情報処理学会論文誌 Vol.53 No.2 847–856 (Feb. 2012)

ログの改ざんと喪失を防止するシステムの仮想計算機モニタによる実現

佐藤 将也[1,a)] 山内 利宏[1]

受付日 2011年5月30日, 採録日 2011年11月7日

概要：計算機の動作を把握するためには，計算機上のログが重要である．しかし，攻撃や問題の発生によりログの改ざんや喪失が起こる可能性がある．この問題に対処するため，仮想計算機モニタ（以降，VMM）を用いてログの改ざんと喪失を防止するシステムを提案する．提案システムでは，対象の仮想計算機（以降，VM）で動作している応用プログラム（以降，AP）とオペレーティングシステム（以降，OS）のログを，ゲスト OS のソースコードの修正なしに VMM が取得する．また，取得対象となる VM からログを隔離することで，取得したログの安全性を確保できる．さらに，ログを隔離し多重化することでログの改ざんを検出できる．

キーワード：ログの保護，セキュリティ，仮想計算機技術，デジタルフォレンジック

Logging System to Prevent Tampering and Loss with Virtual Machine Monitor

Masaya Sato[1,a)] Toshihiro Yamauchi[1]

Received: May 30, 2011, Accepted: November 7, 2011

Abstract: Logging information is necessary in order to understand a computer's behavior. However, there is a possibility that attackers will delete logs to hide the evidence of their attacking and cheating. Moreover, various problems might cause the loss of logging information. To address these issues, we propose a system to prevent tampering and loss of logging information using a virtual machine monitor (VMM). In this system, logging information generated by the operating system (OS) and application program (AP) working on the target virtual machine (VM) is gathered by the VMM without any modification of the kernel source codes. The security of the logging information is ensured by its isolation from the VM. In addition, the isolation and multiple copying of logs can help in the detection of tampering.

Keywords: protection of log, security, virtualization technology, digital forensics

1. はじめに

AP や OS の出力するログは，計算機の動作を正確に把握するために重要である．しかし，攻撃や情報漏えいの痕跡を消すためにログが改ざん，または消去される可能性がある．また，計算機の障害，プログラムの誤動作，利用者の誤操作，またはプログラムの問題により，ログが喪失する可能性がある．

サーバやデスクトップ用途で広く利用されている Linux では，ログの書き出しに syslog プログラムを用いており，攻撃者や不正者がファイルに出力された AP によるログ（ユーザログ）やカーネルによるログ（カーネルログ）を改ざんできる．また，syslog のプログラム自体が改変された場合，出力されたログは信頼できなくなる．さらに，短い間に大量のログが出力された場合，カーネル内のリングバッファに格納されたカーネルログが上書きされ，古いログが喪失することがある．

[1] 岡山大学大学院自然科学研究科
Graduate School of Natural Science and Technology, Okayama University, Okayama 700–8530, Japan
[a)] m-sato@swlab.cs.okayama-u.ac.jp

このようなログの保護に関する研究として，デジタルフォレンジックがある．デジタルフォレンジックとは，法的紛争や訴訟に対し，電磁的な記録の法的な証拠性を明らかにするための科学的調査手法，および調査技術である．デジタルフォレンジックの分野では，ログ情報の保護に関する様々な方式が研究されている [1], [2], [3], [4], [5], [6]．しかし，これらの研究は，主にログファイルの保護を目的としており，ログの書き出しプログラムが攻撃されるなど，ファイルへログが書き出される前に攻撃を受けた場合には対処できないものが多い．

そこで，本論文ではこれらの問題へ対処するため，仮想計算機モニタにより，ログの改ざんと喪失を防止するシステムを提案する．提案システムでは，ログを取得する対象の OS（以降，監視対象 OS）を VM 上で動作させる．また，監視対象 OS 上のユーザログとカーネルログを監視対象 OS のソースコードの修正なしに VMM でも取得する．ユーザログは，監視対象 OS のシステムコールを VMM によりフックすることで，ログの出力を検知し，取得する．提案システムは，ユーザプロセスにおけるユーザログ送信システムコールの発行直後にログを取得するため，ユーザログ送信システムコール終了後のログに対する改ざんなどの攻撃の影響を受けない．また，カーネルログは，バッファへログが出力される直前に現在のバッファの内容を VMM が取得する．これにより，次のログがバッファへ書き込まれる際に，前回出力されたログを必ず取得できる．このため，まだ取得されていない古いログがバッファの上書きにより喪失する問題へ対処できる．本論文では，提案システムの Linux を対象とした実現方式，および評価結果について述べる．

2. 既存のログ保存方式

2.1 syslog

Linux 2.6.26 上で調査した syslog の処理を**図 1** に示し，以下で syslog によるユーザログとカーネルログの取得方法について述べる．

syslog デーモン（sysklogd）に対するユーザログの送信は，ライブラリで提供される syslog 関数を利用する．syslog 関数は，send または write システムコールで/dev/log へログを送信する．syslog デーモンは，/dev/log に対して read システムコールを発行することでユーザログを取得する．

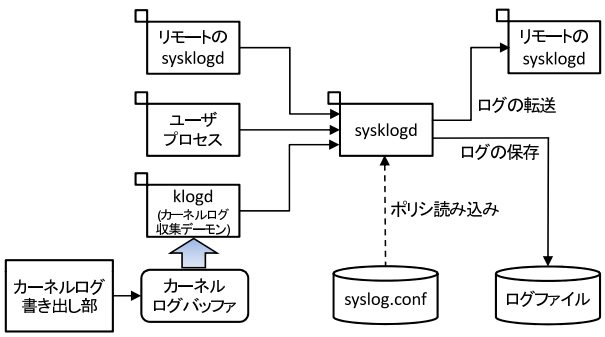

図 1 syslog の処理
Fig. 1 Architecture of syslog.

