博士論文

ログ保全と攻撃難化による
セキュリティ向上技術に関する研究

佐藤 将也

2014年9月

岡山大学大学院
自然科学研究科

目次

図 目 次

表 目 次

概要

計算機は，低価格化と高度化により，官庁，企業，および大学等の組織での業務用途に留まらず，個人用途でも広く利用されている．また，計算機の扱う情報は，増加し，多様化している．このような環境において，計算機や情報システムの悪用により，サイバー攻撃が発生している．これにより，経済的被害や信用の低下などの被害が発生している．サイバー攻撃による被害の抑制において，計算機利用者の不注意や脆弱性を利用した攻撃を防止するためにどれだけ対策をしても十分なことはなく，侵入や情報の取得などが行われることを前提とした対策が必要不可欠になっている．そこで，まずは，攻撃者による侵入や目的の遂行の痕跡が記録される可能性のあるログの保全を確実にすることが重要である．また，攻撃者による目的の遂行を困難にすることができれば，攻撃による被害を抑制できる可能性がある．このことから，攻撃の検知や防止を行うソフトウェア（以降，セキュリティソフトウェア）を攻撃から保護することが重要である．

既存研究におけるログ保全と攻撃難化の方法の多くは，応用プログラム（Application Program，以降，AP）やオペレーティングシステム（Operating System，以降，OS）により実現されている．しかし，AP や OS による方法は．攻撃者により無効化される可能性がある．そこで，近年，仮想化技術による対処が注目されている．仮想化技術をログの保全や攻撃難化に利用することには，セキュリティソフトウェアの隔離，検査や監視の容易さ，および仮想計算機モニタ（Virtual Machine Monitor，以降，VMM）への攻撃の難しさといった利点がある．仮想化技術は，将来的には，より多くの環境で利用され，仮想計算機（Virtual Machine，以降，VM）の利用を前提としたソフトウェア構成が一般的になることが予想される．このため，VM の利用を前提とした環境において，利用者が計算機を安全に利用するためのソフトウェア実行基盤の構築が重要となる．しかし，既存研究における対処では，攻撃の証拠となり得るログの保全が十分でない．また，既存研究におけるセキュリティソフトウェアの保護は，セキュリティソフトウェアの VMM への移植や新規作成の工数が問題となる．そこで，本論文では，ログ保全の信頼性を向上するために，ログの改ざんや消失を VMM により防止するシステムを提案する．また，ログ保全の信頼性を向上しつつ，ログ保全のオーバヘッド

自然科学研究科
産業創成工学専攻

を抑制する手法として，ライブラリの置き換えによるVM外部への低オーバヘッドなログ転送手法を提案する．さらに，セキュリティソフトウェアの改変なしの保護を実現するために，プロセス情報の不可視化によりプロセスの特定を困難にする攻撃回避手法を提案する．

ログ保全の信頼性を向上するために，ログの改ざんや消失をVMMにより防止するシステムを提案した．このシステムは，VM上におけるログ出力をVMMにより検知し，ログを取得する．これにより，生成されてできるだけ早い段階でログを取得し，VM外部へ転送することで，攻撃者によるログ改ざんの可能性を低減する．また，Linuxにおいて発生するカーネルログ消失の問題について，リングバッファへのログ書き込みをVMMにより検知し，異なるVMに転送することで対処する．さらに，転送したログとVM上に存在するログを定期的に自動的に比較することで，ログの改ざんや改ざん箇所を検出する．このシステムをVM上のAPやOSを一切改変することなく，VMMの改変のみにより実現することで，攻撃者に機構自体を攻撃される可能性を低減し，かつ多種のOSが動作するVMへ提案手法を適用可能にする．このシステムについて，ログの改ざんや消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消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消失する問題について，提案システムの導入により対処できることを確認した．

ログ保全の信頼性向上は重要である一方で，ログ保全における性能低下が問題となる場合がある．このため，実際のAPの運用環境において，保全のためにログのVM外部への転送を実現するには，転送のオーバヘッドの削減が要求される．そこで，ログ保全の信頼性を向上しつつ，ログ保全のオーバヘッドを抑制する手法として，ライブラリの置き換えによるVM外部への低オーバヘッドなログ転送手法を提案した．ログの改ざんや消失を防止するシステムの性能低下の主な原因は，APからのログ転送をVMMにより検知するためのAPの動作監視である．このため，VM上にログ転送機能を挿入し，ログ転送の契機をVMMに明示的に通知することで，APの動作監視を不要にし，ログ転送のオーバヘッドを削減する．この手法について，評価結果より，本手法でログを転送するよりも前にログを改ざんするのは難しいことを示した．また，多種のOSへの適用の容易さを示した．性能評価では，入出力処理の負荷が高いAPにおいては，本手法の適用による性能低下は非常に小さいことを示した．さらに，複数VMが走行する環境における性能評価では，VM数の増加による性能低下への本手法の影響は小さいことを示した．

セキュリティソフトウェアの改変なしの保護を実現するために，プロセス情報の不可視化によりプロセスの特定を困難にする攻撃回避手法を提案した．この手法では，攻撃者から攻

撃対象プロセスの特定を困難にするために，プロセス情報へのアクセス制御とプロセス情報の置換を提案する．プロセス情報へのアクセス制御により，許可されていない領域からのプロセス情報の読み込みを禁止する．プロセス情報の置換では，プロセス情報の置換対象のプロセスの走行中は，本来のプロセス情報を利用させ，それ以外では偽のプロセス情報へ置換する．これらの機能により，プロセス情報をもとにした攻撃対象プロセスの特定を困難にする．また，この機能を VMM の改変により実現する．これにより，OS レベルで動作するマルウェアからの攻撃にも耐性のある実行環境の提供を目指す．評価より，重要プロセスの走行中は，重要プロセスのプロセス情報は偽の情報に置換できていることを確認した．

以上より，将来的に更なる普及が予想される仮想化技術の利用により，攻撃の痕跡の削除を困難にし，かつセキュリティソフトウェアへの攻撃を回避することで，計算機利用環境の安全性を向上できることを示した．その具体例として，VM 上のソフトウェアが出力するログの保全と VM 上で動作するセキュリティソフトウェアへの攻撃難化を実現する手法を示した．

自然科学研究科
産業創成工学専攻

第 1 章

序論

1.1 研究背景

1.1.1 攻撃の高度化

計算機は，低価格化と高度化により，官庁，企業，および大学等の組織での業務用途に留まらず，個人用途でも広く利用されている．また，計算機の扱う情報は，増加し，多様化している．計算機は，業務用途では，複数の計算機をネットワークに接続した情報システムとして運用されることがある．情報システムは，電子政府 [1] や勘定系システムなど，様々なサービスを企業や個人に提供している．また，企業におけるインターネット利用率も増加しており [2]，計算機や情報システムは，より多くの情報を管理し，相互にやりとりするようになっている．

計算機の扱う情報が増加し，多様化する中で，計算機や情報システムの悪用により，不正侵入，情報の取得，改ざんや破壊，情報システムの停止や誤作動，および悪意あるソフトウェア（以降，マルウェア）の実行や DDoS (Distributed Denial of Services) 攻撃などのサイバー攻撃が発生している．これにより，経済的被害や信用の低下などの被害が発生している．サイバー攻撃は，2000 年代初頭は，いたずら目的のものが多かった [3]．いたずら目的の攻撃は，単独の攻撃者による企業の Web ページ改ざんのように，技術力を誇示する攻撃であった．しかし，2004 年頃から金銭目的やサービス妨害を目的とした攻撃が増加しており，2009 年頃からは情報の取得を目的とした攻撃が増加している．近年被害が顕在化してきた標的型攻撃においては，攻撃者は，組織内ネットワークに侵入し，様々な手法を組み合わせて組織の保有する機密情報を取得する．米国においては，サイバー攻撃による経済的損失は最

大 140 億ドルに達するとの見積り [4] があり，経済的被害も大きいことが分かる．標的型攻撃により機密情報を取得された場合には，組織の信用は低下し，活動に悪影響を及ぼす可能性がある．このように，経済的被害や信用の低下が問題となっているため，サイバー攻撃による被害を抑制することは重要な課題である．

サイバー攻撃の多くは，以下の手順で行われる．

(1) 事前調査

(2) 侵入

(3) 目的の遂行

(4) 痕跡の削除

事前調査は，攻撃対象の組織内の計算機利用者や計算機上で運用されているサービスを調査する段階である．組織内の計算機利用者の調査は，計算機利用者の不注意を攻撃に利用するために行われる．組織内の計算機により運用されているサービスの調査は，サービスを提供しているソフトウェアの脆弱性を攻撃に利用するために行う．脆弱性は，ソフトウェアの設計や実装のバグのことであり，攻撃に利用される可能性がある．

侵入は，攻撃対象の組織内のネットワークに接続されている計算機上で攻撃活動を行うための準備段階である．攻撃者は，事前調査で収集した情報をもとに侵入することが多い．侵入は，利用者の不注意，およびソフトウェアの脆弱性等を利用する．利用者の不注意を利用した攻撃の代表的なものに，標的型攻撃で利用される標的型メールがある．標的型メールは，組織内の計算機利用者が普段の業務で受信するようなメールの文面を装い，マルウェアを添付ファイルとして送付する．組織内の計算機は，計算機利用者が標的型メールであることに気付かずに添付ファイルを開封することで，マルウェアに感染する．マルウェアは，組織外との通信経路であるバックドアを作成し，組織外からの侵入を可能にする．また，ソフトウェアの脆弱性は，組織内の計算機上で攻撃者の任意のコードを実行するために利用される場合がある．攻撃者は，脆弱性を持つソフトウェアに攻撃者の意図したコードを実行させ，実行させたコードで直接攻撃する場合やマルウェアをダウンロードさせて侵入する場合がある．

目的の遂行は，攻撃者の本来の目的を遂行する段階である．組織内の情報の取得が目的の場合，組織内の計算機に侵入した攻撃者は，組織内の情報を取得し，組織外に送信する．また，組織で運用しているサービスの動作の妨害が目的の場合，攻撃者は，サービスの運用に必要なプログラムやデータを破壊する．侵入された計算機は，他の計算機を攻撃するために利用される場合がある．

痕跡の削除は，攻撃の後処理の段階である．攻撃者は，目的を遂行した後に，攻撃者が組織内に侵入したことや目的を遂行したことを隠ぺいするために，侵入や目的の遂行の痕跡を削除する．組織内の計算機の利用者や管理者からは，侵入や目的の遂行の痕跡を削除されると，攻撃者による計算機への侵入，情報の取得，およびプログラムやデータの破壊を検知することが困難になる．また，組織内の計算機が他の計算機への攻撃に利用された場合，痕跡を削除されると，組織内の計算機利用者は，加害者と誤認される可能性がある．

サイバー攻撃による被害を抑制するには，上記手順のいずれかの段階で攻撃を検知し，防止する必要がある．上記手順のうち，計算機利用者の不注意を防止する必要があるため，(1) へ完全に対処するのは難しい．これは，計算機利用者の不注意を完全になくすのは，ソフトウェアによる対処では難しいためである．また，ソフトウェアの脆弱性を攻撃に利用されることを防止する必要があるため，(1)，(2)，および (3) へ完全に対処するのは難しい．脆弱性には，既知の脆弱性と未知の脆弱性がある．既知の脆弱性を利用した攻撃に対処するには，既知の脆弱性に対処したソフトウェアを利用すればよい．しかし，未知の脆弱性は，発見されるまで対処できない．このため，脆弱性を利用した攻撃に完全に対処するのは難しい．このように，計算機利用者の不注意や脆弱性を利用した攻撃を防止するためにどれだけ対策をしても十分なことはなく，侵入や情報の取得などが行われることを前提とした対策が必要不可欠になっている．一方，(4) への対処は，(1), (2), および (3) への対処に比べ，実現できる可能性が高い．これは，侵入や目的の遂行の痕跡は高い確率で記録されることから，この記録を攻撃者に削除される前に保全できれば，対処できるためである．

以上のことより，まずは，(4) への対処として，攻撃者による侵入や目的の遂行の痕跡が記録される可能性のあるログの保全を確実にすることが重要である．また，攻撃者が脆弱性を利用して攻撃した場合に，侵入を許したとしても，情報の取得，およびプログラムやデータの破壊など，攻撃者による目的の遂行を困難にすることができれば，攻撃による被害を抑制できる可能性がある．このことから，(3) への対処として，攻撃の検知や防止を行うソフトウェア（以降，セキュリティソフトウェア）を攻撃から保護することが重要である．

1.1.2 ログ保全と攻撃難化の必要性

ログ保全に関して，ログの証拠性を明らかにする手法として，デジタル・フォレンジックがある．個人や組織の行動の正当性を検証するデジタル・フォレンジックは，計算機が広く利用されるにつれ，必要性と有用性が高まっている [5]．デジタル・フォレンジックにおいて，事故や不正行為，犯罪に関わる機器に残されたデータから，電磁的証拠となり得るものを確

実に，そのままで，収集・取得し，保全しておくことは，最も重要である．これは，電磁的証拠の原本同一性に疑義が生じると，後の電磁的証拠の分析結果の信頼性を失うためである．

電磁的証拠の一つにソフトウェアの動作を記録したログがある．デジタル・フォレンジックでは，ログが電磁的証拠であることから，サイバー攻撃や組織内部の不正行為によるログの改ざんや，ソフトウェアの問題によるログの消失を防止することでログを保全する必要がある．被害の拡大や攻撃の成功率上昇のために，ログを改ざんするサイバー攻撃がある．サイバー攻撃によりログが改ざんされると，攻撃内容や被害状況の分析が困難になる．このため，サイバー攻撃によるログの改ざんを防止する必要がある．また，計算機の操作履歴を記録したログの保全は，組織内部の不正行為防止のためにも重要である [6, 7, 8, 9]．組織内部で不正行為が行われた場合，計算機の操作履歴が確実に保存されていれば，不正行為を行った犯人を特定できる可能性が高くなる．さらに，ソフトウェアの問題により，ログが消失する場合がある．このように，ログは電磁的証拠として有用であることから，ログの保全は，重要な課題である．ログを確実に収集・取得し，保全するためには，ログの改ざんや消失を防止する必要がある．

