

仮想計算機モニタによる カーネルログ取得機能の実現と評価

佐藤 将也^{†1} 山内 利宏^{†1}

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。本論文では、提案システムのうち、OS の出力するログの改ざんの検出と喪失を防止する機能を実現し、評価した結果を報告する。

Implementation and Evaluation of Virtual Machine Based Kernel Log Collector

MASAYA SATO^{†1} and TOSHIHIRO YAMAUCHI^{†1}

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. This paper describes the implementation and evaluation of the mechanism that protects logging information generated by the OS.

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、古いログがファイルに保存されることなく喪失することがある。

このようなログの保護に関連する研究として、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)–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、ファイルヘ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、文献 7) 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した。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、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。

本論文では、文献 7) で述べたカーネルログ取得機能を再設計し、その実現方式を示し、評価結果について述べる。具体的には、カーネルログの取得は、ログを出力する関数の開始直後のメモリ上へブレイクポイントを埋め込むことで実現する、このブレイクポイントでの例外発生を契機とし、バッファヘ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、次のカーネル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

^{†1} 岡山大学大学院自然科学研究科

Graduate School of Natural Science and Technology, Okayama University

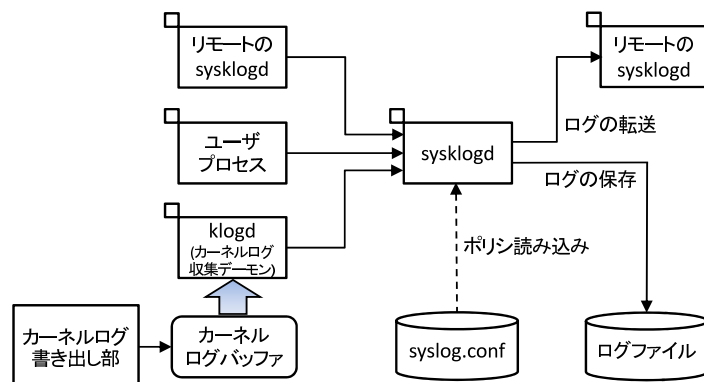


図 1 syslog の処理
Fig. 1 Architecture of syslog.

より喪失する問題へ対処できる。

2. 既存のログ保存方式

2.1 syslog

syslog は、Syslog working group⁸⁾ により標準化されているシステム管理とセキュリティ監視を目的としたクライアントサーバ型のプロトコルである。syslog は、syslog ライブラリと syslog デーモンから構成される。syslog ライブラリはログを送信するためのインタフェースを提供し、syslog デーモンはログの取得、分析、およびログファイルへの出力を行う。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。

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) のポリシーに従い、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。攻撃者は、root 権限を奪取することで、カーネルログを出力可能になる。カーネルログを自由に出力可能な場合、意図的にカーネルログを大量に出力し、有用なログを上書きされる可能性がある。

管理方法の簡易化やセキュリティを向上した syslog デーモン^{9),10)} 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。

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、その連鎖を辿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 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 DigSig¹¹⁾ 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、SecVisor¹²⁾ 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、本手法の継続的な利用は難しい。

3. 既存方式の問題点とシステムへの要求

3.1 既存方式の問題点

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留めるべきである。

3.2 システムへの要求

上記の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、常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。

4. 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

4.1 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。

(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

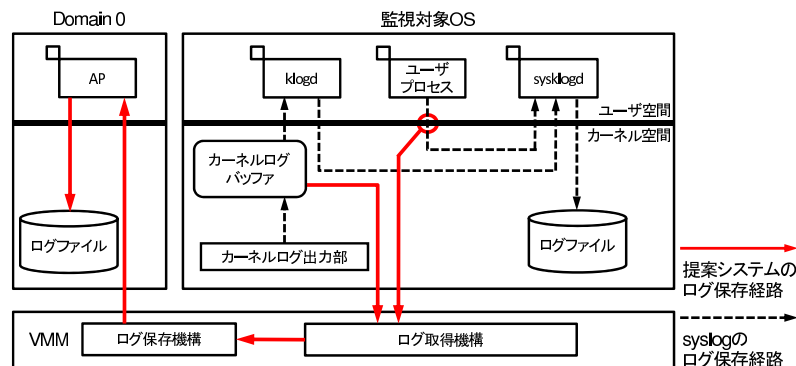


図 2 提案システムの全体像
Fig. 2 Architecture of the proposed system.