カーネルは，内部のバッファ（以降，カーネルログバッファ）にログを蓄積する．klogd はカーネルログバッファからログを取得し，syslog デーモンへログを送信する．

また，syslog はログの振り分け機能を持つ．syslog デーモンは，起動時に読み込んだ設定ファイル（syslog.conf）のポリシに従い，書き出し先のファイルを決定する．

syslog には，以下の問題がある．

(1) ログファイルの改ざんが可能
ログファイルへのアクセス権限を持つユーザは，意図的にログを改ざんできる．

(2) syslog デーモンの動作の変更によるログの書き出しの停止
syslog デーモンは，設定ファイルを改ざんされることで，ログを書き出さないように動作を変更される可能性がある．また，syslog デーモン自体が改ざんされると，ログを書き出さない可能性がある．

(3) カーネルログの構造に関する問題
klogd が長期間ログを収集しない場合，またはログが収集されるよりも短い期間に大量のカーネルログが出力された場合，古いログは新しいログで上書きされ，喪失する．

管理方法の簡易化やセキュリティを向上させた syslog デーモン [7], [8] が開発されているものの，上記の問題は解決されていない．

2.2 ログファイルの保護

ファイル分散保存システムを用いたファイルの保護手法 [1] が提案されている．この方式は，保護するファイルを暗号化し，分割した後に電子署名を施す．これを LAN 内の計算機のディスク上に分散保存する．また，ファイル削除への耐性を持つログ情報の保護手法として逃げログ [2] が提案されている．この手法では，ファイルのバックアップを複数作成し，ファイルシステム内に保持する．定期的に原本のファイルとバックアップを比較することで，不正な操作によるファイルへの変更があった場合，改ざんを検知し，改ざん内容を修復することができる．これらの研究は，ファイルシステムによりファイルを保護している．このため，いずれの場合も，攻撃者がファイルシステムを解析し，攻撃を加える可能性がある．

文献 [3] では，ファイルの操作履歴を仮想化技術を用いて保護する手法が提案されている．この手法では，あるゲスト OS 上のファイルシステムへ送られたファイルの read/write 操作を VMM がフックし，異なるゲスト OS へ保存する．ファイルの操作履歴を保護することで，ファイ

ルの改ざんの検知や復元が可能となる．この手法は，仮想化技術によりログを物理的に隔離するため，ファイルシステムへの攻撃によるログの改ざんを防止できる．しかし，この手法はファイルシステムの操作内容のみを取得するため，syslog で扱うようなシステムの動作履歴の保護を対象としていない．

また，ファイルの完全性の保護手法として，ヒステリシス署名を用いたものがある．しかし，ヒステリシス署名は，その連鎖をたどることでファイルを改変される問題がある．この問題への対処として，セキュリティデバイスを用いる方式が提案されている [4]．この方式では，連鎖用のデータとしてファイルではなく，セキュリティデバイス内の耐タンパ領域にある情報を用いる．これにより，ヒステリシス署名の問題へ対処している．

これらの研究は，ファイルに書き出されたログを保護するためのものである．提案システムでは，ログの出力要求時にログを取得し保護するため，これらの方式よりもログの取得契機が早く，より確実にログを保護できる．

2.3 syslog の保護

文献 [5] では，syslog 自体の正当性を保証する研究が提案されている．この方式では，Trusted Platform Module（TPM）によるトラステッドブートと Secure Virtual Machine（SVM）による Late Launch を組み合わせ，syslog デーモンが改ざんを受けていないことを保証する．正当性を保証された syslog は，受け取ったログを他計算機上の syslog へ転送する．しかし，この方式では，syslog の設定ファイルへの攻撃者による改変へ対処できない．また，カーネルログの喪失には対処できない．

2.4 独自のログの保護手法

文献 [6] では，syslog による既存のログ管理方式ではなく，監査用の独自のログ取得機構が提案されている．この手法では，Linux Security Modules（LSM）を利用してログを取得し，取得したログを保存したファイルの完全性を保証するため Mandatory Access Control（MAC）によるアクセス制御を用いている．また，ルートキットによるアクセス権限の変更への防衛策として DigSig [9] を利用している．DigSig はプログラムに署名を付加し，実行時に検証することで未知のプログラムの実行を防止する機構である．また，SecVisor [10] を用いることで，LSM によるログ取得機構と DigSig の安全性を保障している．この手法は，ログの取得と保存において安全性を確保できる．しかし，カーネルを改変するため，バージョンアップによるカーネルの変更を強く意識する必要がある．カーネルの改変は一般的に困難であり，Linux は頻繁にカーネルが更新されている．このため，継続的な利用が困難である．

2.5 問題のまとめ

これまでに述べた方式には，以下のいずれかの問題が存在する．

（**問題 1**） ログへの攻撃
（**問題 2**） ログの保護機構への攻撃
（**問題 3**） カーネルログの喪失
（**問題 4**） 適用可能な OS のバージョンの限定
（**問題 5**） 導入の困難さ

ログを保護する研究では，（問題 1）への対処が最も重要である．しかし，文献 [3], [5] 以外の方式では，ログが同一計算機のファイルシステム内に存在する限り，ファイルシステムの解析により，ログを攻撃される可能性がある．（問題 2）については，文献 [3], [6] 以外の方式では，十分ではない．また，（問題 3）へ対処した研究は，著者らが調べた限りでは存在しない．また，上述した方式の多くは，Linux カーネルに改変を加えるものである．Linux は，活発に開発が進められており，カーネルの更新が多い．カーネルに改変を加える手法では，更新が起こるたびにカーネルに修正を加える必要がある．このため，（問題 4）と（問題 5）の問題がある．カーネルの修正は一般的には困難な作業であるため，カーネルの更新によるログの保護機構の修正は最小限にとどめるべきである．