攻撃者は，目的遂行の成功率上昇のために，ルートキットと呼ばれるマルウェアを利用する場合がある．ルートキットは，セキュリティソフトウェアへの攻撃やセキュリティソフトウェアの利用する情報の改ざんにより，セキュリティソフトウェアの機能を無効化し，マルウェアの存在や攻撃の痕跡を隠ぺいする機能 [10, 11] を持つマルウェアである．セキュリティソフトウェアの機能を無効化するようなルートキット [12, 13, 14] は，攻撃対象の計算機に侵入した後，目的の遂行に不利となるセキュリティソフトウェアを無効化し，目的の遂行に利用するマルウェアやルートキット自身の検知を困難にする．Hsu らの調査 [15] では，調査対象とした合計 2,063 のマルウェア検体のうち 18 のマルウェアがセキュリティソフトウェアを停止または無効化する機能を持つと述べられている．セキュリティソフトウェアを停止または無効化された場合，攻撃者による侵入や目的の遂行の検知が遅れ，被害が拡大する可能性がある．このため，セキュリティソフトウェアを攻撃から保護することで，攻撃者による侵入や目的遂行の検知率を向上し，攻撃を難化する必要がある．

以上のように，ログの保全と攻撃難化が必要である．しかし，既存研究における対処は，攻撃者により無効化される可能性がある．既存研究におけるログの保全や攻撃難化の多くは，応用プログラム（Application Program，以降，AP ）やオペレーティングシステム（Operating System，以降，OS）により実現されている．既存研究は，その機能の実現に用いたソフトウェア自身が安全な場合は，有効である．しかし，管理者権限で動作するプロセスを攻撃者に乗っ取られた場合や，OS と同等の動作レベルであるスーパバイザモードで動作するマル

ウェアを攻撃者により挿入された場合には，既存研究による対処は無効化される可能性がある．そこで，近年，APやOSによる対処ではなく，仮想化技術による対処が注目されている．

1.1.3 仮想化技術

近年，仮想化技術は，広く利用されるようになっている．従来は，サーバ用途の計算機における資源の有効利用のために仮想化技術が用いられていた．近年では，従来と同様の用途で用いられる一方で，個人用途の計算機でも仮想化技術が利用されている．また，近年では，ネットワークを利用して計算機資源や情報を扱うクラウドコンピューティングが普及しており，企業や個人に対して仮想計算機（Virtual Machine，以降，VM）を貸し出すサービスが普及するなど，仮想化技術の利用は広まっている．

このように仮想化技術の利用が広まる中で，セキュリティソフトウェアを安全に実行するために仮想化技術を利用する手法が研究されている．仮想化技術では，VMを作成し，VM上でAPやOSを動作させる．VMは，仮想計算機モニタ（VM Monitor，以降，VMM）により管理され，一台の計算機上で複数のVMを走行可能である．また，それぞれのVM上では，異なるOSを走行可能である．VMMは，ベアメタル型 [16, 17, 18] とホスト型 [19, 20] に分類できる [21]．ベアメタル型は，計算機が起動すると，まずVMMが起動し，VMM上で各VMが走行する．一方，ホスト型は，OSの起動後，APとしてVMMを起動し，その上でVMが走行する．VM上で動作するOSをゲストOSと呼び，ホスト型のVMMを提供するOSをホストOSと呼ぶ．

仮想化の方式には，準仮想化と完全仮想化がある．VM上でAPやOSが動作する場合，特権命令やメモリ操作をVMがすべて実行できてしまうと，他のVMやVMMに影響を及ぼす．このため，VM上では，実行可能な命令が制限される．VMMは，VM上で実行された特権命令による例外などを検知し，エミュレートした結果をVMに返却する．この方式は，VMの処理にVMMが介在するため，性能が低下する．準仮想化は，特権命令など一部の命令をゲストOSがVMMへの依頼に変換して実行する．これにより，不要な例外の発生を抑制することで，VMの性能を向上している．しかし，ゲストOSが実行する命令をVMMへの依頼に変換するため，ゲストOSが準仮想化に対応している必要がある．完全仮想化は，VM上で実行される命令をVMMがエミュレートするため，準仮想化のようにOSを改変する必要はないものの，性能は低い．しかし，Intel Virtualization Technology（以降，VT）のような仮想化支援機能を搭載したCPUでは，VMX命令セットや動作モードにVMX rootモードとVMX non-rootモードが搭載されたことで，完全仮想化における性能低下を軽減している．VTの機能の一部であるVT-xを利用した場合，VMをVMX non-rootモードで走行

させ，VMM を VMX root モードで走行させる．VMX non-root モードでは，特権命令やセンシティブ命令と呼ばれる他の VM に影響を与える可能性のある命令が実行されると，VM exit が発生し，VMX root モードに処理が遷移する．VM 上での特権命令やセンシティブ命令の実行による VM exit が発生すると，VMX root モードへの処理の遷移に応じて VMM が命令を処理する．その他の命令は，VM 上で直接実行できる．このため，VT を利用する環境では，完全仮想化された VM の性能が向上している．

仮想化技術を攻撃の検知や防止，およびログの保全に利用することには，以下の利点がある．

(1) セキュリティソフトウェアの隔離

(2) 検査や監視の容易さ

(3) VMM への攻撃の難しさ

仮想化技術を利用することで，セキュリティソフトウェアを監視対象のソフトウェアから隔離できる．AP や OS による対処は，攻撃者により無効化される可能性がある．そこで，仮想化技術を利用し，VM 上の AP や OS からセキュリティソフトウェアを隔離することで，セキュリティソフトウェアの無効化を防止できる．

また，仮想化技術は，セキュリティソフトウェアによる検査や監視への利用が容易である．VMM は，VM を制御するためのソフトウェアであり，VM に割り当てたメモリへのアクセスや VM からのデバイスへのアクセスを監視できる．セキュリティソフトウェアは，攻撃の検知や防止に，ネットワークを流れるパケットや計算機上で発生したイベントを利用する．VMM は，これらの情報を取得可能であることから，セキュリティソフトウェアによる検査や監視に VMM は適している．

さらに，VM から VMM への攻撃は，AP や OS への攻撃よりも難しい．これは，VMM のコードの量の小ささと，VMM が提供する機能が限定されているためである．

VMM は，VM を管理するため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 3.2 は 1,500 万行以上である [22] のに対し，Xen は 15 万行に満たないと述べられており [23]，Xen のコードの量は 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 OS より低く，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といえる．

VMM は，提供する機能を限定しているため，VMM への攻撃は，OS への攻撃よりも難しい．AP は，主にサービスの提供に用いられることから，攻撃に利用されやすい．また，OS

は，AP の利便性向上のために，様々な機能を AP に提供している．この機能を利用して，OS が攻撃される場合がある．一方，VMM は，ゲスト OS に対して，VM 管理のための最小限の機能しか提供しない場合が多い．このため，VMM への攻撃は，OS への攻撃よりも難しい．

VMM への攻撃手法 [24] が報告されているものの，攻撃可能な環境は，限定されている．また，VMM の完全性検証によって攻撃を検知する手法 [25] が提案されているため，VMM への攻撃は，AP や OS への攻撃よりも，十分に難しい．

1.2 関連研究

ログ保全と攻撃難化について，実現に用いられる AP，OS，および VMM の 3 種類のソフトウェアごとに，関連研究を述べる．AP や OS は，攻撃への対処を実現するために，従来から利用されている．AP と OS では，攻撃の影響を受ける可能性や開発の難度が異なるため，ここでは，AP と OS それぞれを用いた関連研究を述べる．また，仮想化技術の高度化により，ログ保全と攻撃難化に VMM を利用することが現実的になってきている．以上のことから，AP，OS，および VMM の 3 種類のソフトウェアごとに，関連研究を述べる．表 1.1 に，ログ保全と攻撃難化の関連研究と実現に用いられるソフトウェアの関係を示す．

1.2.1 ログ保全

AP レベルでのログ保全の関連研究として，ログの完全性を考慮した syslog[26, 27, 28, 29] やログファイルの完全性検証 [30, 31, 32, 33] がある．syslog は，Linux におけるログ管理で標準的に利用されているプログラムやプロトコルである．ログの完全性を考慮した syslog デーモン [26, 27] や syslog プロトコル [28, 29] が提案されている．これらを用いることで，リモート計算機へ転送する通信路上でログが改ざんされた場合に，ログの改ざんを検出できる．しかし，改ざんの検出のみで防止はできない．また，syslog デーモンを置き換える攻撃 [66] や，OS を改ざんすることでログを改ざんする攻撃 [67] には対処できない．ログファイルの完全性検証 [30, 31, 32, 33] は，ログファイルへ署名を付与することで，ログファイルの改ざんを検出する．この手法も，改ざんの検出のみで，改ざんを防止できない．

OS レベルでのログ保全の関連研究として，ログファイルの改ざん検知と復元 [34]，ログ構造化ファイルシステム [35]，アクセス制御機構 [36, 37, 38, 39, 40]，暗号化ファイルシステム [41, 42]，およびファイルの分散保存 [43] がある．NIGELOG[34] は，ファイルシステム内にログファイルの完全性を検証する機構を実現している．これにより，ログファイルの改ざん

表 1.1 ログ保全と攻撃難化における関連研究と実現に用いられるソフトウェア

	AP	OS	VMM
ログ保全	• ログの完全性を考慮した syslog[26, 27, 28, 29] • ログファイルの完全性検証 [30, 31, 32, 33]	• ログファイルの改ざん検知と復元 [34] • ログ構造化ファイルシステム [35] • アクセス制御機構 [36, 37, 38, 39, 40] • 暗号化ファイルシステム [41, 42] • ファイルの分散保存 [43]	• syslog デーモンの保護 [44] • VM 上の情報取得 [45, 46, 47, 48] • 独自のログ保護手法 [49] • VM 上のファイルアクセスログの取得 [50]
攻撃難化	• IDS [51, 52, 53, 54]	• IDS [39, 55]	• VMI [45] • IDS [56] • VM 上の AP や OS の動作解析 [46, 57, 58, 59, 60, 61] • VM 上の AP や OS の安全な実行 [62, 63, 64, 65]

を検知できる．また，NIGELOG は，ログファイルの改ざんを検知した場合に，バックアップからログファイルを復元する．nilfs[35] は，ファイルへの追記のみを許可することで改ざ

んを防止するファイルシステムである．アクセス制御機構 [36, 37, 38, 39, 40] は，許可しないユーザやプロセスによるログファイルへのアクセスやアクセスの種類を制限できる．これにより，ログの改ざんを防止する．暗号化ファイルシステム [41, 42] は，暗号化された状態でファイルを保存し，許可されたユーザしかファイルを閲覧できないようにする．これにより，攻撃者による改ざん対象のログエントリの判別を困難化し，改ざんを困難化する．ファイルの改ざんを防止する研究として，ファイルを分散保存する手法 [43] が提案されている．この手法は，ファイルを分割した後に署名を施し，ローカルエリアネットワーク内のディスクに分散して保存する．この手法はファイルの改ざん防止を目的としており，ログファイルの改ざん防止に利用できる．しかし，ファイルの削除には対処できない．

VMM レベルでのログ保全の関連研究として，syslog デーモンの保護 [44]，VM 上の情報取得 [45, 46, 47, 48] ，VM 上のファイルアクセスログの取得 [50]，および独自のログ保護手法 [49] がある．syslog デーモンの保護 [44] は，耐タンパ性モジュールである Trusted Platform Module と AMD 社のプロセッサに搭載されている機能である Secure Virtual Machine (SVM) により，syslog デーモンが改ざんを受けていないことを保証する．VM 上の情報取得に仮想化技術を用いる手法 [45, 46, 47, 48] は，VM 上のログ取得にも応用できる．VM の検査手法（VM Introspection，以降，VMI）[45] や CloudSec[48] では，VM 内部の状態を監査するために，メモリやレジスタなどのハードウェアレベルの情報を収集する．ReVirt[47] は，VM 上の攻撃を解析するために，状態変化を起こすような命令の実行履歴を収集している．SIM [46] は，VM 内部の動作を監視するエージェントを VM 上に挿入し，エージェントを VMM により保護する手法である．VM 上のファイルアクセスログを取得する手法 [50] は，準仮想化環境においてファイルアクセスが必ず管理用の VM を経由することに着目して，VM ごとのファイルアクセスログを取得し，保存する．独自のログ保護手法は，Linux Security Module (LSM) を利用してログ取得機構を Linux カーネル内に実現し，DigSig[68] と SecVisor[69] を利用して，ログ取得機構を保護する．これらの手法は，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．

VMM によるログ保全手法は，OS よりも攻撃の難しい VMM を利用する点で有効である．しかし，既存研究における対処は，十分でない．既存研究は，攻撃のタイミングの考慮が不十分であり，ログ保全の前段階でログを改ざんされる可能性がある．このため，保存経路のできるだけ早い段階でログを取得し，保全する必要がある．

1.2.2 攻撃難化

AP レベルでの攻撃難化の関連研究として，侵入検知システム（Intrusion Detection System, 以降，IDS）[54, 51, 52, 53] や侵入防止システム（Intrusion Prevention System, 以降，IPS）がある．IPS は，IDS と同様の侵入検知に加え，検知結果をもとに侵入を防止する．IDS には，IPS の機能を持つものもある．IDS は，侵入検知に利用する情報源により，ネットワークベース IDS（Network-based IDS，以降，NIDS）とホストベース IDS（Host-based IDS，以降，HIDS）に分けられる．NIDS は，ネットワークを流れるパケットを解析し，正常な通信と異なる通信パターンを検知することで，侵入を検知する．snort[54] は，事前に設定したルールに合致するパケットを検出した場合に，警告を表示する．NIDS による検知では，正常な通信か否かを判定する基準の設定が難しい．HIDS は，端末上で発生したイベントを解析し，侵入を検知する．HIDS による検査には，AP や OS の出力したログの検査，AP の振舞い検査，およびファイルの完全性検査がある．AP や OS は，動作履歴をログとして出力するため，サイバー攻撃を受けた際に特徴的なログを調査することで，侵入を検知できる．swatch[53] は，ログを定期的に検査することで侵入を検知する IDS である．AP の振舞い検査では，正常な AP と異なる振舞いをする AP をマルウェアとして検知する．AP や OS の出力したログの検査，および AP の振舞い検査も，NIDS と同様に，ログや振舞いが正常か否かを判定する基準の設定の難しさが問題である．ファイルの完全性検査 [51, 52] では，事前にファイルのハッシュ値を取得しておき，定期的にファイルのハッシュ値を検査する．ファイルの更新時には，ハッシュ値も更新する．これにより，端末利用者の意図しないタイミングで攻撃者によりファイルが改ざんされた場合に，ハッシュ値を検査することで，改ざんを検知できる．