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。カーネルログ取得機能の詳細は 5 章で述べる。

(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、監視対象 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。

4.2 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 13) 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)¹⁴⁾。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通りである。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、CR3 レジスタを用いる。CR3 レジスタには、ページディレクトリの値が格納されているため、プロセスごとにユニークな値が設定されている。

4.3 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。

5. カーネルログ取得機能

5.1 実現方式

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。

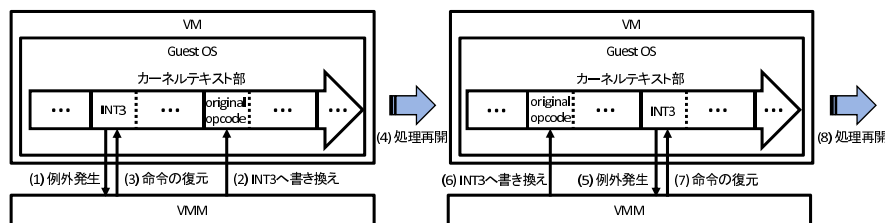


図 3 ゲスト OS へのブレークポイントの設定
Fig. 3 Breakpoint embedding in a guest OS.

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。

さらに、カーネルログ取得機能は、関数の実行を VMM が検知することでログを取得するため、監視対象 OS 上でログを収集するデーモンが停止しても継続してログを取得可能である。例えば、攻撃者が攻撃の痕跡を隠すためにログ収集デーモンを停止しても、提案システムではログを取得できる。

以下では、図 3 を用いて、監視対象 OS へのブレークポ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。

5.2 攻撃への耐性

ログの正当性を保証するためには、出力して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におけるカーネルログの取得機構を比較する。

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、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一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、カーネル内のログ生成部を改良する必要がある。これは、(要望 1) を満たさない。SecVisor¹²⁾ 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。

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

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

6. カーネルログ取得機能の評価

6.1 評価の目的と評価環境

監視対象 OS 上におけるカーネルログの改ざんと喪失の防止について評価する。また、カーネルログ取得機能の導入により発生するオーバーヘッドを測定した。

評価環境を表 1 に示す。VMM として Xen¹⁵⁾ 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は一切修正していない。

6.2 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 VMM 内で保持し、Domain 0 上でファイルへ保存する。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、あらゆる操作が可能になった場合においても、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。

6.3 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、バッファを使い切る量のログを printk 関数により出力した。一度の出力では、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切る

```
May 19 10:52:26 debian kernel: [ 11.890364] EXT3-fs: mounted filesystem with ordered data mode.
May 19 10:52:26 debian kernel: [ 14.043270] eth1: link up, 100Mbps, full-duplex, lpa 0x05E1
May 19 10:53:21 debian kernel: th world.
May 19 10:53:21 debian kernel: [ 241.698831] Hello 51th world.
May 19 10:53:21 debian kernel: [ 241.699593] Hello 52th world.
:
May 19 10:53:21 debian kernel: [ 249.187210] Hello 3498th world.
May 19 10:53:21 debian kernel: [ 249.187973] Hello 3499th world.
```

図 4 監視対象 OS 上のカーネルログ

Fig. 4 A kernel log of the target OS gathered by syslog.

```
(XEN) KERNLOG:<6>[ 11.890364] EXT3-fs: mounted filesystem with ordered data mode.
(XEN) KERNLOG:<6>[ 14.043270] eth1: link up, 100Mbps, full-duplex, lpa 0x05E1
(XEN) KERNLOG:<4>[ 241.646670] Hello 0th world.
(XEN) KERNLOG:<4>[ 241.646670] Hello 1th world.
:
(XEN) KERNLOG:<4>[ 241.698069] Hello 50th world.
(XEN) KERNLOG:<4>[ 241.698831] Hello 51th world.
(XEN) KERNLOG:<4>[ 241.699593] Hello 52th world.
:
(XEN) KERNLOG:<4>[ 249.187210] Hello 3498th world.
(XEN) KERNLOG:<4>[ 249.187973] Hello 3499th world.
```

図 5 提案システムで取得したカーネルログ

Fig. 5 A kernel log of the target OS gathered by the proposed system.