3. 仮想計算機モニタによりログの改ざんと喪失を防止するシステム

3.1 システムへの要求

本論文では，2 章で述べた問題を解決するシステムを提案する．（問題 1）への対処では，ログを計算機レベルで隔離することが有効である．計算機内部でログを保護した場合，様々な攻撃によりログが攻撃される可能性がある．また，（問題 2）への対処では，ログの保護機構自体を AP と OS から隔離することが有効である．AP やカーネル内部でログを保護した場合，ログの保護機構自体が攻撃され，保護したログの信頼性が失われる可能性がある．さらに，（問題 3）への対処として，カーネルログを喪失前に取得する必要がある．つまり，リングバッファによるカーネルログの上書きへ対処する方法が必要である．最後に，（問題 4）と（問題 5）への対処として，OS に依存しない方式の実現がある．これにより，OS の実装を意識する必要がなく，様々な環境への適用が可能となる．また，カーネルに依存したシステムと異なり，カーネルのバージョンアップへ容易に対応でき，つねに最新のカーネルを利用できる．

各問題への対処から，提案システムへの要求は以下のようになる．

（**要件 1**） すべてのログの取得
（**要件 2**） 取得したログの隔離
（**要件 3**） ログの保護機構自体の安全性の確保
（**要望 1**） OS の種類やバージョンに依存しないシステム

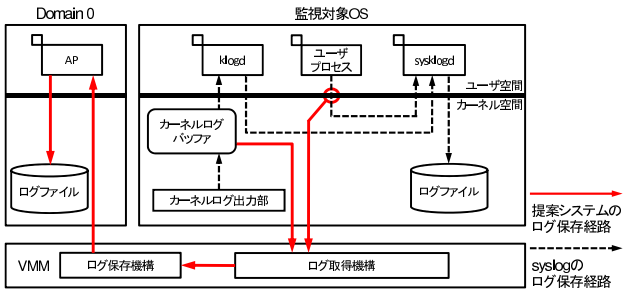

図 2 提案システムの全体像
Fig. 2 Architecture of the proposed system.

の実現

なお，（要望 1）は，ログの保護の観点からは要件とはならないが，重要な課題である．

3.2 提案システムの構成

提案システムの全体像を**図 2** に示す．提案システムでは，監視対象 OS を VM 上で動作させる．ログ取得機構とログ保存機構は VMM 内で動作する．また，監視対象 OS のソースコードを改変しないために，完全仮想化に対応した VMM を対象とする．なお，本論文では，VMM は安全であり，VMM の管理者は不正を行わないと仮定する．

（要件 1）については，ログ取得機構により監視対象 OS 内のユーザログとカーネルログの出力を検知し，取得することで満たす．ユーザログは，/dev/log への send または write システムコールにより出力される．ログ取得機構は，このシステムコールを検知し，ログを取得する．また，カーネルログは，カーネル内部の関数である printk 関数により，カーネルログバッファへ蓄積される．ログ取得機構は，printk 関数の実行を検知し，カーネルログバッファに蓄積されているログを取得する．

（要件 2）と（要件 3）については，ログ取得機構とログ保存機構を VMM 内で実現することで満たす．VMM 内で動作するログ取得機構が監視対象 OS のログを取得し，ログ保存機構へ送信するため，取得したログとログ取得機構は，監視対象 OS の外に隔離されている．このため，ゲスト OS への攻撃により，取得したログとログ取得機構自体が影響を受ける可能性は低い．

（要望 1）は，提案システムを VMM 内で実現し，監視対象 OS を完全仮想化環境で動作させることで実現する．提案システムは，監視対象 OS のソースコードを修正することなく，ログの出力時に VMM に制御を移行できる．ログ取得機構の実現に必要な情報は，監視対象 OS の printk 関数の開始アドレスやカーネルログバッファの領域などのシンボル情報のみである．

3.3 ログ取得機構

3.3.1 ユーザログ取得機能

本機能は，図 2 に示すように，ユーザプロセスが syslog デーモンへログを送信する際に発行するシステムコールを VMM がフックすることでログを取得する．このため，ユーザログ取得機能では，監視対象 OS のソースコードを修正することなく監視対象 OS で発生するシステムコールを VMM により検知する仕組みが必要となる．そこで，提案システムでは，文献 [11] で用いられたゲスト OS のシステムコールの発行によりページ例外を発生させる手法を用いた．この手法では，監視対象 OS の sysenter_eip_msr レジスタの内容を監視対象 OS がアクセスを許可されていない場所へ書き換える．これにより，システムコールの発行でページ例外を発生させ，VMM へ処理を移行させる（VM Exit）[12]．VMM へ処理が移行した後，監視対象 OS のユーザログを取得し，ページ例外の発生を隠ぺいした後，監視対象 OS の処理を再開させる．これにより，監視対象 OS は，システムコールを VMM にフックされたことを意識せずに動作できる．

システムコールのフックによるユーザログの取得手順は以下のとおりである．なお，ログを送信するために，ユーザプロセスは事前にソケットを作成し，/dev/log ソケットファイルに対して connect を発行する．

(1) プロセスごとに，ログの送信に利用するソケット番号を特定する．具体的には，/dev/log ソケットに対する connect システムコールを検知し，connect システムコールの第 1 引数からソケット番号を取得する．
(2) (1) で取得したソケット番号への send と write システムコールを検知し，システムコールの第 2 引数で指定された文字列をログとして取得する．

なお，ログを送信したプロセスの識別には，プロセスごとにユニークな値が設定される CR3 レジスタを用いる．

3.3.2 カーネルログ取得機能

本機能は，監視対象 OS における printk 関数の実行を契機として，ログを取得する．提案システムでは，監視対象 OS 内にブレークポイントを設定する．監視対象 OS において，ブレークポイントを設定した場所に処理が到達すると，ブレークポイント例外が発生し，VMM へ処理が移行する．これを契機とすることで，VMM によるカーネルログ取得が可能となる．