OS レベルでの攻撃難化の関連研究として，OS 内で実現された IDS[39, 55] がある．LIDS[39] は，Linux カーネルにおいて実現された IDS/IPS であり，ファイルの不正な書き換えや，ネットワーク設定の変更，およびカーネルモジュールの挿入や削除といった，システムの動作の変更を検知し，防止する機能を持つ．I3FS[55] は，ファイルの完全性検査により侵入を検知するファイルシステムである．

VMM レベルでの攻撃難化の関連研究として，VMI[45]，IDS[56]，VM 上の AP や OS の動作解析 [46, 57, 58, 59, 60, 61] ，および VM 上の AP や OS を安全に実行する手法 [62, 63, 64, 65] がある．VM 内の状態を VM 外から検査する方式として，VMI[45] が研究されている．VMI は，VM の利用するレジスタやメモリの情報をもとにゲスト OS の状態を再構築することで，VM 外部で VM の状態を検査する．また，IDS を監視対象の VM とは異なる VM に隔離することで，IDS の信頼性を向上する手法 [56] が研究されている．VMM を用いて VM 上の AP

や OS の動作解析を行う研究 [46, 57, 58, 59, 60, 61] がある．IntroVirt[57] は，VM 上の AP や OS の動作を解析し，動作を再現することで，VM 上のソフトウェアにおける脆弱性を利用した攻撃を検知する．Antfarm [58] は，VM 上のプロセスの生成，削除，およびプロセス切り替えを追跡することで，VM 外から VM 内のプロセスの状態を把握する．これにより，AP や OS が攻撃を受けた状態でも，VMM によりプロセスの状態を追跡できる．Lares[59] や SIM[46] は，VM 上にエージェントを配置し，ゲスト OS の監視を行う．エージェントを配置したメモリは VMM により書き込みが制限されるため，ゲスト OS やゲスト OS 上の AP からはエージェントを攻撃できない．Ether[60] は，ページ単位のアクセス制御や VM exit を発生させる命令をゲスト OS のメモリへ埋め込むことで，ゲスト OS の動作を解析する．Aftersight[70]，PoKeR[71]，および K-Tracer[61] は，カーネルやカーネルレベルルートキットの動作を VMM により解析する．SHELLOS[72] 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である．これらは，攻撃難化のための手法を VMM により実現することや，実現した AP や OS を VMM により保護することで，攻撃難化のための手法を無効化されることを防止している．

また，VMM により，VM 上の AP や OS を安全に実行する環境を提供する手法 [62, 63, 64, 65] が研究されている．監視対象の VM とは異なる VM 上にマルウェア解析ソフトウェアを配置し，マルウェア解析ソフトウェアの発行するシステムコールを監視対象の VM にリダイレクトする手法 [62] が提案されている．これにより，マルウェア解析ソフトウェアへの攻撃を防止しながら監視対象の VM を監視できる．また，既存のマルウェア解析ソフトウェアを改変する必要がないという利点がある．Terra[63] や Qubes OS[64] は，VMM 上に信頼できる AP を動作させる VM と信頼できない AP を動作させる VM に分割することで，信頼できない AP により信頼できる AP が攻撃されることを防止する．これらの手法は，VM 間の隔離が OS 上におけるプロセス間の隔離よりも強い点を利用し，AP の安全性を高めている．HUKO[65] は，ゲスト OS 上での実行状態を 4 状態に分類し，状態遷移を VMM により監視する．これにより，信頼できないカーネルコードが動作している状態でのみ，ゲスト OS の動作を監視することで，VMM によるゲスト OS の監視で生じる性能低下を抑えながら，カーネルのコード領域やデータ領域が不正に改変されるのを防止する．

VMM による攻撃難化手法は，既存のセキュリティソフトウェアを VMM や他の VM に移植する必要や新規にセキュリティソフトウェアを作成する必要がある．既存の AP や OS による攻撃難化手法の多くは有用であるものの，手法を実現したソフトウェアを攻撃される可能性がある．対処としてソフトウェアを VMM や他の VM に移植する場合や新規に作成す

る場合は，その工数が問題となる．このため，既存の AP や OS による攻撃難化手法を実現したソフトウェアを改変することなく，攻撃から保護する必要がある．

1.3 研究の目的

仮想化技術は，将来的には，より多くの環境で利用され，VM の利用を前提としたソフトウェア構成が一般的になることが予想される．このため，VM の利用を前提とした環境において，利用者が計算機を安全に利用するためのソフトウェア実行基盤の構築が重要となる．しかし，既存研究における対処では，攻撃の証拠となり得るログの保全が十分でない．また，既存研究におけるセキュリティソフトウェアの保護は，セキュリティソフトウェアの VMM への移植や新規作成の工数が問題となる．以上のことから，本研究の目的は，VM 上で動作するソフトウェアの出力するログを保全し，VM 上で動作するセキュリティソフトウェアへの攻撃を難化することとする．

VM の利用を前提とした環境におけるログ保全では，攻撃者によるログの改ざんを防止するために，保存経路のできるだけ早い段階でログを取得し，VM レベルで隔離することを目的とする．これにより，ログが改ざんされる契機を最小化することで，ログ保全の信頼性を向上させる．

また，VM の利用を前提とした場合，一台の計算機上で多数の VM が動作する状況が想定される．このため，多数の VM から出力されるログを安全に取得できる必要がある．多数の VM が走行する環境において，それぞれの VM からログを取得する処理オーバヘッドが大きい場合には，計算機上で動作するソフトウェア全体の処理性能が低下する可能性がある．そこで，多数の VM から出力されるログを安全に取得するために，VM 上のログを低オーバヘッドで取得することを目的とする．

セキュリティソフトウェアへの攻撃難化について，セキュリティソフトウェアへの攻撃を防止する手法が研究されている．しかし，既存研究では，既存のセキュリティソフトウェアを移植する工数や新規にソフトウェアを作成する工数が問題となる．そこで，既存のセキュリティソフトウェアを改変なしに保護するために，攻撃者による攻撃対象の特定を困難にし，攻撃を回避することを目的とする．

1.4 課題

1.3 節の目的を達成するために，本論文では，以下を課題とする．

（課題 1） ログの改ざんや消失の防止

（課題 2） ログ保全処理のオーバヘッド削減

（課題 3） プロセス情報の不可視化によるプロセス特定の困難化

ログ保全の信頼性を向上するために，ログの改ざんや消失を VMM により防止するシステムを提案する．このシステムでは，ログの取得対象の OS 上で生成されたログの syslog デーモンへの転送を VMM が検知し，ログを複製し，取得対象とは異なる VM に転送する．syslog デーモンへのログの転送の検知は，VM 上で発行されるシステムコールを VMM により検知することで実現する．VM 上においてログが生成されてできるだけ早い段階でログを取得し，VM 外部へ転送することで，攻撃者によるログ改ざんの可能性を低減する．さらに，Linux において発生するカーネルログ消失の問題については，リングバッファへのログ書き込みを VMM により検知し，異なる VM に転送することで対処する．さらに，転送したログと VM 上に存在するログを定期的に自動的に比較することで，ログの改ざんを検出する．従来のログの完全性検証手法でもログの改ざんは検出できるものの，提案手法は，ログの改ざんだけでなく，改ざん箇所も検出する．これにより，攻撃者の意図や目的の分析や推論を容易にする．VM 上の AP や OS を一切改変することなく，VMM の改変のみにより VM 上のログの取得を実現することで，攻撃者に機構自体を攻撃される可能性を低減し，かつ多種の OS が動作する VM へ提案手法を適用可能にする．この手法について，設計，実現，および評価を行った結果を報告する．

ログ保全の信頼性向上は重要である一方で，ログ保全における性能低下が問題となる場合がある．ログの保存処理は，AP において，性能低下の主要な原因の一つと考えられている [73]．このため，実際の AP の運用環境において，保全のためにログの VM 外部への転送を実現するには，転送のオーバヘッドの削減が要求される．そこで，ログ保全の信頼性を向上しつつ，ログ保全のオーバヘッドを抑制する手法として，ライブラリの置き換えによる VM 外部への低オーバヘッドなログ転送手法を提案する．この手法では，ログの転送機能を VM 上に挿入し，VM 上の AP によるログ出力を VMM により通知することで，低オーバヘッドで VM 外部にログを転送する手法を提案する．ログの改ざんや消失を防止するシステムの性能低下の主な原因は，AP からのログ転送を VMM により検知するための AP の動作監視である．このため，VM 上にログ転送機能を挿入し，ログ転送の契機を VMM に明示的に通知することで，AP の動作監視を不要にするログ転送機能を実現する．ログ転送機能の VM 上への挿入では，新規にプログラムを作成するのではなく，既存のライブラリを改変したライブラリに置き換えることで実現する．これにより，追加のプログラムのインストールが不要

になり，導入が容易になる．ただし，VM 上に挿入したログ転送機能は，攻撃者から改変され，無効化される可能性がある．このため，挿入したログ転送機能を改変されないように，VMM により保護する必要がある．本研究では，置き換えたライブラリへの攻撃を防止する手法も提案する．この手法について，AP を用いて性能を評価した結果を報告する．また，複数の VM を動作させた場合における性能変化について評価した結果を報告する．

ただし，ログの改ざんや消失を VMM により防止するシステムよりもログ保全の信頼性は低下する．このため，ログの信頼性が要求される環境では，ログの改ざんや消失を VMM により防止するシステムを利用し，高負荷な環境では，ライブラリの置き換えによる VM 外部への低オーバヘッドなログ転送手法を利用することが望ましい．

セキュリティソフトウェアの改変なしの保護を実現するために，プロセス情報の不可視化によりプロセスの特定を困難にする攻撃回避手法を提案する．この手法では，攻撃者から攻撃対象プロセスの特定を困難にするために，プロセス情報へのアクセス制御とプロセス情報の置換を提案する．プロセス情報へのアクセス制御により，許可されていない領域からのプロセス情報の読み込みを禁止する．プロセス情報の置換では，プロセス情報の置換対象のプロセスの走行中は，本来のプロセス情報を利用させ，それ以外では偽のプロセス情報へ置換する．これらの機能により，プロセス情報をもとにした攻撃対象プロセスの特定を困難にする．また，この機能を VMM の改変により実現する．これにより，OS レベルで動作するマルウェアからの攻撃にも耐性のある実行環境の提供を目指す．本手法について，その実現方式と動作を確認した結果を報告する．

1.5 論文の構成

第 2 章では，ログ保全の信頼性を向上するために，ログの改ざんや消失を防止する手法について述べる．

第 3 章では，VM の利用を前提とした環境におけるログ保全の処理性能向上のために，ライブラリの置き換えにより VM 外部へ低オーバヘッドでログを転送する方式について述べる．

第 4 章では，プロセス情報の不可視化により，攻撃者から攻撃対象プロセスの特定を困難にする攻撃回避手法について述べる．

第 5 章では，本論文の結論と今後の課題について述べる．

第 2 章

ログの改ざんや消失を防止する手法

2.1　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，ログの改ざんや消失が起こる可能性がある．この問題に対処するため，VMM を用いてログの改ざんや消失を防止するシステムを提案する．提案システムは，対象の VM で動作している AP や 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．ログの生成後のできるだけ早い段階でログを取得することで，改ざんの契機を低減する．また，VMM の取得したログを取得対象の VM とは別の VM に保存することで，ログの安全性を確保する．さらに，ログを隔離し多重化することで，ログの改ざん箇所を検出できる．

2.2　既存のログ保存方式

2.2.1　syslog

Linux 2.6.26 上で調査した syslog の処理を図 2.1 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．