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。

監視対象 OS におけるカーネルログを図 4 に、提案システムで取得したカーネルログを図 5 に示す。図 4 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 5 では、図 4 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できることを確認した。なお、さらに printk 関数の発行回数を増やすと、3,450 回分のログをすべて上書きでき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わかる。

6.4 カーネルログの取得におけるオーバーヘッドの測定

カーネルログの取得では、printk 関数実行時にブレークポイント例外を発生させ、カーネルログバッファから VMM メモリをコピーする。このため、printk 関数実行時の処理

表 2 printk 関数実行時の提案システムにおけるオーバーヘッド (単位: μ s)
Table 2 Overheads in the proposed system when a printk is invoked (μ s).

	printk 関数の実行時間
Xen	9.89
提案システム	67.34
オーバーヘッド	57.45 (581%)

時間が増加する。そこで、改変を加えていない Xen と提案システムにおいて printk 関数の実行時間を測定し、比較することで、カーネルログ取得機能によるオーバーヘッドを明らかにする。測定結果を表 2 に示す。

表 2 から、printk 関数のオーバーヘッドは約 57μ s 程度であり、システムの性能への影響は小さいと推察できる。

7. 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のうち、カーネルログ取得機能の実現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。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 によるカーネル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、カーネル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、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問題は、提案システムの導入により対処できることを確認した。

性能評価として、カーネルログの取得によるオーバーヘッドを測定した。測定結果から、printk 関数の実行時のオーバーヘッドは約 57μ s であり、システムの性能への影響は小さいことを示した。

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 (2009).
- 4) Ashino, Y. and Sasaki, R.: Proposal of Digital Forensic System Using Security Device and Hysteresis Signature, *Proceedings of the Third International Conference on International Information Hiding and Multimedia Signal Processing (IIH-MSP 2007) - Volume 02*, 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 (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 (2010).
- 7) 佐藤将也, 山内利宏: 仮想計算機モニタによるログの改ざんと喪失防止システムの設計, 情報処理学会シンポジウムシリーズマルチメディア, 分散, 協調とモバイル (DI-COMO2010) シンポジウム, Vol.2010, No.1, pp.213–220 (2010).
- 8) IETF Syslog Working Group: IETF Syslog Working Group Home Page, <http://www.employees.org/~lonvick/index.shtml>
- 9) Adiscon’s rsyslog: The enhanced syslogd for Linux and Unix rsyslog, <http://www.rsyslog.com/>
- 10) The free software company BalaBit: Syslog Server | syslog-ng Logging System, <http://www.balabit.com/network-security/syslog-ng/>
- 11) Apvrille, A., Gordon, D., Hallyn, S., Pourzandi, M. and Roy, V.: DigSig: Runtime Authentication of Binaries at Kernel Level, *Proceedings of the 18th USENIX Conference on System Administration*, pp.59–66 (2004).
- 12) Seshadri, A., Luk, M., Qu, N. and Perrig, A.: SecVisor: a tiny hypervisor to provide lifetime kernel code integrity for commodity OSes, *Proceedings of 21st ACM SIGOPS Symposium on Operating Systems Principles*, pp.335–350 (2007).
- 13) Dinaburg, A., Royal, P., Sharif, M. and Lee, W.: Ether: malware analysis via hardware virtualization extensions, *Proceedings of the 15th ACM conference on Computer and Communications Security*, pp.51–62 (2008).
- 14) Intel: Intel 64 and IA-32 Architectures Software Developer’s Manual Volume 3B: System Programming Guide, Part 2, <http://www.intel.com/Assets/PDF/manual/253669.pdf> (2009).

- 15) 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, *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pp.164–177 (2003).