提案システムでは，カーネルログを取得するために，監視対象 OS のカーネルログ出力部にブレークポイントを設定する．ブレークポイントの設定は，メモリ上へロードされているカーネルに INT3 命令を埋め込むことによって実現する．この方式では，メモリ上のデータを書き換えるため，カーネルのソースコードの修正は不要である．なお，INT3 命令の埋め込みは，提供しているすべての VM のメモリ空間を管理できる VMM により実現している．この埋め込み対象のメモリ領域は，監視対象 OS からは書き込み不可のままであるため，安全性は低下しない．

以下では，**図 3** を用いて，監視対象 OS へのブレークポ

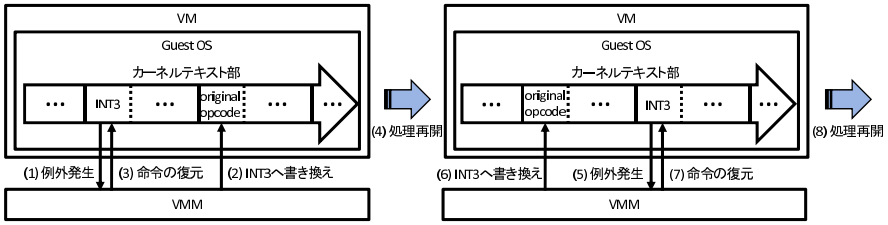

図 3 ゲスト OS へのブレークポイントの設定
Fig. 3 Breakpoint embedding in a guest OS.

イントの設定とカーネルログ取得の手順について述べる．なお，最初の INT3 命令は OS の起動直後に printk の開始アドレスに埋め込んでおく．

(1) ブレークポイント例外の発生により VMM へ処理が移行する．VMM へ処理が移行した後，カーネルログを取得する．
(2) 再度例外を発生させるため，次の命令の場所に INT3 命令を埋め込む．これは，(1) で例外を発生させた場所へ再びブレークポイントを設定する契機を得るためである．
(3) 例外が発生したアドレスのメモリ上の値を復元する．
(4) 復元した命令から監視対象 OS の処理を再開する．
(5) ブレークポイント例外の発生により VMM へ処理が移行する．
(6) 最初に設定していた場所に再度ブレークポイントを設定する．これにより，次に監視対象 OS が printk 関数を実行した際に，再び (1) と同じ場所でブレークポイント例外を発生させることができる．
(7) 例外が発生したアドレスのメモリ上の値を復元する．
(8) 復元した命令から監視対象 OS の処理を再開する．

VMM へ処理が移行すると，ログ取得機構が監視対象 OS のカーネルログバッファの状態を確認する．カーネルログバッファに新たに蓄積されたログが存在した場合，ログを取得する．その後，監視対象 OS の処理を再開させる．

3.4 ログ保存機構

ログ保存機構は，ログ取得機構の取得したログを保持する VMM 内のプログラムである．ログ取得機構により取得したログは，ログ保存機構に送信され，VMM が標準で保持するログバッファに格納する．提案システムでは，VMM として Xen を用いる．Xen における特権ドメイン（Domain 0）上で動作する VMM 管理用のデーモン（xend）は，VMM 内のログバッファの内容を読み込む機能を持つ．Domain 0 では，この機能を利用して，監視対象 OS のログを VMM から読み込み，ファイルへ書き出す．

4. 評価

4.1 評価の目的と評価環境

監視対象 OS で発生するログの喪失の防止，改ざんと喪失の検知，および攻撃への耐性について考察する．また，提案システムの導入により発生するオーバヘッドを測定した．

表 1 評価環境
Table 1 Environment used for evaluation.

OS	Domain 0	Linux 2.6.18-xen
	HVM Domain	Linux 2.6.26
VMM		Xen 3.4.1
syslog		rsyslogd 3.18.6
CPU		Intel Core 2 Duo E6600
Memory	Physical	2,048 MB
	Domain 0	1,024 MB
	HVM Domain	1,024 MB

評価環境を**表 1** に示す．VMM として Xen [13] を用い，監視対象 OS として Linux を用いた．Xen と CPU は完全仮想化に対応したものを用いた．監視対象 OS は完全仮想化しており，カーネルのソースコードにはいっさい修正を加えていない．なお，提案システムは，監視対象 OS が動作する VM のレジスタの値とメモリの書き換えのみにより実現している．このため，KVM などの Xen 以外の VMM を利用したシステムの実現も可能である．また，提案システムは，CPU の仮想化支援機能による完全仮想化を利用している．これは，監視対象 OS を書き換えないという要求を満たすためである．この要件により，監視対象 OS が Linux でない場合も，提案システムを適用できる．監視対象 OS を Linux に限定した場合，VM のレジスタの値とメモリの書き換えが可能であれば，CPU の仮想化支援機能を利用しない準仮想化においても提案システムを適用可能である．

4.2 ログの改ざんの防止

提案システムで取得したログは，監視対象 OS から独立した場所で保持する．このため，攻撃者やマルウェアが監視対象 OS の特権を奪取し，監視対象 OS 内であらゆる操作が可能になった場合でも，監視対象 OS から隔離したログを攻撃することは困難である．

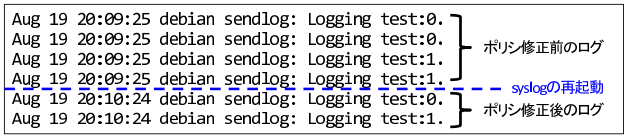

図 4 監視対象 OS 上のユーザログ
Fig. 4 A user log gathered by syslog.

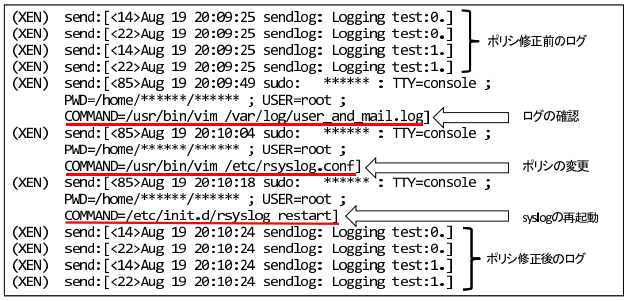

図 5 提案システムで取得したユーザログ
Fig. 5 A user log gathered by the proposed system.