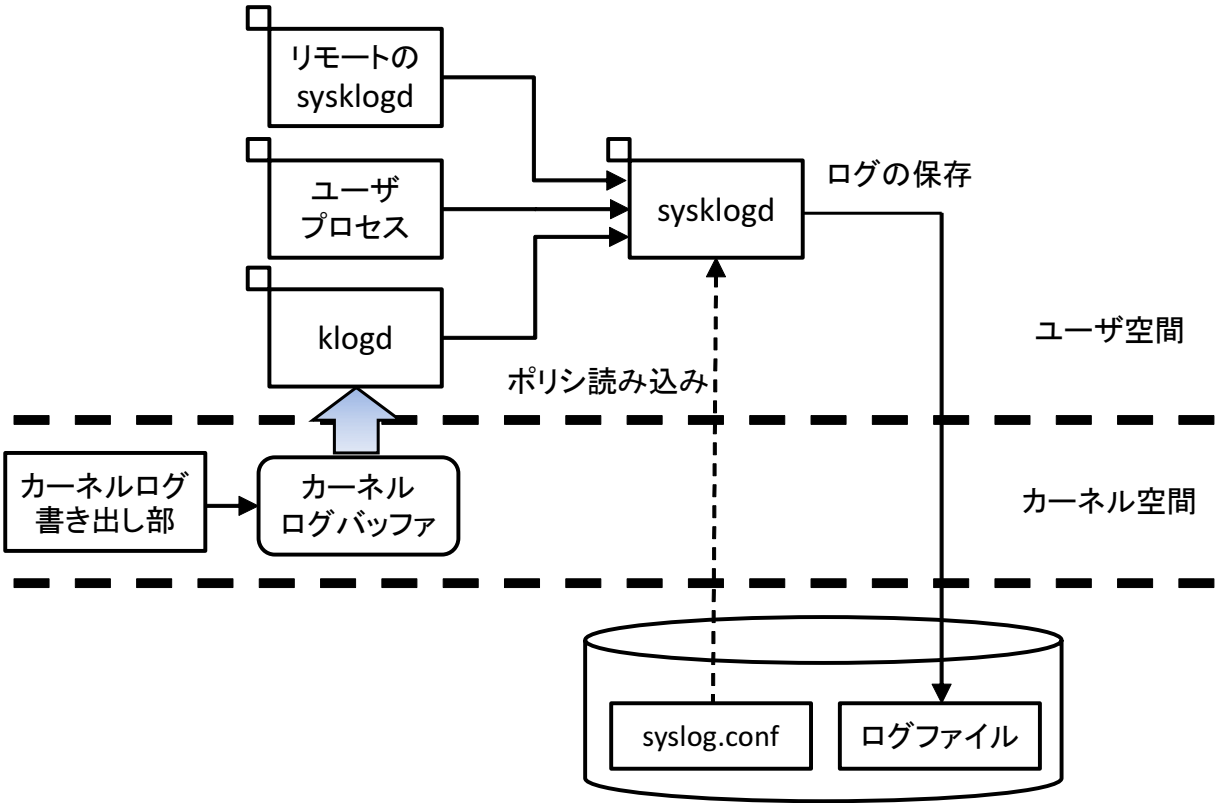

図　2.1　syslog の処理

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した syslog デーモン [26, 27] 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．

以降，ログや syslog を保護する関連研究について述べる．

2.2.2　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 [43] 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 [34] 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 [50] 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 [30]．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．

2.2.3　syslog の保護

文献 [44] 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．

2.2.4　独自のログの保護手法

文献 [49] 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[68] 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[69] 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．

2.2.5　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 [50, 44] 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 [50, 49] 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．

2.3　仮想計算機モニタによりログの改ざんや消失を防止するシステム

2.3.1　システムへの要求

2.2.5 項で述べた問題を解決するシステムを提案する．(問題 1) への対処には，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) への対処には，ログの保護機構自体を AP やカーネル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，カーネルログを消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の実現

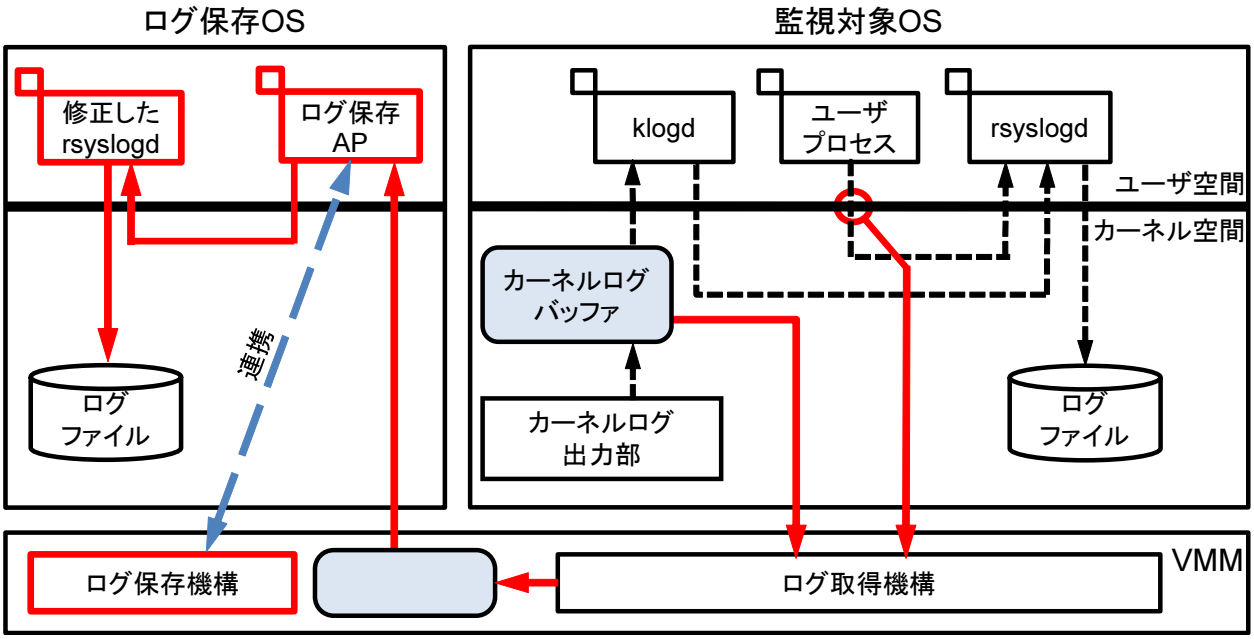

図　2.2　提案システムの全体像

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ものの，重要な課題である．

2.3.2　提案システムの構成

提案システムの全体像を図 2.2 に示す．提案システムは，監視対象 OS を VM 上で動作させることを想定する．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，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，これら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．

2.3.3　ログ取得機構

ユーザログ取得機能

本機能は，図 2.2 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には，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，提案システムは，文献 [60] 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．図 2.3 にユーザログ取得におけるシステムコールフック処理の流れを示す．この手法は，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とで実現する．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）[74]．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．

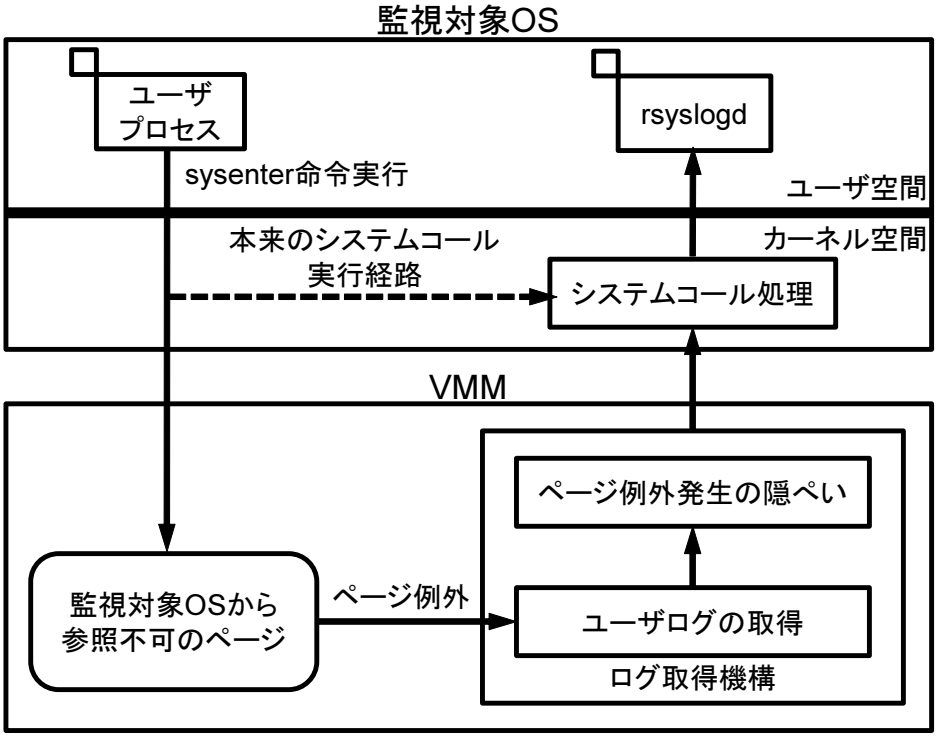

図 2.3 ユーザログ取得におけるシステムコールフック処理の流れ

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．

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

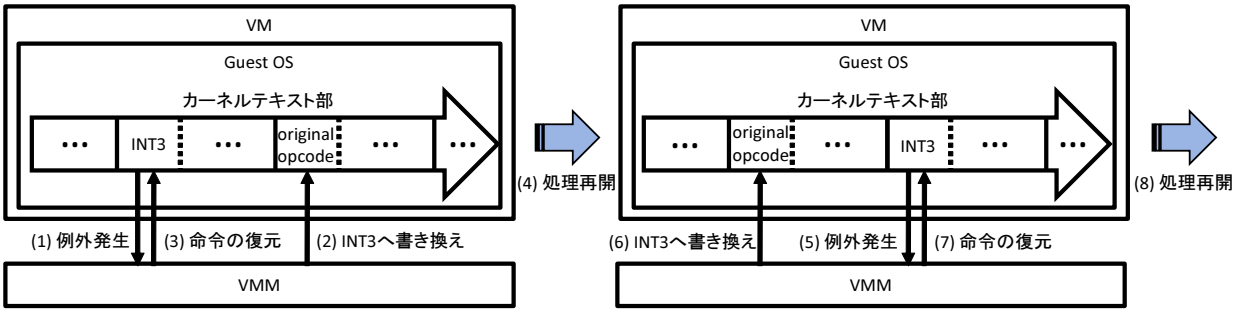

図　2.4　ゲスト OS へのブレークポイントの設定

性は低下しない．

以下，図 2.4 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．

2.3.4　ログ保存機構

ログ保存機構の要件

本項は，2.2.5 項で述べた問題へ対処するため，自動的にログをファイルへ書き出すログ保存機構を提案する．ログ保存機構により 2.2.5 項の問題へ対処するには，以下の要件がある．

(1) ログを漏れなくファイルへ書き出すこと

(2) ログ保存機構による性能低下を小さくすること

ログを確実に保存するために要件 (1) が必要である．また，ログ保存機構の実現により発生するオーバヘッドは，監視対象 OS の性能へ影響する．このため，要件 (2) が必要である．

ログ保存機構の構造

ログ取得機構は，監視対象 OS のメモリ領域にあるログを Xen 内部へ確保したバッファへコピーする．ログ保存機構は，監視対象 OS のログを取得したことをログ保存 OS で動作するログ保存 AP へ通知する．ログ保存 AP は通知を受けると，VMM に対してログのコピーを要求する．要求を受けた VMM は，ログ保存 AP へログをコピーし，ログ保存 AP は syslog デーモンを経由してログをファイルへ書き出す．

ここで，ログ保存 AP からログを受け取るために，ログ保存 OS で既に起動している syslog デーモンとは別の syslog デーモンを用意する．これは，単体の syslog デーモンで監視対象 OS とログ保存 OS の両方のログを保存した場合に，提案手法で取得したログとログ保存 OS のログが混在することを防ぐためである．新たに用意する syslog デーモンの詳細は後述する．

VMM とログ保存 AP の通信

VMM とログ保存 AP の通信には，イベントチャネルを用いる．イベントチャネルは，VMM で検知した IRQ の VM への通知，VMM が生成した擬似 IRQ の VM への通知，および VM 間通信を実現するための機構である．VMM とログ保存 AP の通信には，VM 間通信のためのイベントチャネルを利用し，VMM から VM へイベントを配送することで通知を実現した．

VMM とログ保存 AP の通信は，ログ取得機構による監視対象 OS のログの取得をログ保存 AP へ通知するために行う．この際，ログ保存 AP へ通知するタイミングとして，ログ取得ごとに通知する方式と一定量のログを蓄積してから通知する方式がある．提案するログ保存機構は，ログ取得ごとに通知する方式を用いる．

一定量のログを蓄積してから通知する方式は，ログ取得ごとに通知する方式よりもログのコピー回数が少なく，性能低下が小さい．しかし，一定量のログを蓄積する方式は，不意な電源断などによりメモリ上のログが消失した場合，消失するログの量が大きくなる．要件 (1) を考慮すると，より確実にログを保存するためには，ログ取得ごとに通知する方式が優れている．また，一定量のログを蓄積する場合，VMM の確保するメモリ領域が大きくなる問題がある．VMM が必要以上にメモリを占有すると VM へ割り当て可能な領域が小さくなる．

以上のことから，提案するログ保存方式は，ログ取得ごとに通知する方式を用いる．

VMM からログ保存 AP へのログのコピー

VMM が監視対象 OS から取得したログをログ保存 AP へコピーする際，以下の 2 点を考慮する必要がある．

(1) イベントの配送契機

(2) ログのコピーにおけるオーバヘッドの削減

イベントは，はじめに配送対象の VM のキューに格納され，配送対象の VM がスケージューリングされると配送される．このため，即座には配信されない．ログ取得機構によるログ取得の通知にはイベントを用いるため，イベント配送の遅延を考慮する必要がある．

イベントの配送における遅延へ対処するため，ログ保存機構はログ取得機構が取得したログをメモリ上にバッファリングする必要がある．ただし，ログ保存機構は，一定量のログを蓄積してログ保存 AP へ通知する方式とは異なり，ログ取得機構が監視対象 OS からログを取得した直後にログ保存 OS へイベントを配送する．このため，イベントがログ保存 OS へ到達すると，ログ保存 AP が VMM に対してログのコピーを要求する．

また，要求 (2) を満たすために，ログのコピーにおけるオーバヘッドを削減する必要がある．ログ保存 OS にイベントが到達するまでの間，取得したログは VMM 内のバッファに蓄積される．ログ保存 OS からハイパーコールが発行されると，VMM は蓄積したログをログ保存 OS へコピーする．

ログ保存機構は，ログ取得機構が短期間に大量のログを取得した場合にのみバッファリングする．ログ保存機構は，監視対象 OS 上のログの送信をログ取得機構が検知すると，イベントをログ保存 AP に送信する．Xen のイベントは非同期でゲスト OS に通知される．ログ取得機構は取得したログを VMM 内に蓄積し，ログ保存 OS がハイパーコールを発行すると，ログ保存 OS にコピーする．この場合，ログ取得機構がログを取得するたびにログ保存 OS

