認してしまい，誤判定してしまう．TA-Salvia は，これを防ぐためにページサイズ毎に分割している．出力を禁止するデータが含まれていた場合に，通常の write システムコールではなく，kernel_write 関数で書き込んでいるのは，複製したバッファがカーネル空間内に確保されているので，通常の write システムコールを利用できないためである．そのため，TA-Salvia は，カーネル空間のバッファを書き込む kernel_write 関数を用いてデータを書き込んでいる．

send と sendto システムコールは，write システムコールと同様に第 2 引数のバッファを書き換えることで実現する．writev システムコールは，複数のバッファをまとめて書き込むシステムコールである．このシステムコールは，アクセス制御機能を追加した write システムコールをバッファの個数回呼び出すように実装を変更することでアクセス制御を実現した．

4.4.2 sendfile と sendfile64 システムコール

sendfile と sendfile64 システムコールは，あるファイルディスクリプタから別のファイルディスクリプタにデータをコピーするシステムコールである．このコピーは，カーネル内に用意されたパイプを通して行われる．sendfile と sendfile64 システムコールのアクセス制御は，データをパイ

プに書き込んだ後に行う．このパイプへの書込みは，ページサイズ毎に行われるため，分割してアクセス制御を行わなくてよい．TA-Salvia は，パイプにデータを書き込む関数からページを取得し，これをカーネルに一時的にマップする．TA-Salvia は，マップしたページ内のデータをアスタリスクに書き換えることでアクセス制御を実現する．

4.4.3 mmap システムコール

mmap システムコールは，ファイルをメモリにマップするシステムコールである．ファイルをメモリにマップすることで，メモリ操作と同じようにファイルを操作することが可能となる．mmap システムコールのアクセス制御は，メモリへの書込みのタイミングを把握することが困難であるため，書込みが終わったデータがディスク上のファイルと同期を取るときに行う．具体的には，同期処理に使用される request_queue にダーティページが格納される直前をフックし，そのページ内のデータをアスタリスクに書き換えることでアクセス制御を実現している．

5. 評価

5.1 機能評価

5.1.1 cat コマンドとリダイレクトを用いた検証

cat コマンドとリダイレクトを用いて，別々のポリシを持つ 2 のファイルを 1 つのファイルに統合する．統合されたファイルのデータを確認し，TA-Salvia がバイト単位で出力を制御できるかを確認する．この検証のために，user1 と user2 というユーザを追加し，それぞれのユーザが所有者の 2 つのファイル（user1.txt と user2.txt）を作成した．また，user1.txt のデータの先頭から 10 バイト分に 1 番のタグを，user2.txt のデータの先頭から 10 バイト分に 2 番のタグを設定している．さらに，TA-Salvia には，あらかじめ 1 番のタグに「root と user1 による書込みを許可する (1)」，2 番のタグに「root と user2 による書込みを許可する（2)」というポリシを紐付けている．この検証は，root が cat コマンドとリダイレクトを利用することで，2 つのファイルを 1 つのファイル（user1_user2.txt）に統合し，各ユーザがそのファイルを cat コマンドで標準出力に書き込むことで，アクセス制御を行えているかを確認する．

cat コマンドとリダイレクトを用いた検証の結果を**図 3** に示す．まず，1 行目で root は，cat コマンドとリダイレクトを用いて 2 つのファイルを 1 つのファイルに統合した．次に，root は，cat コマンドを用いて user1_user2.txt 内のデータを出力した．root は，どちらのポリシにも違反しないので，通常通りに出力された．その後，root から user1 に切り替えて，同様に cat コマンドを実行した．このとき，8 行目を見ると，user2.txt からコピーされたデータがアスタリスクに変換されていることが分かる．これは，user2.txt からコピーされたデータが（2）のポリシに違反しているからである．同様に，user2 が cat コマンドを実行した場合，user1.txt からコピーされたデータが（1）のポリシに違反しているため，TA-Salvia によって出力が禁止された．このことから，TA-Salvia によって，ポリシに違反したデータの出力だけを禁止できていることが確認できた．

```
1  root@salvia:/home/salvia# cat user1.txt user2.txt > user1_user2.txt
2  root@salvia:/home/salvia# cat user1_user2.txt
3  USER1 DATA
4  USER2 DATA
5  root@salvia:/home/salvia# su user1
6  user1@salvia:/home/salvia$ cat user1_user2.txt
7  USER1 DATA
8  **********
9  user1@salvia:/home/salvia$ exit
10 exit
11 root@salvia:/home/salvia# su user2
12 user2@salvia:/home/salvia$ cat user1_user2.txt
13 **********
14 USER2 DATA
```