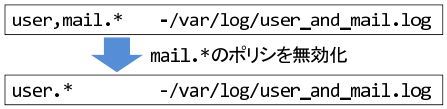

図 6 ログ書き出しポリシの変更
Fig. 6 Manipulation of configuration.

4.3 ログの喪失の防止

4.3.1 ユーザログの喪失の防止

攻撃者が設定ファイルを改ざんし，syslog の振舞いを変更することで，特定のログを書き出させない状況を想定して評価した．

図 4 と**図 5** は，同じ処理に対する監視対象 OS と提案システムのログである．図 4 には，図 5 の sudo コマンドについてのログが存在しないが，これは，監視対象 OS では，user と mail ファシリティのログしか出力していないためである．**図 6** のようにポリシを変更し，syslog の振舞いを変更させた前後でお互いのログを比較した．

図 4 では，ポリシ変更前は user と mail ファシリティのログがそれぞれ 2 回ずつ出力されているのに対し，ポリシ変更後ではログが少なくなっている．これは，ポリシの変更で除外された mail ファシリティのログが出力されていないことを示す．一方，図 5 では，ポリシ変更後もログの量は変わっていない．これにより，提案システムは，syslog の振舞いに関係なく，確実にログを取得できることを確認した．

4.3.2 カーネルログの喪失の防止

カーネルログが大量に出力された場合を想定し，監視対象 OS と提案システムのカーネルログを比較し，評価した．Debian 5.0.3 に標準で搭載されているカーネルでは，カーネルログバッファは 131,072 バイトである．このため，バッファを使いきる量のログを printk 関数により出力した．1 度の出力では，21 バイトの長さの文字列を指定した．なお，printk 関数では，内部でログ用のフォーマットへ変換するため，この場合の最終的な文字列の長さは 38 バイトである．このことから，printk 関数を 3,450 回発行すればバッファを使いきることになる．このため，本実験では printk 関数を 3,500 回発行した際に監視対象 OS と提案システムの双方で取得したログを比較する．

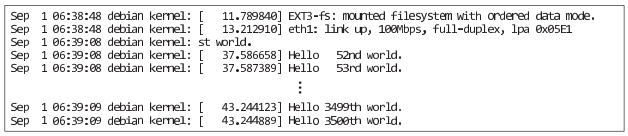

図 7 監視対象 OS 上のカーネルログ
Fig. 7 A kernel log of the target OS gathered by syslog.

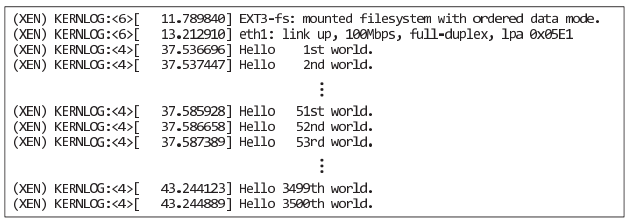

図 8 提案システムで取得したカーネルログ
Fig. 8 A kernel log of the target OS gathered by the proposed system.

監視対象 OS におけるカーネルログを**図 7** に，提案システムで取得したカーネルログを**図 8** に示す．図 7 の 3 行目では，カーネルログが不自然な位置から始まっており，3 行目以降のログでは 51 番目以降のログしか取得できていない．これは，監視対象 OS 上のカーネルログがバッファの上書きにより喪失したためである．一方，図 8 では，図 7 の 3 行目で途切れている部分もすべて取得できている．これにより，提案システムは，監視対象 OS では喪失するログをすべて取得可能であることを確認した．なお，さらに printk 関数の発行回数を増やすと，3,450 回分のすべてのログを上書きすることが可能であることを確認しており，カーネルログ保護の重要性が高いことが分かる．

4.4 ログの改ざんと喪失の検出

ログの改ざんと喪失の検出は，監視対象 OS と提案システムの双方で取得したログの比較により可能となる．このことを，監視対象 OS 上のログを改ざんした後，提案システムで取得したログと監視対象 OS 上の改ざんされたログとを比較し，確認した．

具体的には，syslog 関数を利用してユーザログを出力するプログラムとして，sudo を実行した．実行直後は，監視対象 OS のログファイルと提案システムで取得したログの内容は同一のものであった．この後，実行したコマンドの内容を隠ぺいするため，監視対象 OS のログファイル中の sudo で実行したコマンドの内容を書き換えた．これ以降，

監視対象 OS のログファイルと提案システムで取得したログを比較すると，sudo で実行したコマンドの内容が異なることを検出できた．

4.5 攻撃への耐性

ログの正当性を保証するためには，出力から保存されるまでの経路の安全性を確保する必要がある．このため，ログの保存経路の安全性について，既存方式と提案システムを比較する．

ユーザログは以下のタイミングで攻撃を受ける可能性がある．

(1) AP がログを生成したとき
(2) AP から syslog へログが送られるまでの間
(3) syslog がファイルへ書き出すまでの間
(4) ファイルへ書き出された後

syslog によるログの管理では，どのタイミングでログを攻撃されても改ざんの防止および検知ができない．このため，(1)〜(3) についてログの改ざんを検知するためには，ログを扱うプログラムの正当性を保証する必要がある．(3) については，プログラムのハッシュ値と Late Launch を用い，syslog と syslog から生成されたログの完全性を保証する研究がある [5]．しかし，この方式は TPM を用いる．TPM は性能を考慮した設計になっていないため，オーバヘッドが大きい．また，(4) については，文献 [4] のようにヒステリシス署名を用いることでログの改ざんを検知できる．また，ログの改ざん防止のために，ファイルシステムによる保護 [1], [2] を用いて対処する方法もある．これは，ファイルシステムを改良するため，監視対象 OS のソースコードを修正しないという（要件 4）を満たさない．