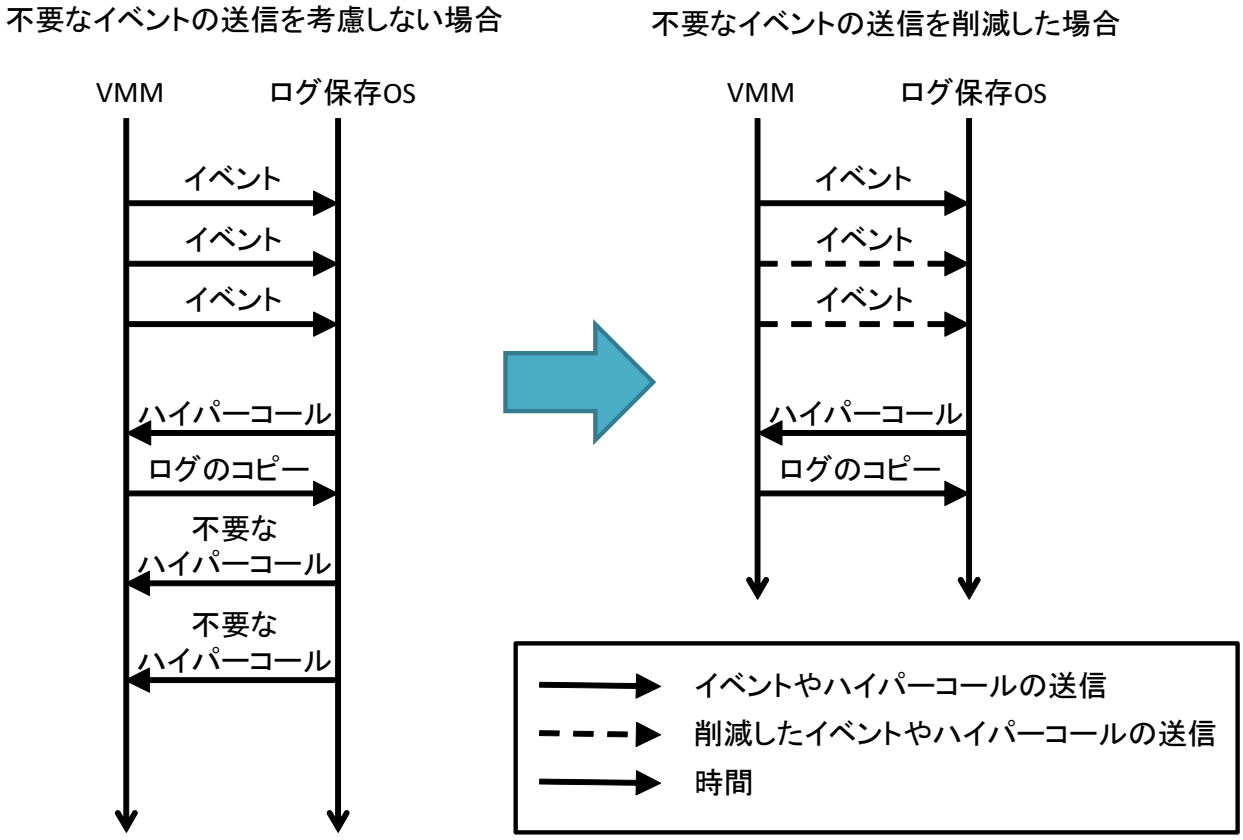

図　2.5　不要なイベント送信の削減

へイベントを送信すると，ログ保存 OS は，イベントを受信するたびにハイパーコールを発行する．しかし，ログ保存機構は，ハイパーコールを受信すると，蓄積したログを一度に全てログ保存 OS にコピーする．このため，ログ保存 OS は，不要なハイパーコールを発行することになる．ハイパーコールが発行されると，ゲスト OS と VMM の間のモード遷移が起こるため，不要なハイパーコールの発行は，性能低下に繋がる．

この問題への対処として，ログ保存機構は，不要なイベントの送信を抑制する．ログ取得機構は，監視対象 OS から取得したログが VMM 内のバッファに蓄積されている場合，イベントを送信済みか否かを判断し，イベントが未送信の場合のみ，イベントを送信する．これにより，VMM は，一度のイベント通知と一度のハイパーコール発行で VMM 内のすべてのログをログ保存 OS にコピーできるため，不要なハイパーコールを発行しない．図 2.5 に不要なイベントを削減した場合のイベント通知とハイパーコール発行の様子を示す．

ログのコピーによるオーバヘッドを削減するためには，ログの総コピー回数を削減する必要がある．しかし，現在のログ保存機構では総コピー回数の削減は実現されておらず，今後

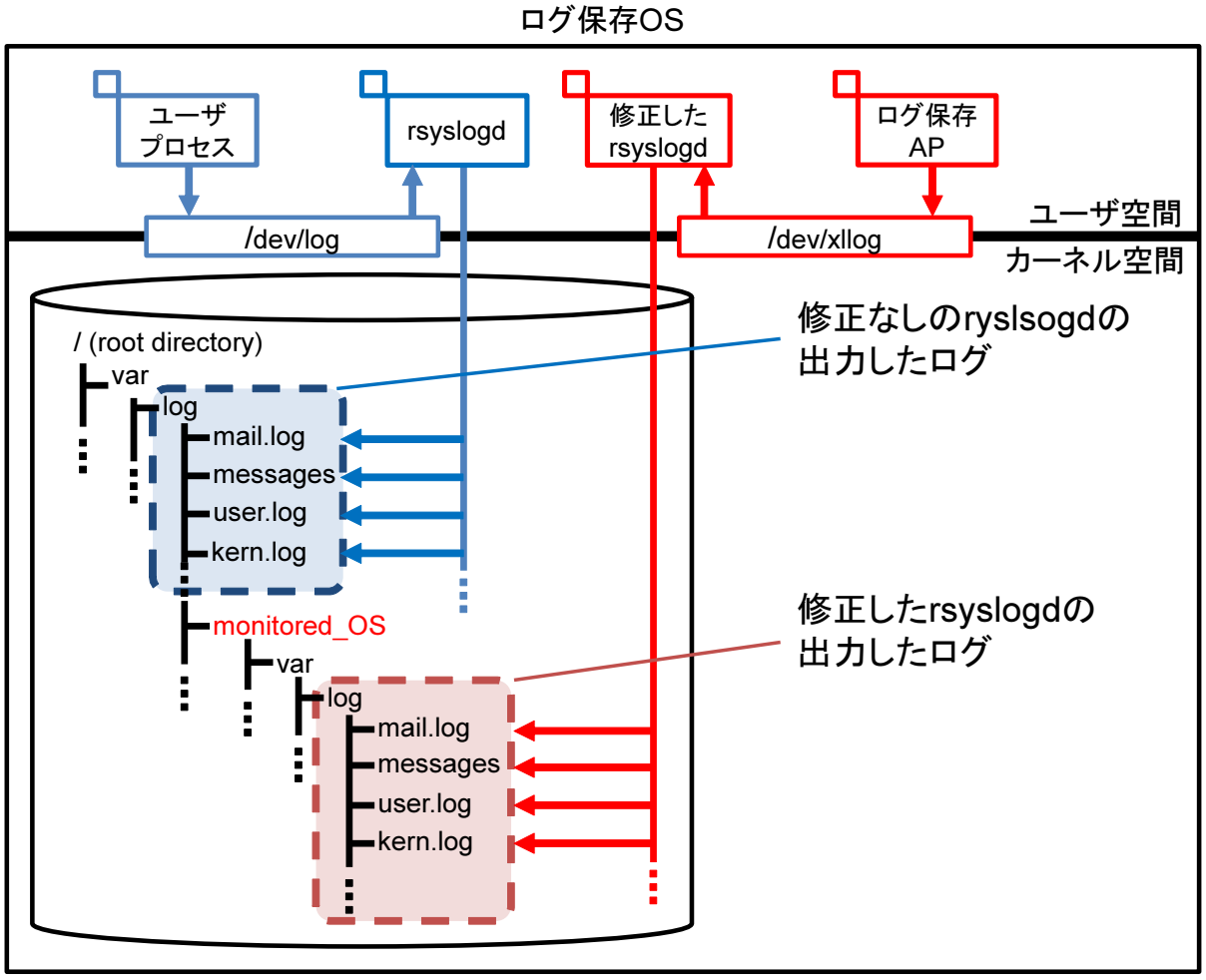

図　2.6　修正した rsyslogd を用いたログの書き出し

の課題である．

ログ保存 OS におけるログの保存処理

図 2.6 に修正した syslog デーモンを経由してログ保存 AP がログを書き出す処理を示す．ログ保存 AP は，ログをファイルへ書き出す際に syslog デーモンを用いる．ここで，ログ保存 AP からログを受け取る syslog デーモンとログ保存 AP 以外の AP からログを受け取る syslog デーモンを共存させるため，syslog デーモンを修正した．syslog デーモンは/dev/log ソケットファイルからログを取得する一方で，修正した syslog デーモンは/dev/xllog を経由してログを取得する．ログ保存 AP は，ログ保存機構からログを取得し，/dev/xllog ソケットファイルを経由して修正した syslog デーモンへログを送る．

また，ログの比較による改ざん検知を容易にするため，修正した syslog デーモンは，監視対象 OS 上の syslog デーモンと同じポリシファイルを利用する．ただし，ログ保存 OS 上のポリシファイルを利用するとログ保存 AP とそれ以外の AP が出力したログが混在してしまう．この対処として，修正した syslog デーモンによるログの出力先のディレクトリを異なる場所へ変更した．たとえば，あるログを/var/log/messages へ出力するようにポリシファイルが記述されていた場合，/var/log/monitored_OS/var/log/messages へ出力するようにポリシファイルを修正する．

これらの対処により，提案手法で取得したログと監視対象 OS 上のログを容易に比較可能となる．

2.4　ログの改ざんや消失を検知する機能

2.4.1　基本方式

監視対象 OS 上のログファイルとログ保存 OS 上のログファイルを比較する．2.3.4 項で述べたように，ログ取得機構が監視対象 OS から取得したログは，ログ保存 OS 上へ保存されている．ログ保存の際，監視対象 OS 上のログファイルが格納されるディレクトリのディレクトリ構造を監視対象 OS 上で再現する．このため，ログファイルに対する改ざんやログの消失が発生した場合，監視対象 OS とログ保存 OS でログファイルを比較することで，ログの改ざんや消失を検知できる．

2.4.2　改ざんや消失を検知するために比較する項目

比較によりログの改ざんを検知するため，以下の項目を比較する必要がある．

(1) ホスト名

(2) 時刻

(3) ユーザ名

(4) ログメッセージ

ホスト名は，ログがどの OS で出力されたものかを判別するために必要である．ログの出力元の OS を判別しなければ，ログの比較はできない．また，時刻，ユーザ名およびログメッ

セージは，ログの出力された環境を知るために必要な情報である．攻撃者は，コマンドの実行時刻や実行したユーザを隠ぺいするためにログを改ざんする．また，あるログが改ざんされた場合，ヒステリシス署名などによりログの改ざんを検知できる．しかし，削除の内容や目的は判断できない．ログメッセージは，攻撃者の目的を知るために必要となる．

2.4.3　ログの比較

ログファイルの取得

監視対象 OS 上のログファイルとログ保存 OS 上のログファイルを比較し，改ざんを検知する．監視対象 OS 上のログファイルを取得するため，監視対象 OS の動作する VM のディスクイメージをログ保存 OS からマウントした．この方法により，監視対象 OS を停止する必要なしにログファイルの比較が可能である．

比較

ログファイルの比較には diff を利用した．2.3.4 項で述べたように，ログ保存 OS 上の修正を加えた syslog デーモンは監視対象 OS 上の syslog デーモンと同じポリシファイルを利用している．このため，diff で OS 間のログファイルを比較することで容易に改ざん箇所や改ざん内容を検知できる．

図 2.7 に監視対象 OS とログ保存 OS の間でログファイルを比較する様子を示す．ログ保存 OS における /var/log/monitored_OS/var/log ディレクトリは，監視対象 OS における /var/log ディレクトリに対応する．図 2.7 に示すように，提案したログの改ざん検知手法では，OS 間でログファイルを比較するだけで監視対象 OS におけるログの改ざんを検知できる．

2.4.4　ログの整形

syslog デーモンの出力するログにはホスト名が含まれる．しかし，修正した syslog デーモンの出力するログには，監視対象 OS のホスト名ではなく，ログ保存 OS のホスト名が記録される．このため，ログを比較する前に，ログ保存機構の保存したログファイルに含まれるホスト名を監視対象 OS のホスト名へ置き換えた．