図 3 cat コマンドとリダイレクトを用いた検証結果

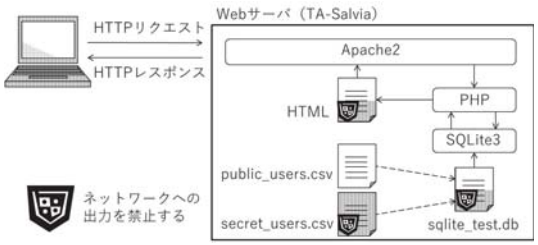

図 4 Web サーバの構成

5.1.2 Web サーバを用いた検証

これまでより複雑で実環境に近い環境として，Apache2，PHP5，SQLite3 を組み合わせた環境を用意した．この環境で評価を行うことで，TA-Salvia が実環境においても利用可能であることを示す．用意した Web サーバの構成を**図 4** に示す． Apache2 で PHP5 を動作させ，PHP のスクリプトを用いて SQLite3 のデータベースにアクセスする．SQLite3 のデータベースには，2 つのテーブル（PUBLIC_USERS と SECRET_USERS）を用意している．これらのテーブルは，INTEGER 型の id と TEXT 型の name の 2 つの要素を持つ．これらのテーブルに 2 つの CSV ファイル（public_users.csv と secret_users.csv）を SQLite3 にインポートして各テーブルに挿入している．secret_users.csv のデータすべてに 1 番のタグを設定している．また，TA-Salvia には，あらかじめ 1 番のタグに「ネットワークへの出力を禁止する」というポリシを紐付けている．これにより，secret_users.csv が源のデータの出力だけを伏せ字に変換できているかを確認する．ただし，前提の設定として Apache2 の mod_deflate モジュールを無効化しておく．このモジュールは，転送するデータを gzip で圧縮することで転送量を削減するモジュールである．Apache2 は，デフォルトで mod_deflate モジュールが有効になっており，これを無効にしない場合，gzip 圧縮時に暗黙的フローが発生する．TA-Salvia が利用している Argos は，暗黙的フローが発生するとデータの追跡を行えない．これにより，TA-Salvia のアクセス制御に失敗してしまうため，この検

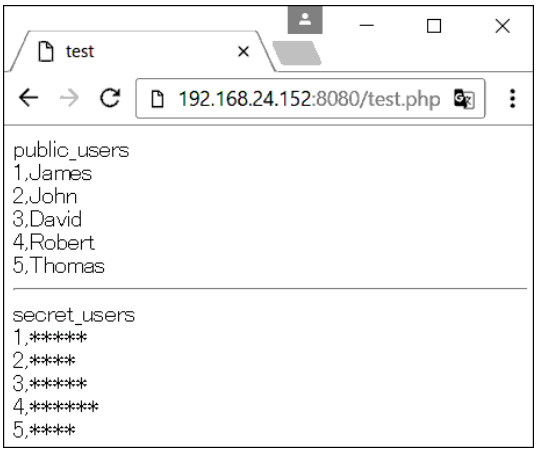

図 5 Web サーバを用いた検証結果

証では，mod_deflate モジュール無効にしている．

この検証では，別の端末からブラウザを用いて Web サーバにアクセスした．その結果を**図 5** に示す．ブラウザの上部には，PUBLIC_USERS テーブルのデータが表示されており，下部は，SECRET_USERS テーブルのデータが表示されている．PUBLIC_USERS テーブルのデータには，タグを設定していないので，通常通り表示されている．一方，SECRET_USERS テーブルのデータは，1 列目に数値，2 列目にアスタリスクが表示されている．このことから，TEXT 型のデータは，TA-Salvia で制御できているが，INTEGER 型のデータは，制御できなかったことが確認できた．制御に失敗しているのは，secret_users.csv のデータを SECRET_USERS テーブルに挿入したときに，文字列から数値に変換され，暗黙的フローが発生したことが原因である．TA-Salvia が利用している Argos は，暗黙的フローが発生するとデータの追跡を行えない．これにより，数値に変換されたデータの制御が行えなかった．今回の検証から，TA-Salvia は，明示的フローであれば，バイト単位でアクセス制御が可能であるということが確認できた．