提案システムは，(2) でログ送信システムコールが発行された直後にログを取得し，OS がログ保存処理を行う前にログを取得し，保存できる．このため，提案方式において攻撃を受ける可能性があるのは (1) の場合のみであり，既存の方式や研究と比べて，ユーザログの安全性を高めることができる．(1) の場合に対処するには，ログを生成するすべてのプログラムが改ざんを受けていないことが保証される必要がある．プログラムの完全性を保証する手法として，起動時に署名を検査することでプログラムの完全性を保証する DigSig [9] が提案されている．しかし，この手法はプログラム起動時に署名を検査するため，カーネルを改変する必要がある．これは，（要望 1）を満たさない．

次に，カーネルログは以下のタイミングで攻撃を受ける可能性がある．

(1) カーネル内でログを生成するとき
(2) ログをカーネルログバッファへ出力するとき
(3) 収集されるまでにカーネルログバッファへ蓄積されている間
(4) カーネルログデーモンがログを収集するとき
(5) カーネルログデーモンから syslog へログが送られるまでの間
(6) syslog がファイルへ書き出すまでの間
(7) ファイルへ書き出された後

既存方式は，ルートキットなどによりカーネルが攻撃された場合，どのタイミングでログを攻撃されても対処できない．また，カーネルが安全であったとしても，ユーザログ同様，最終的には syslog がログを管理し，ファイルへ書き出すため，攻撃を受ける可能性がある．提案システムは，(2) を契機としてログを取得する．ここで，(2) のタイミングにおける攻撃としては，カーネルログバッファへログを出力するための関数が改変されることが考えられる．ただし，提案システムで取得するのは 1 つ前に出力されたログである．したがって，(3) の期間で攻撃される可能性がわずかに存在する．しかし，VMM で未取得のカーネルログは同時に 1 回分の出力しか存在しないため，カーネルログを自由に改ざんするのは簡単ではない．また，提案システムでは，(4) 以降の攻撃へ対処できるため，カーネル内のログ生成部または出力部へ攻撃されない限り，確実にログを取得できる．提案システムの実装を改良し，カーネルログを出力直後に取得できるようにした場合，(3) 以降のタイミングにおける攻撃へ確実に対処できる．

なお，(1) のタイミングでの攻撃へ対処するためには，カーネル内のログ出力に関する部分を改良する必要がある．これは，(要望 1) を満たさない．SecVisor [10] の利用により，(1) と (2) のタイミングでの攻撃へ対処できるが，SecVisor のみでは (3) 以降には対処できない．

なお，提案システムでは，VMM の管理者は信頼できるものと考えるため，管理者の不正によるログの改ざんや削除は想定しない．

4.6 提案システムの導入によるオーバヘッドの測定

4.6.1 ログの取得におけるオーバヘッドの測定

ユーザログの取得では，監視対象 OS 上で発行されるすべてのシステムコールをフックする．この仕組みでは，システムコールが発行されるごとに VMM へ処理が移行する．このため，システムコール呼び出し時の処理時間が増加する．そこで，syslog 関数の実行時に呼び出される主なシステムコールのオーバヘッド，および syslog 関数と printk 関数実行時のオーバヘッドを測定した．測定結果を**表 2** と**表 3** に示す．

表 2 では，syslog の利用時に呼び出されるシステムコールとして connect と write についてオーバヘッドを測定した．connect と write は，いずれも /dev/log へ open を発行した際のソケットに対して処理を行っている．また，getpid についてオーバヘッドを測定することで，システムコール発行時に監視対象 OS と Xen の間の遷移にかかる時間の指標とした．

getpid のオーバヘッドから，監視対象 OS と Xen の間の遷移には，約 2 μs 必要なことが分かる．connect の発行時には，/dev/log への接続かどうかを判定している．getpid のオーバヘッドから，この処理には約 3 μs 必要だと推察できる．また，write のオーバヘッドも同様に約 47 μs であることが分かる．write のオーバヘッドが他のシステムコールと比較して大きいのは，write で送信するログを VMM がコピーしているからである．

また，表 3 における syslog のオーバヘッドと表 2 における write のオーバヘッドの差が 5 μs 程度であることから，syslog 関数のオーバヘッドの多くは write によるものだと推察できる．write 以外のシステムコールのオーバヘッドは数 μs 程度であり，syslog 関数における約 54 μs のオーバヘッドと比べると，システムの性能に大きな影響はないと推察できる．

さらに，表 3 から，printk 関数のオーバヘッドは約 57 μs 程度であり，システムの性能に大きな影響はないと推察できる．この結果は，ユーザログ取得機能における syslog 関数のオーバヘッドとほぼ同じである．これは，printk 関数では，syslog 関数同様，内部でメモリ上のデータをコピーするためであると推察できる．

4.6.2 LMbench による性能測定

ログの送信に関係しないシステムコールの性能への影響を明らかにするため，LMbench 3.0 による性能測定を行った．測定結果を**表 4** に示す．

Null Call では，getppid の処理時間を測定しているが，これは，getpid とほぼ同様の処理をしており，表 2 における getpid と同様の結果であった．プロセスの生成と実行では，20〜50 μs 程度のオーバヘッドが発生している．また，0 KB および 10 KB のファイル生成では，5〜20 μs 程度のオーバヘッドが発生している．これは，プロセスの生成と実行，ファイルの生成とファイルへのデータ書き出し時のシステムコールを提案システムが検知することで発生するものだと推察できる．ファイルの削除では，データの書き出しが発生しないため，生成時よりもオーバヘッドは小さい．

4.6.3 提案システムの導入による AP の性能への影響

提案システムの導入による AP の性能への影響を評価するため，修正なしの Xen と提案システムを実現した場合の Xen において，監視対象 OS 上で動作する Web サーバの性能を測定し，比較した．評価に用いた Web サーバは，thttpd 2.25b である．thttpd は，標準の設定で syslog によりログを出力する．提案システムは，syslog を用いたログを検知し，取得するため，thttpd は提案システムの AP の性能への影響の評価に適している．評価では，ApacheBench 2.3 により，100 Mbps の通信路において，1 KB と 100 KB のファイルに対し，それぞれ 100 回ずつアクセスした際のスループットを測定した．測定結果を**表 5** に示す．表 5 の括弧内の数字は，修正なしの Xen の性能を 100%としたときの性能比を示している．