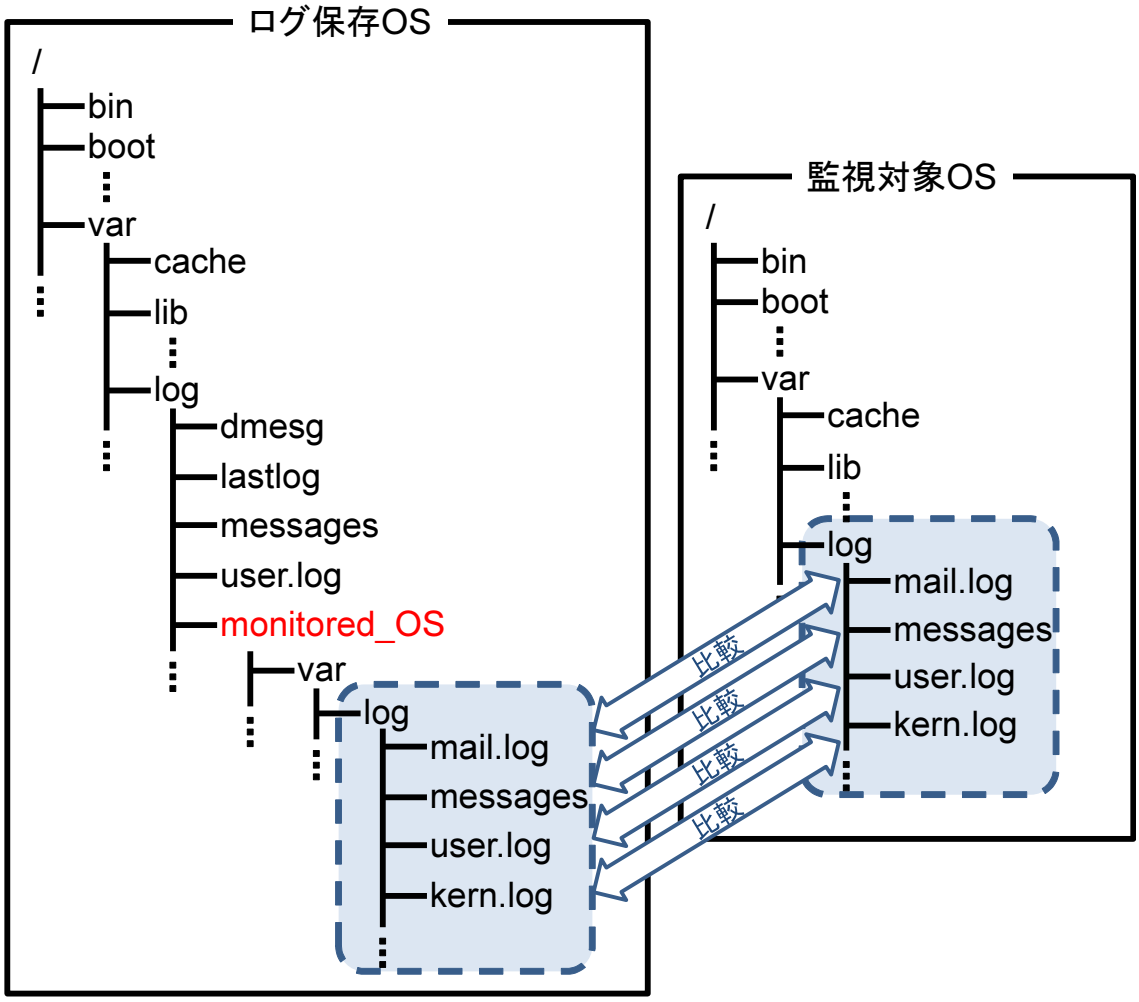

図　2.7　監視対象 OS とログ保存 OS の間でのログの比較

2.4.5　改ざん検知の効果

ログの改ざんの検知では，ログ保存機構で保存したログファイルと監視対象 OS 上のログファイルを比較する．ログが改ざんされていた場合，ログファイルの比較により，改ざんの有無を検知できる．また，ログファイルが改ざんされていた場合，改ざん箇所を特定できる．

指定したファイルやディレクトリのシグネチャを比較することでファイルの変更，置換，および削除を検知できる Tripwire[75] が提案されている．しかし，変更箇所と変更内容の検知は不可能である．文献 [75] は，ファイルを単純に比較する方法が前述の点で優れている．しかし，ファイルのコピーを保持する場合，必要なディスク領域とファイルのコピー時間において Tripwire が優れると述べている．提案するログ保存機構は，syslog を利用したログの

みを検知し，新たに出力されたログのみを取得し，コピーを作成する．このため，ファイルの更新ごとにコピーを作成する必要はない．また，新たに出力されたログのみを検知し，コピーするため，コピー時間は小さい．このことから，提案方式には，文献 [75] で指摘されている問題はない．

ただし，提案システムのログ取得機構の取得対象は，syslog を利用したテキスト形式のログである．このため，syslog を利用しないログは，ログファイルの比較による改ざんや消失の検知の対象でない．たとえば，Apache Web サーバは，syslog を利用せず，独自にログを出力する．利用者が Web サーバを対象としたログの改ざんや消失防止，および検知を要求する場合，Apache 特有の機能を利用しないのであれば，Apache の代替として syslog を利用する Web サーバを用いることで対処できる．

2.5　評価

2.5.1　評価の目的と評価環境

監視対象 OS で発生するログの消失の防止，改ざんや消失の検知，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．

評価環境を表 2.1 に示す．VMM として Xen [16] 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．この要求により，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．

2.5.2　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．

表　2.1　評価環境

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
メモリ	Physical	2,048 MB
	Domain 0	1,024 MB
	HVM ドメイン	1,024 MB

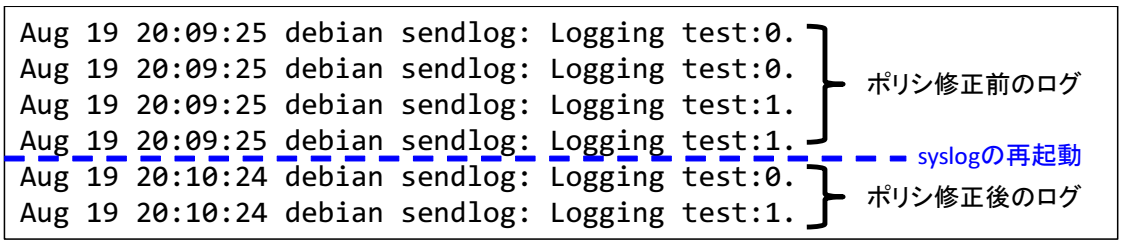

図　2.8　監視対象 OS 上のユーザログ

2.5.3　ログの消失の防止

ユーザログの消失の防止

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 2.8 と図 2.9 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 2.8 には，図 2.9 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 2.10 にログ書き出しポリシ変更の様子を示す．図 2.10 では，user と mail ファシリティを持つログを/var/log/user_and_mail.log に書き出しているポリシを変更し，user ファシリティを持つログのみを書き出すように，ポリシを変更している．図 2.10 で示すように syslog の振舞いを変更させた前後で双方のログを比較した．

図 2.8 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

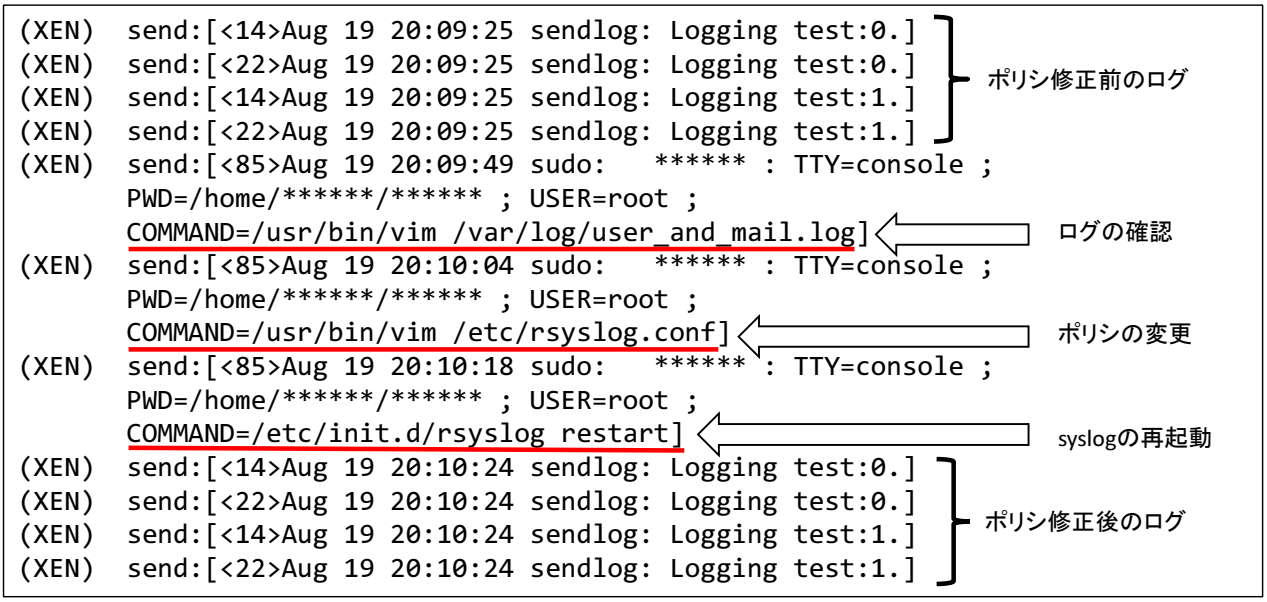

図　2.9　提案システムで取得したユーザログ

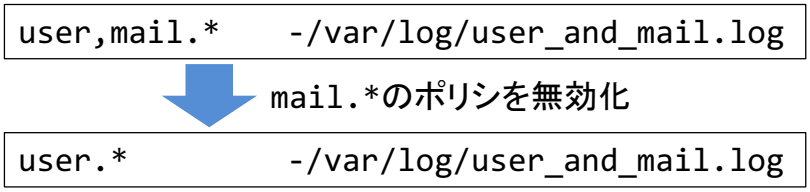

図　2.10　ログ書き出しポリシの変更

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 2.9 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．

カーネルログの消失の防止

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

```
Sep  1 06:38:48 debian kernel: [   11.789840] EXT3-fs: mounted filesystem with ordered data mode.
Sep  1 06:38:48 debian kernel: [   13.212910] eth1: link up, 100Mbps, full-duplex, lpa 0x05E1
Sep  1 06:39:08 debian kernel: st world.
Sep  1 06:39:08 debian kernel: [   37.586658] Hello   52nd world.
Sep  1 06:39:08 debian kernel: [   37.587389] Hello   53rd world.
                                              ⋮
Sep  1 06:39:09 debian kernel: [   43.244123] Hello 3499th world.
Sep  1 06:39:09 debian kernel: [   43.244889] Hello 3500th world.
```

図 2.11 監視対象 OS 上のカーネルログ

```
(XEN) KERNLOG:<6>[   11.789840] EXT3-fs: mounted filesystem with ordered data mode.
(XEN) KERNLOG:<6>[   13.212910] eth1: link up, 100Mbps, full-duplex, lpa 0x05E1
(XEN) KERNLOG:<4>[   37.536696] Hello    1st world.
(XEN) KERNLOG:<4>[   37.537447] Hello    2nd world.
                                            ⋮
(XEN) KERNLOG:<4>[   37.585928] Hello   51st world.
(XEN) KERNLOG:<4>[   37.586658] Hello   52nd world.
(XEN) KERNLOG:<4>[   37.587389] Hello   53rd world.
                                            ⋮
(XEN) KERNLOG:<4>[   43.244123] Hello 3499th world.
(XEN) KERNLOG:<4>[   43.244889] Hello 3500th world.
```

図 2.12 提案システムで取得したカーネルログ

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 2.11 に，提案システムで取得したカーネルログを図 2.12 に示す．図 2.11 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消失したためである．一方，図 2.12 では，図 2.11 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 では消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．

```
5,9d4
< Nov 27 22:56:22 debian sudo:   **** : TTY=pts/0 ; PWD=/home/**** ; USER=root ; COMMAND=/bin/login thief
< Nov 27 22:56:24 debian login[2626]: pam_unix(login:session): session opened for user thief by ****(uid=0)
< Nov 27 22:56:51 debian sudo:    thief : TTY=pts/0 ; PWD=/home/thief/vanish ; USER=root ; COMMAND=/bin/bash
< Nov 27 22:58:08 debian sudo:    thief : TTY=pts/0 ; PWD=/home/thief/vanish ; USER=root ; COMMAND=./vanish
thief thief-host thief-adr
```

図 2.13 監視対象 OS とログ保存 OS のログファイルに diff を適用した際の出力結果

2.5.4 ログファイルの改ざん検知

監視対象 OS においてログファイルを改ざんされる状況を想定し，提案方式で改ざんを検知できるか評価した．

実験では，ログを改ざんするソフトウェアとして Vanish[76] を用いた．Vanish は，教育目的で開発されたソフトウェアであり，指定されたユーザ名，ホスト名および IP アドレスを含むログを削除する．実験では，ログの改ざんを検知するため，監視対象 OS 上の auth.log を Vanish を用いて改ざんした．ここで，Vanish により改ざんするファイルには特権ユーザ以外は書き込めないファイルが含まれているため，Vanish の利用には特権が必要となる．ユーザが sudo コマンドを用いて実行したコマンドは，auth.log に記録される．実験では，不正な操作をするユーザ `thief` として監視対象 OS にログインし，Vanish によりログを改ざんした．

図 2.13 は，実験後に監視対象 OS とログ保存 OS の間で auth.log を diff により比較した結果を示している．図 2.13 に示したログは，Vanish の実行により削除されているため，監視対象 OS の auth.log には存在しない．

実験結果により，ログ保存機構により書き出されたログは監視対象 OS 上でのログファイルへの改ざんの影響を受けないことを示した．また，監視対象 OS のログファイルにおける改ざん箇所を検出できることを示した．

2.5.5 攻撃への耐性

ログの正当性を保証するためには，ログが出力され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 [44]．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 [30] 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．さらに，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 [43, 77] 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[68] 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) カーネルログバッファに蓄積され収集されるまでの間

(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

表 2.2 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）

	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%)

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[69] 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，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．

2.5.6　提案システムの導入によるオーバヘッドの測定

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

表 2.3 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）

	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%)

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 2.2 と表 2.3 に示す．

表 2.2 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 2μs 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 3μs 必要だと推察できる．また，write のオーバヘッドも同様に約 47μs 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 2.3 における syslog のオーバヘッドと表 2.2 における write のオーバヘッドの差が 5μs 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 54μs 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 2.3 から，printk 関数のオーバヘッドは約 57μs 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．

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 2.4 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

表 2.4 LMbench による測定結果（単位：μs）

	Process			0KB File		10KB File	
	Null Call	Fork	Exec	Crete	Delete	Create	Delete
Xen	0.12	698	1706	5.50	3.44	22.3	8.08
提案システム	2.16	753	1725	11.3	6.67	40.0	11.8
オーバヘッド	2.04 (1700%)	55 (8%)	19 (2%)	5.8 (105%)	3.23 (100%)	17.7 (79%)	3.0 (37%)

表 2.5 Web サーバのスループットの測定に用いたソフトウェア

	サーバ計算機			クライアント計算機	
CPU	Core 2 Duo (2.40 GHz)			Pentium 4 (3.0 GHz)	
メモリ	監視対象 OS	1 GB	2 GB	1 GB	
	ログ保存 OS	1 GB			
NIC	100 Mbps			100 Mbps	
ソフトウェア	thttpd 2.25b			ApacheBench 2.3	
				リクエスト数	100
				接続の並列度	1