5.2 性能評価

本論文では，cp コマンドを用いてファイルのコピーを行うことで，TA-Salvia によるアクセス制御のオーバヘッドを計測する．計測のために，ランダムなデータが書き込まれたサイズの異なる 3 つのファイルを用意した．用意したファイルサイズは，1MiB，10MiB，100MiB の 3 つである．cp コマンドを用いて，これらのファイルを別のファイルにコピーする．比較対象は，以下の 4 つである．比較対象は，QEMU-1.1.0 + Normal Kernel，Argos-0.7.0 + Normal Kernel，TA-Salvia （ポリシなし），TA-Salvia（write 禁止ポリシあり）の 4 つである．Normal Kernel は，何も変更を加えていないカーネルのことである．また，動作環境は，**表 1** の通りである．メモリは，1GB 分をゲストマシンに割り当てている．ディスクイメージは，5GB 分を用意している．

表 1 動作環境

ホスト CPU	Intel Core i7-902 2.67GHz
ゲスト OS	Debian 7 32bit（Kernel 3.9.9）
ゲストメモリ	1GB
ディスクイメージサイズ	5GB

表 2 cp コマンドの実行速度（単位：s）

	1MiB	10MiB	100MiB
QEMU-1.1.0 + Normal Kernel	0.143	0.386	3.834
Argos-0.7.0 + Normal Kernel	0.320	1.687	16.604
TA-Salvia（ポリシなし）	0.333	2.017	63.797
TA-Salvia（write 禁止ポリシあり）	5.716	55.044	546.068

動作結果を**表 2** に示す．まずは，QEMU と Argos の比較を行う．Argos は，QEMU を拡張して動的テイント解析機能を追加している．実行速度が QEMU より Argos のほうが遅いのは，動的テイント解析のオーバヘッドによるものである．次に Argos と TA-Salvia（ポリシなし）の比較を行う．TA-Salvia は，ファイル書込み時にタグが設定されているかの確認を行う．実行速度が Argos より TA-Salvia（ポリシなし）のほうが遅いのは，このタグ確認処理によるものである．最後に TA-Salvia のポリシなしとポリシありの場合の比較を行う．TA-Salvia は，書き込むデータにタグが設定されていた場合，1 バイトずつタグを取り出して，伏せ字に置き換えるという処理を行う．1MiB のファイルの場合，TA-Salvia は，約 100 万回のタグ取出しを行っており，100MiB のファイルの場合，約 1 億回のタグ取出しを行っている．実行速度がポリシなしよりもポリシありのほうが大幅に低下しているのは，このタグ取出し処理によるものである．

6. 関連研究

シャドウディスクを用いた研究に API Chaser[10] と TEMU[11] がある．API Chaser と TEMU は，共に QEMU をベースに開発されており，動的テイント解析機能を持つ．API Chaser は，解析対象のマルウェアのプログラムコードにタグを付与し，追跡することでコードの識別を可能にする．API Chaser は，Argos を用いて追跡するため，マルウェアが他のプロセスにプログラムを注入したとしてもコードの識別が可能である．API Chaser は，シャドウディスクを用意することで，マルウェアが行ったファイル書込みや OS がメモリページをスワップアウトした場合でも，追跡できるようにしている．API Chaser のシャドウディスクは，セクタ番号，タグを保持するバッファで構成されたデータ構造体を定義し，その構造体を 1 つのノードとした 2 分木で管理されている．API Chaser は，ディスク上の全データに対応するエントリをあらかじめ確保するとメモリ消費量が膨大になってしまうため，タグを保持