表 5 より，提案システムは，1 KB のファイルの要求では，修正なしの Xen の 96%，100 KB のファイルの要求では，修正なしの Xen の 98%の性能を維持していることが分かる．このことから，提案システムの導入による AP の性能の低下は小さいことが分かる．

表 2 システムコール発行時の提案システムにおけるオーバヘッド（単位：μs）

Table 2 Overheads in the proposed system when a system call was invoked (μs).

	connect	write	getpid
Xen	1.16	92.90	0.23
提案システム	6.55	142.20	2.23
オーバヘッド	5.39（465%）	49.30（53%）	2.00（870%）

表 3 syslog 関数と printk 関数実行時の提案システムにおけるオーバヘッド（単位：μs）

Table 3 Overheads in the proposed system when a syslog and printk are invoked (μs).

	syslog	printk
Xen	102.39	9.89
提案システム	156.56	67.34
オーバヘッド	54.17（53%）	57.45（581%）

表 5 thttpd のスループット（単位：要求数/s）

Table 5 Throughputs in thttpd Web server (Requests/s).

	1 KB File	100 KB File
Xen	1,009.92（100%）	87.88（100%）
提案システム	964.81（96%）	86.30（98%）

表 4 LMbench による測定結果（単位：μs）

Table 4 Results of LMbench process and file micro benchmarks (μs).

	Process			0 KB File		10 KB File	
	Null Call	Fork	Exec	Crete	Delete	Create	Delete
Xen	0.12	698	1,706	5.50	3.44	22.3	8.08
提案システム	2.16	753	1,725	11.3	6.67	40.0	11.8
オーバヘッド	2.04（1,700%）	55（8%）	19（2%）	5.8（105%）	3.23（100%）	17.7（79%）	3.0（37%）

5. 関連研究

5.1 システムやデータの保護

仮想化技術を用いてデータやシステムを保護する研究が行われている．文献 [14], [15] は，VM を利用した侵入検知システムを提案している．これらの研究では，侵入検知システムへの攻撃を防止するために仮想化技術を用いており，監視対象の VM 内の状態を特権 OS から監視する．さらに，文献 [15] は，監視する動作を限定することで，侵入検知システムの導入による性能の低下を抑制している．提案システムはこれらの研究と同様に，VMM を利用して監視対象の VM から情報を取得している．これらの研究は，侵入検知のための情報を VM から取得しているのに対し，提案システムは VM におけるログを保護するために情報を取得している．

文献 [11] は，VM を用いてマルウェアの挙動を解析する手法を提案している．また，文献 [16] は，VM によりサンドボックスを実現し，サンドボックス内でプログラムの動作を検証することで，不正プログラムを検知し，攻撃を防止する手法を提案している．これらの研究は，VM 上のプログラムの動作の解析を目的としている．一方，提案システムは，ログ出力の契機に着目し，システムの動作状況を示すログを取得し，保護することで，ログを用いた VM の動作の検証を実現している．

文献 [17] は，VM の動作履歴を収集し，VM の動作を再現する方式を提案している．この方式では，システムの動作状況を再現するための情報を VM から取得する．一方，提案システムは，システムの動作状況の把握のために用いられるログに着目し，取得することで，ログの保護とログの改ざん検出を実現している．

5.2 VMM の安全性

VMM は，VM を提供するための最小限の機能を実現したソフトウェアである．このため，OS に比べてコードの量が少ない．コードの量を比較すると，Linux 2.6.35 は 1300 万行以上である [18] のに対し，Xen は 15 万行に満たないと文献 [19] で述べられており，Xen のコードの量は監視対象 OS である Linux に比べて少ない．コードの量が少ないため，バグが混入する可能性は低く，脆弱性を利用して VMM を攻撃するのは OS を攻撃するよりも難しい．

OS を攻撃する手段として，デバイスドライバの脆弱性が利用される．デバイスの高機能化にともない，デバイスドライバは複雑になり，コードの量が増加している．このため，バグの混入による脆弱性が存在する可能性が高くなる．デバイスドライバの脆弱性を利用することで，攻撃者は任意のカーネルコードを実行可能になる．しかし，VMM はゲスト OS とは別のメモリ領域で動作しており，ゲスト OS のデバイスドライバの脆弱性を用いた VMM への攻撃は，OS への攻撃よりも難しい．

6. おわりに

監視対象 OS のログの改ざんと喪失を防止するシステムを提案し，その方式と評価結果について述べた．提案システムは，ログ出力要求時にユーザログとカーネルログを確実に取得できる機構と，VMM 内に取得したログを保存する機構を持つ．ユーザログ取得機能では，VMM がユーザログの出力を検知し，ログの書き出しプログラムが受信する前に取得する．AP のログ送信要求直後に提案システムがログを取得するため，AP のログ送信要求終了後にログを改ざんする余地を残さない．また，カーネルログ取得機能では，監視対象 OS におけるカーネルログの出力を検知し，ログを取得する．カーネルログの出力を契機とすることで，バッファ上の古いログが上書きされて喪失する問題を防止できる．これらの機能により，監視対象 OS のカーネルのソースコードを改変することなく，VMM によるログの取得が可能となる．本機構は，VMM で実現しているため，監視対象 OS から独立している．これにより，攻撃が困難な機構を実現した．

また，実現した機能について，ログの改ざんや喪失の起こる環境を想定し，評価した．評価結果から，監視対象 OS 上のログファイルの改ざん検出と syslog デーモンの動作の変更によるユーザログの喪失防止が可能なことを示した．また，大量にカーネルログが出力されることで古いログが喪失してしまうという問題について，提案システムの導入によりこの問題へ対処できることを確認した．