をしており，表 2.2 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，20〜50μs 程度のオーバヘッドが発生している．また，0KB および 10KB のファイル生成では，5〜20μs 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．

提案システムの導入による AP の性能への影響

AP の性能への影響を評価するため，監視対象 OS で Web サーバを動作させ，スループットを測定した．表 2.5 に測定環境を示す．測定では，監視対象 OS 上で Web サーバとして thttpd 2.25b を動作させ，クライアント計算機から ApacheBench 2.3 を用いて監視対象 OS 上の Web サーバに対してファイル要求を発行した．thttpd は，ログの出力に標準で syslog を利用する．

測定では，ApacheBench により 1KB と 100KB のファイルに対して 100 回ずつ要求を発行し，1 秒あたりの要求数をスループットを測定した．ApacheBench による接続の並列度は 1

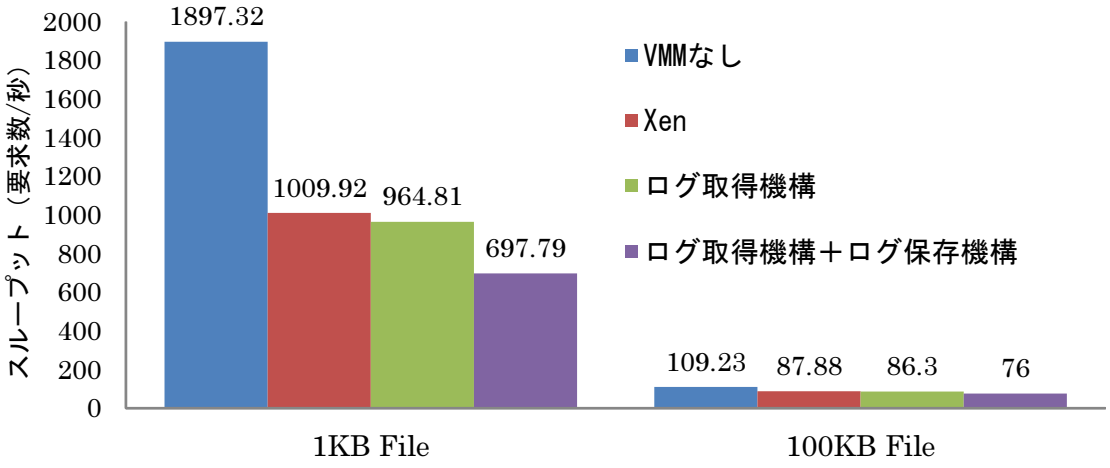

図　2.14　各環境における Web サーバのスループットの測定結果（単位：要求数/秒）

表　2.6　各環境における Web サーバのスループットの比較

	1 KB File				100 KB File			
	(1)	(2)	(3)	(4)	(1)	(2)	(3)	(4)
(1)　VMM なし	1	1.88	1.97	2.72	1	1.24	1.27	1.44
(2)　Xen	0.53	1	1.05	1.45	0.80	1	1.02	1.16
(3)　ログ取得機構	0.51	0.96	1	1.38	0.79	0.98	1	1.14
(4)　ログ取得機構＋ログ保存機構	0.37	0.69	0.72	1	0.70	0.86	0.88	1

である．

図 2.14 に測定結果を示す．また，表 2.6 は，(1)VMM なしの場合，(2)Xen を利用した場合，(3) ログ取得機構を適用した場合，および (4) ログ取得機構とログ保存機構を適用した場合における Web サーバのスループットについて，各環境の性能比を示している．ログ取得機構とログ保存機構の両方を利用した場合，1 KB のファイル要求では Xen の約 69%，100 KB のファイル要求では約 86%であった．ここで，ファイルサイズが 100KB の時は，1KB の時と比べて性能低下の割合が小さい．これは，1 回のログ出力にかかる時間がデータサイズに関わらず一定なのに対し，データの転送にかかる時間はデータサイズに依存するためだと推察できる．このため，データサイズが大きくなると，1 回の処理に占めるデータ転送時間の割合が増加し，ログ出力時のオーバヘッドによる処理時間への影響は小さくなる．

表 2.7 は，Web サーバへのファイル要求間隔による応答速度の比較であり，連続した要求と 1 回の要求で応答速度を比較している．連続試行回数は，Web サーバへ連続してファイルを要求した回数であり，連続して 100 回要求した場合は，ファイルサイズ 1KB のときは 0.2

表 2.7 Web サーバへの要求間隔による応答速度の比較

ファイルサイズ	連続試行回数	Xen (ms)	ログ取得機構＋ログ保存機構 (ms)
1KB	100	0.99	1.43
	1	1.15	1.45
100KB	100	11.38	13.16
	1	11.71	12.75

秒以内，100KB の時は 1.4 秒以内に全ての処理が完了している．表 2.7 より，提案手法を導入した場合，1 回のみファイルを要求した際は，連続して要求した際よりも応答速度が同等か高速になっているといえる．ファイルサイズ 1KB の場合は 0.02 ミリ秒応答速度が低下しているが，100KB の場合は 0.41 ミリ秒高速である．これは，提案手法によるログ保存の際，ログ保存 OS がスケジューリングされるまで実際はディスクアクセスが生じないためだと推察できる．連続して要求した際は，ディスクアクセスにより平均応答速度が低下する．この結果より，ディスクアクセスの削減は，提案手法の導入による応答時間の低下を抑えられるため有効であり，ログを VMM 内でバッファリングする方式が有効であるといえる．

2.6　関連研究

2.6.1　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 [45] 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．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 [60] 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 [72] 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 [47] 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．

2.6.2　VMM の安全性

VMM への攻撃を防止し，VMM の完全性を保証する手法が研究されている [78, 79]．HyperSafe [78] はベアメタル型の VMM の完全性を保証し，HyperLock [79] はホスト型の VMM の完全性を保証する．特に，本研究においては，文献 [78] の手法を併用することで，より信頼性の高いログ取得機構を実現できる．

2.7　まとめ

本章では，監視対象 OS のログの改ざんや消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．ユーザログ取得機構では，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 のログ送信要求終了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消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，ログの改ざんや消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消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消失する問題について，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，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 2–5μs 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，測定した処理のオーバヘッドは，約 5–50μs であり，その影響は小さいことを示した．

第 3 章

仮想計算機外部への低オーバヘッドなログ転送方式

3.1　概要

カーネルレベルで動作するマルウェアによるログの改ざんは，防止が困難であり，ログの改ざんにより，攻撃の検知の遅れや攻撃内容の把握が困難になる問題がある．カーネルレベルでのログの改ざんに対処できる手法として，VMM を用いる手法が提案されているものの，性能低下が問題となっている．

本章では，VM 上のライブラリの置き換えにより，VM 上のログを VM 外部へ低オーバヘッドで転送する手法について述べる．この手法では，置き換えたライブラリがログの転送依頼を VMM に送信し，VMM が VM 上のログを取得する．VMM は，取得したログを異なる VM に転送し，保存することで，取得したログの安全性を確保する．置き換えたライブラリは，VM に侵入した攻撃者により改変される可能性もあるため，VMM により保護する必要がある．このため，本章では，置き換えたライブラリの保護手法についても考察する．

3.2　VM 上のライブラリの修正による安全で高速なログ転送手法

3.2.1　対象と想定する環境

本研究は，カーネル，ロギングデーモン，およびログファイルへの攻撃によるログの改ざんの防止を目的とする．攻撃者がすべての AP によるログ出力を監視し，攻撃者に不都合な

ログをすべて改ざんするのは難しい．このため，攻撃者は，カーネル空間やロギングデーモンのように，すべての AP からのログが経由する箇所を攻撃する．このため，カーネル空間やロギングデーモンにおけるログの改ざんを防止することが重要である．

本研究では，ユーザレベルやカーネルレベルで動作できるマルウェアの利用を想定する．また，VMM への攻撃は困難であると想定する．Xen への攻撃について報告 [24] があるものの，攻撃可能な環境は限定されている．また，VMM への攻撃は，完全性検証によって検知可能である [25]．このため，VMM への攻撃は十分に難しいものとする．

3.2.2　目的

本研究の目的を以下に示す．

（目的 1） ログの改ざんが困難で高速なログの転送手法の提案

（目的 2） 多種の OS へ容易に適用可能なログの転送手法の提案

このためには，既存のログ転送手法における以下の問題の解決が必要である．

(1) プロセス間通信によるログの転送は，カーネルレベルの攻撃により妨害され，ログを改ざんされる可能性がある．

(2) VM 上の AP によるログ出力を検知するために VMM により AP の動作を監視すると，VM の性能が低下する．

(3) 多種の OS からログを取得するには，ログの取得機構をそれぞれの OS に対応させる必要がある．

(4) コードの追加によりバグが混入する可能性が増加する．

これらの問題を解決するために，ログの改ざんの影響を受けることのないログの転送手法が必要である．特に，adore-ng[67] のようなカーネルレベルマルウェアからのログの改ざんの防止を目指す．また，実際の運用を想定した場合には，性能低下の小さいログ取得手法が望ましい．さらに，仮想化環境では，複数の VM 上で多種の OS が動作する環境が想定されるため，OS から独立したログ取得手法が望ましい．また，追加するコード量は小さいことが望ましい．

3.2.3　考え方

文献 [80] で述べられているように，ログの取得機構を VMM に移植するのは，機構の安全性を向上するのに有効な手法である．しかし，既存のログ取得機構 [47] は Logging and Replay に着目している．これは，VMM が元来管理している情報を応用しているためである．この情報は，ディスクアクセス，センシティブ命令の実行，および特権命令の実行などを含む．一方，本研究では，syslog の扱うイベントログに着目する．

VMI のように，VM 内の情報を取得する手法が研究されている．しかし，VMI は性能低下が大きい．実際に利用されるシステムにおいて，性能低下は提供するサービスの品質の低下に繋がるため，情報取得における性能低下はできるだけ小さいことが望ましい．このため，本研究では，ログの取得における高速さに着目する．本研究では，安全にログを取得する機構に VMM を利用するだけでなく，VM 上のライブラリを改変することで，高速なログの取得手法の実現を目指す．

3.2.4　要求

目的を達成するために，以下の要件がある．

（要件 1） ログ保存経路におけるできるだけ早い段階でログを転送すること

（要件 2） VM からログを隔離すること

（要件 3） ログ取得機構自体を安全にすること

（要件 4） ログ取得機構を OS から独立させ，できるだけ小さくすること

（要件 5） ログの転送による性能低下をできるだけ抑えること

ログの転送経路において，プロセスの生成したログは，カーネルを経由する．これは，ログの転送にプロセス間通信が利用されており，カーネルがプロセス間通信の機能を提供しているためである．このため，カーネル空間におけるログの改ざんを防止するには，ログがカーネル空間に到達する前に VM 外部へ転送することが有効である．ログファイルの改ざんを防止するためには，ログファイルを VM から隔離する必要がある．ログの取得機構自体の安全性を向上するためには，機構を VM 外部に実現するのが有効である．また，OS への依存箇所を少なくすることで，多種の OS への移植は容易となる．また，プログラムサイズを小さくすることで，バグの混入の可能性を低減できる．また，既存のログ取得機構 [81] では，

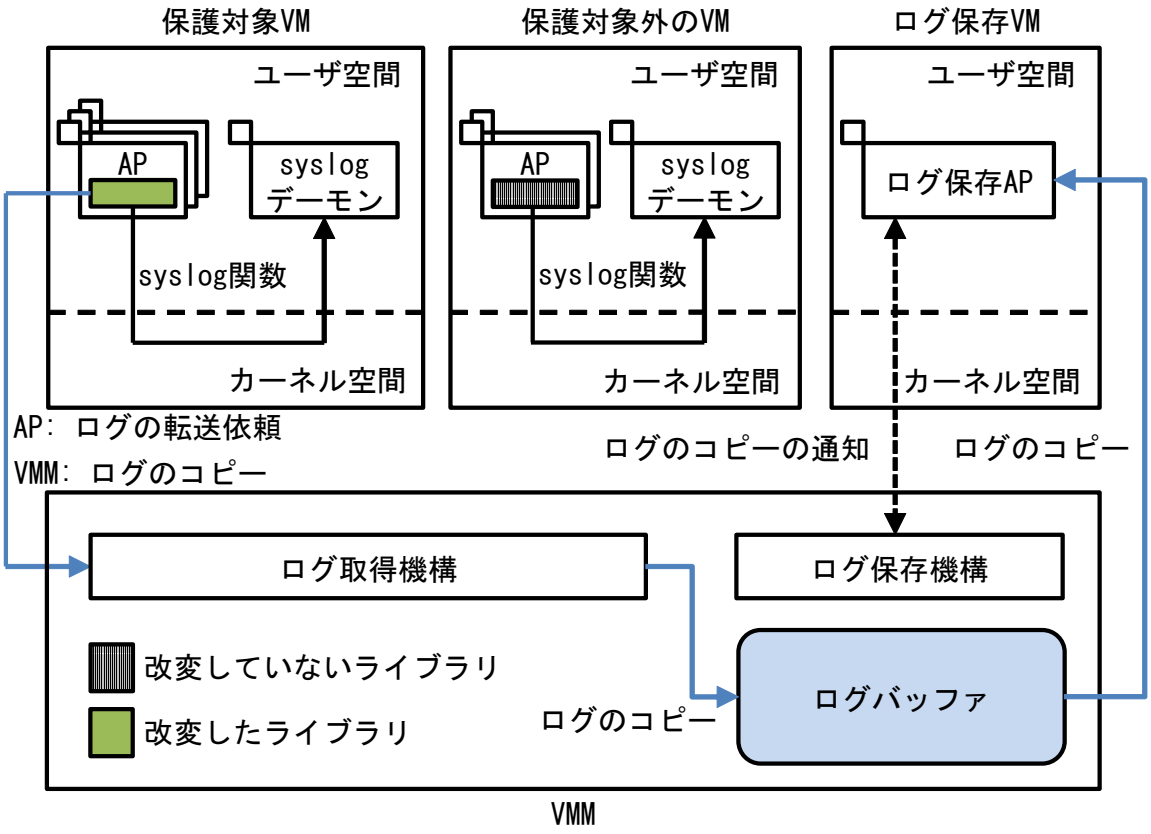

図　3.1　提案手法の全体像

VMM が VM 上の AP によるログ出力を検知してログを取得する．この際，AP からのログ出力を検知するために，AP の発行するすべてのシステムコールを VMM が検知する．システムコールの発行の検知では，システムコールが発行される度に VM exit を発生させる仕組みを用いる．このため，AP によるシステムコール発行のたびに性能低下が発生する．このことから，AP によるログ出力検知のための VM exit を削減することで，ログ取得による性能低下を抑えられる．