しているデータのみメモリを割り当てている．TEMU は，マルウェア解析，リバースエンジニアリング，脆弱性の発見，ソフトウェアの動作テストなどに利用するために拡張可能な動的テイント解析用のプラットフォームとして開発された．TEMU も API Chaser と同様にシャドウディスクを用意することで追跡を途切れさせないようにしている．TEMU のシャドウディスクは，ディスクを 64 バイトごとに区切り，その index とタグを保持するバッファで構成されたデータ構造体を定義し，ハッシュテーブルを用いて管理している．API Chaser と TEMU は，主にスワップアウト時に追跡が途切れないようにするために用意している．そのため，そのため解析終了後は，シャドウディスクの内容を維持せずに破棄している．一方，TA-Salvia は，ポリシの管理のためにシャドウディスクを利用している．そのため，シャドウディスクのタグをシャットダウン後も維持する必要がある．TA-Salvia は，ディスクイメージとしてシャドウディスクを管理することで，シャットダウン後もタグを維持することができる．

7. おわりに

本論文では，ファイルよりもさらに粒度の細かいバイト単位でアクセス制御を行う手法について述べた．提案手法を実現するために，TA-Salvia を拡張して，ディスク上のファイルのデータにもタグを保持できるシャドウディスクやバイト単位で出力を制御する機構を追加した．これにより，ファイル毎に管理してアクセス制御を行っていたことが原因で発生する過剰なアクセス制御の問題を解決することができる．また，評価では，別々のポリシを設定した複数のファイルを 1 つのファイルに統合した場合でも，個々のデータが持つポリシにしたがって制御できることを確認した．さらに，より実環境に近い環境として，Apache2，PHP5，SQLite3 を組み合わせた環境で評価を行った．評価結果から，gzip 圧縮や数値変換による暗黙的フローが発生した場合にアクセス制御を行えなかったが，明示的フローであれば正しくデータを追跡し，バイト単位でアクセス制御が行えることを確認できた．今後は，まだ TA-Salvia で実装できていないシステムコールへの対応やアクセス制御が行えなかった暗黙的フローへの対策を検討していく必要がある．

参考文献

[1] NPO 日本ネットワークセキュリティ協会：2015 年情報セキュリティインシデントに関する調査報告書【速報版】，http://jnsa.org/result/incident/index.html (2016).
[2] Loscocco, P. and Smalley, S.: Integrating Flexible Support for Security Policies into the Linux Operating System, *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference*, pp. 29–42 (2001).
[3] 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，*Proceedings of the Linux Conference 2004*, pp. 1–10 (2004).
[4] 松本隆志, 大月勇人, 明田修平, 齋藤彰一, 毛利公一：Argos の動的テイント解析機能を利用した OS による情報漏洩防止手法，研究報告コンピュータセキュリティ（CSEC），Vol. 2016, No. 33, pp. 1–8 (2016).
[5] Portokalidis, G., Slowinska, A. and Bos, H.: Argos: An Emulator for Fingerprinting Zero-day Attacks for Advertised Honeypots with Automatic Signature Generation, *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, EuroSys '06, pp. 15–27 (2006).
[6] 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. SIG3(ACS13), pp. 1–15 (2006).
[7] 内匠真也，奥野航平，大月勇人，瀧本栄二，毛利公一：コンパイラと OS の連携によるデータフロー追跡手法，情報処理学会論文誌，Vol. 56, No. 12, pp. 2313–2323 (2015).
[8] 奥野航平, 内匠真也, 大月勇人, 瀧本栄二, 毛利公一：コンパイラを用いた情報フロー制御による情報漏洩防止機構，研究報告コンピュータセキュリティ（CSEC），Vol. 2015, No. 13, pp. 1–8 (2015).
[9] Bellard, F.: QEMU, a Fast and Portable Dynamic Translator, *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '05, pp. 41–41 (2005).
[10] Kawakoya, Y., Iwamura, M., Shioji, E. and Hariu, T.: API Chaser: Anti-analysis Resistant Malware Analyzer, *The 16th International Symposium on Research in Attacks*, pp. 23–25 (2013).
[11] Yin, H. and Song, D.: TEMU: Binary Code Analysis via Whole-System Layered Annotative Execution, Technical Report EECS-2010-3, EECS Department (2010).