性能評価として，ユーザログとカーネルログそれぞれの取得で発生するオーバヘッドを測定し，ログのコピーを必要とする write システムコールでは，約 47 μs，他のシステムコールでは，約 2〜5 μs のオーバヘッドがあることを示した．また，LMbench を用いた性能測定により，測定した処理へのオーバヘッドは，約 5〜50 μs であり，その影響は小さいことを示した．

謝辞 本研究の一部は，科学研究費補助金若手研究（B）（課題番号：21700034）による．

参考文献

[1] 東森ひろこ，手塚 伸，宇田隆哉，伊藤雅仁，市村 哲，田胡和哉，星 徹：デジタルフォレンジックを目的としたファイル分散保存システムの実装および評価，情報処理学会論文誌，Vol.50, No.6, pp.1561–1574 (2009).

[2] 高田哲司，小池英樹：逃げログ：削除まで考慮にいれたログ情報保護手法，情報処理学会論文誌，Vol.41, No.3, pp.1–9 (2000).

[3] Zhao, S., Chen, K. and Zheng, W.: Secure Logging for Auditable File System Using Separate Virtual Machines, *IEEE International Symposium on Parallel and Distributed Processing with Applications*, pp.153–160 (online), DOI: 10.1109/ISPA.2009.32 (2009).

[4] Ashino, Y. and Sasaki, R.: Proposal of Digital Foren-

sic System Using Security Device and Hysteresis Signature, *Proc. 3rd International Conference on International Information Hiding and Multimedia Signal Processing* (*IIH-MSP 2007*), Vol.2, pp.3–7 (2007).
[5] Bock, B., Huemer, D. and Tjoa, A.: Towards More Trustable Log Files for Digital Forensics by Means of "Trusted Computing", *24th IEEE International Conference on Advanced Information Networking and Applications* (*AINA*), pp.1020–1027 (online), DOI: 10.1109/AINA.2010.26 (2010).
[6] Isohara, T., Takemori, K., Miyake, Y., Qu, N. and Perrig, A.: LSM-Based Secure System Monitoring Using Kernel Protection Schemes, *International Conference on Availability, Reliability, and Security*, pp.591–596 (online), DOI: 10.1109/ARES.2010.48 (2010).
[7] Adiscon's rsyslog: The enhanced syslogd for Linux and Unix rsyslog, available from ⟨http://www.rsyslog.com/⟩
[8] The free software company BalaBit: Syslog Server | syslog-ng Logging System, available from ⟨http://www.balabit.com/network-security/syslog-ng/⟩
[9] Apvrille, A., Gordon, D., Hallyn, S., Pourzandi, M. and Roy, V.: DigSig: Runtime Authentication of Binaries at Kernel Level, *Proc. 18th USENIX Conference on System Administration*, pp.59–66 (2004).
[10] Seshadri, A., Luk, M., Qu, N. and Perrig, A.: SecVisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity OSes, *Proc. 21st ACM SIGOPS Symposium on Operating Systems Principles*, pp.335–350 (2007).
[11] Dinaburg, A., Royal, P., Sharif, M. and Lee, W.: Ether: malware analysis via hardware virtualization extensions, *Proc. 15th ACM Conference on Computer and Communications Security*, pp.51–62 (2008).
[12] Intel: Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3B: System Programming Guide, Part 2, available from ⟨http://www.intel.com/Assets/PDF/manual/253669.pdf⟩ (2009).
[13] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A.: Xen and the art of virtualization, *Proc. 19th ACM Symposium on Operating Systems Principles*, pp.164–177 (2003).
[14] Garfinkel, T. and Rosenblum, M.: A Virtual Machine Introspection Based Architecture for Intrusion Detection, *Proc. Network and Distributed Systems Security Symposium* (2003).
[15] Azmandian, F., Moffie, M., Alshawabkeh, M., Dy, J., Aslam, J. and Kaeli, D.: Virtual machine monitor-based lightweight intrusion detection, *SIGOPS Oper. Syst. Rev.*, Vol.45, pp.38–53 (online), DOI: http://doi.acm.org/10.1145/2007183.2007189 (2011).
[16] Kevin, Z., Snow, S.K. and Monrose, F.: SHELLOS: Enabling Fast Detection and Forensic Analysis of Code Injection Attacks, *Proc. 20th Conference on USENIX Security Symposium*, pp.123–138 (2011).
[17] Dunlap, G.W., King, S.T., Cinar, S., Basrai, M.A. and Chen, P.M.: ReVirt: enabling intrusion analysis through virtual-machine logging and replay, *SIGOPS Oper. Syst. Rev.*, Vol.36, pp.211–224 (online), DOI: http://doi.acm.org/10.1145/844128.844148 (2002).
[18] Greg Kroah-Hartman, Jonathan Corbet and Amanda McPherson: Linux Kernel Development, The Linux Foundation (2010).
[19] Citrix Systems: Hypervisor - Leading Open Source Hypervisor for Servers, available from ⟨http://xen.org/products/xenhyp.html⟩

佐藤 将也 （学生会員）

2010 年岡山大学工学部情報工学科卒業．同年同大学大学院自然科学研究科博士前期課程入学．現在，在学中．コンピュータセキュリティ，仮想化技術に興味を持つ．

山内 利宏 （正会員）

1998 年九州大学工学部情報工学科卒業．2000 年同大学大学院システム情報科学研究科修士課程修了．2002 年同大学院システム情報科学府博士後期課程修了．2001 年日本学術振興会特別研究員（DC2）．2002 年九州大学大学院システム情報科学研究院助手．2005 年岡山大学大学院自然科学研究科助教授．現在，同准教授．博士（工学）．オペレーティングシステム，コンピュータセキュリティに興味を持つ．2010 年度情報処理学会論文賞受賞．電子情報通信学会，ACM，USENIX 各会員．