3.2.5　提案手法の全体像

提案手法の全体像を図 3.1 に示す．提案手法では，VMM がログの保護対象の VM からログを取得する．本研究では，すべての VM は Intel VT-x の機能により完全仮想化されているものとする．また，ログの保護対象 VM 上の AP が利用するライブラリは，改変したものと置き換える．ログの保護対象の VM 上の AP は，置き換えたライブラリにより，以下の手順でログを VM 外部へ転送する．

(1) AP は，置き換えたライブラリを用いて，VMM に対してログの転送を依頼

(2) VMM 内のログ取得機構は，ログの転送依頼を受け取り，AP から VMM 内のログバッファにログをコピー

(3) VMM は，ログ保存 VM 上のログ保存 AP へ通知し，ログ保存 AP は通知を受信すると，VMM に対してログのコピーを要求

(4) ログ保存 AP は，VMM からログを受信し，ファイルへ書き出し

提案手法では，保護対象 VM 上の AP は，VMM に対してログの転送を依頼する．AP は，ログの転送依頼に，置き換えたライブラリを用いる．置き換えたライブラリは，ログを送信する syslog 関数の呼び出しにより，VMM にログの転送依頼を発行するように改変しておく．VMM は，依頼を検知すると，保護対象 VM 上の AP からログをコピーし，VMM 内のログバッファへ格納する．コピーされたログは，ログ保存 VM へ転送され，ログ保存 AP がファイルへ書き出す．

ログを保護対象 VM からログ保存 VM に転送するために VMM を改変し，ログ取得機構，ログ保存機構，およびログバッファを VMM 内に追加した．また，ログの転送依頼を VMM に送信するために，libc ライブラリを改変した．改変したライブラリは，syslog 関数により syslog デーモンへログを送信する直前に VM exit を発生させる命令を実行し，その命令の実行がログの転送依頼の契機となる．改変したライブラリを持つ VM からのみ，ログの転送依頼を VMM に発行できる．図 3.1 において，保護対象 VM は syslog 関数の呼び出しごとにログの転送依頼を発行する一方で，保護対象外の VM からはログの転送依頼は発行されない．

提案手法は，様々な種類の VMM に適用できる．VMM は，ベアメタル型 (タイプ 1) とホスト型 (タイプ 2) の 2 種類に分類できる [21]．ベアメタル型 VMM は，VM 上で実行される特権命令やセンシティブ命令のみをトラップする．ホスト型 VMM は，すべての命令を代わりに実行する．ベアメタル型 VMM に提案手法を適用する場合には，上述したように，VM 上の一部を書き換える必要がある．一方で，ホスト型 VMM では，VMM が命令を代理で実行するため，VM で実行される命令を検出できる．このため，提案手法をホスト型 VMM に適用するのは容易である．本研究では，性能低下の小ささから，ベアメタル型 VMM を対象とした実現方式について検討している．特に，Intel VT-x を利用する VMM においては，本研究で検討した手法を改変なしに利用できると考えられる．

VM 上での syslog 関数呼び出し直後に VMM がログを取得するため，提案手法は，(要件 1) を満たす．VM から VMM へのログの転送経路にカーネル空間は含まれないため，カーネル空間でログを改ざんすることはできない．ログの保存にログ保存 VM を利用して VM 間

でのログファイルの隔離を実現することから,（要件 2 ）を満たす．メモリ，ネットワーク，およびディスクを含む計算機資源は，VM ごとに確保され，他の VM とは隔離されている．このため，VM の外部に保存されたログファイルを改ざんするのは難しい．また，VM から VMM を攻撃するのは難しいことから，提案手法は,（要件 3）を満たす．ライブラリの改変は OS の構造には依存しないため，多種の OS への適応は容易である．このため，提案手法は,（要件 4）を満たす．VM exit の発生頻度については，文献 [81] で提案した手法ではシステムコールの発行ごとに VM exit が発生するのに対し，提案手法では，ログの転送依頼の場合にしか追加の VM exit は発生しない．このことから，提案手法は,（要件 5）を満たす．

3.2.6　提案手法と VMI の比較

提案手法と VMI は，VM 内部の情報を取得するという観点では，類似している．しかし，以下の差異がある．

- ログの安全性
- VM 内のデータ構造への依存
- 導入による性能低下

提案手法は，VMI よりもログの安全性の点で優れている．VMI は，VM のハードウェアの状態やイベントの発生を監視する．この方法では，VM 上におけるログの生成を即座に検知するのは難しい．VMI がログの生成を検知した場合でも，カーネルモードへの遷移後に検知した場合には，ログは既に改ざんされている可能性がある．一方，提案手法では，ログを送信する関数の呼び出しが VMM へのログの転送の契機となるため，カーネルレベルマルウェアはログを改ざんしたとしても，既に VM 外部へ転送済みであるため，転送済みのログへの改ざんによる影響はない．

VM の状態を監視するために，VMI は，VM 内のカーネルのデータ構造に強く依存した情報を収集する．このため，VMI は，VM 上のデータ構造について事前に十分な情報を取得しておく必要がある．また，VM の状態を監視するために，VMI は非常に多くの情報を取得する．取得する情報には，たとえば，プロセスリストやプロセス情報などがある．これらの情報は，OS のバージョンに非常に強く依存している．

このように，VMI は VM の状態を詳細な情報をもとに検査できる．しかし，詳細な情報を取得するため，VM 上のデータ構造に強く依存する．また，性能低下が問題となる．一方，提案手法は，VMI のように詳細な情報は取得しないものの，VM 上のデータ構造への依存度

は小さい．また，情報取得による性能低下は小さい．文献 [46] では，VMI は，プロセスの生成を監視するために，690%の性能低下が起こる一方で，VM 内部にエージェントを挿入する手法は，13.7%の性能低下しか起きないと述べられている．このことから，VM 内部にエージェントを挿入する手法は，性能面では，非常に有効であることが分かる．提案手法は，VM 内部のライブラリの一部を改変して情報を取得するエージェントの機能を追加していると考えられることから，VM 内部にエージェントを挿入する手法と考えることができる．また，提案手法は，syslog 関数の呼び出しのみを監視する．このため，提案手法の適用による性能低下は，AP が syslog 関数を呼び出した時しか発生しない．このため，提案手法の適用による AP 全体の性能への影響は，十分小さくなることが予測できる．

3.3　実現方式

3.3.1　ログの転送処理の流れ

ログの VM から VMM への転送は，AP によるログの転送依頼と VMM によるログの取得に 2 段階からなる．本章では，これらの処理の実現方式について述べる．AP から VMM へのログの転送依頼について 3.3.2 項で，保護対象 VM から VMM へのログのコピーについて 3.3.3 項で述べる．`libc` ライブラリへの修正については，3.5.3 項で述べる．

保護対象 VM 上の AP からログを VMM に転送した後，VMM は，ログ保存 VM にログを転送する．ログ保存 VM へのログの転送については，3.3.4 項で述べる．

3.3.2　AP から VMM へのログの転送依頼

AP から VMM へのログの転送依頼は，`cpuid` 命令をライブラリ内に埋め込むことで実現する．`cpuid` 命令は，CPU の情報を取得するための命令であり，実行により CPU や VM の状態は変化しない．VT-x による完全仮想化環境においては，`cpuid` 命令が実行されると，CPU の状態に関わらず VM exit が発生する．このため，`cpuid` 命令を VM 上のライブラリに埋め込むことで，VMM へログの転送を依頼する．転送依頼のインタフェースを表 3.1 に示す．埋め込んだコードは，表 3.1 の仕様に従いレジスタに値を格納し，`cpuid` 命令を実行する．ライブラリへ埋め込んだコードは，3.5.3 項で説明する．

メモリの検査による提案手法の検知を困難にするためにも `cpuid` を利用している．VMM を呼び出す命令としては，`vmcall` 命令が利用される．しかし，`vmcall` 命令を利用すると，攻撃者は，メモリを検査して `vmcall` 命令を検出した場合には，提案手法の存在を検知でき

表　3.1　ログの転送依頼のインタフェース

レジスタ	説明
rax	0xffff: ログの転送依頼を意味する値
rbx	ログを格納しているメモリ領域の先頭アドレス
rcx	転送を依頼するログの長さ

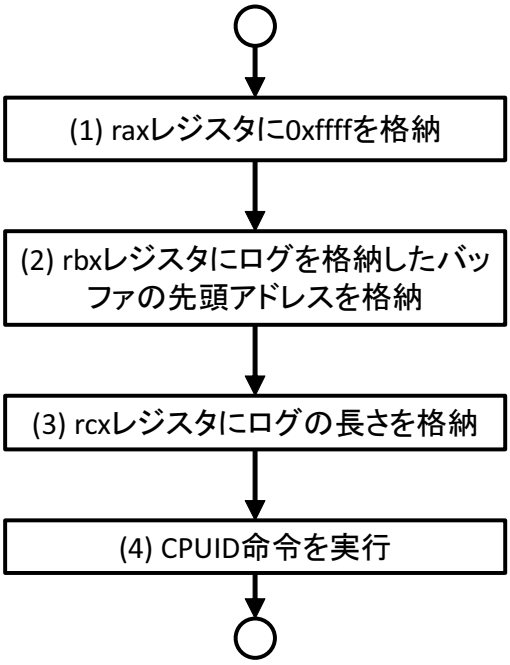

図　3.2　CPUID 命令を用いた VMM へのログの転送依頼

る．一方，cpuid 命令は，Debian で標準ライブラリとして利用される eglibc-2.11.3 を調査したところ，13 箇所で cpuid 命令が利用されていた．このため，攻撃者が提案手法を検知するためには，メモリ上の cpuid をすべて検査し，既存のライブラリと比較して差異を検出する必要がある．提案手法では，cpuid 命令を一つ追加するのみである．このため，cpuid 命令を用いた場合には，vmcall を用いる場合よりも，提案手法の検知は困難である．

図 3.2 は，ログの転送依頼の処理を示している．まず，保護対象 VM 上の AP は，0xffff を rax レジスタに格納し，ログを格納したバッファの先頭アドレスを rbx レジスタに格納し，ログの長さを rcx レジスタに格納する．その後，AP は，cpuid 命令を実行することで，ログの転送依頼を発行する．

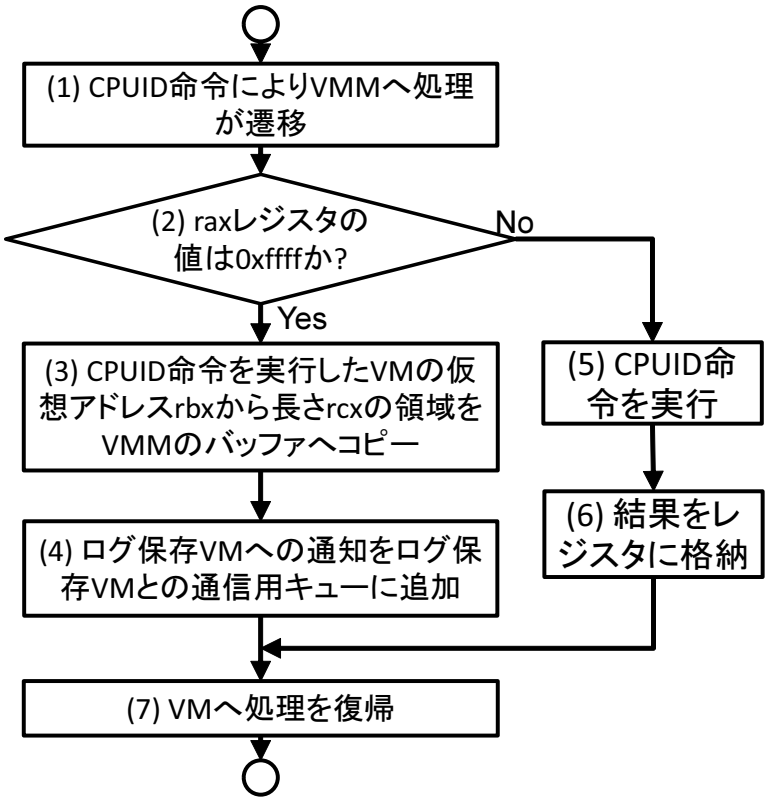

図 3.3 転送依頼受信時の VMM によるログのコピーの処理の流れ

3.3.3 保護対象 VM から VMM へのログのコピー

図 3.3 に VMM によるログのコピー処理の流れを示す．cpuid 命令は，ログ転送の契機として利用され，VMM は，cpuid 命令の実行を検知すると，AP から VMM へログをコピーする．VMM は，ログのコピーが完了すると，ログ保存 VM にログの転送を通知し，処理を VM に返す．ログのコピーにおいて，rax レジスタの値が 0xffff でない場合には，ログをコピーせず，cpuid 命令をエミュレートし，処理を VM に返す．

図 3.4 に示すように，VMM 内のログバッファはリングバッファとして実現している．これは，高負荷な環境において大量にログの転送依頼が VM から送信された場合でも，ログの消失をできるだけ防止するためである．複数の VM からログの転送依頼が送信された場合には，すべてのログは同一のログバッファに格納される．それぞれのログがどの VM から取得したものかは，ログの先頭に VM ごとの ID (VMID) を付加することで判別する．

手順 (3) において，ログの長さがログバッファの空き領域よりも長い場合には，VMM は保護対象 VM をサスペンドし，ログの一部のみを取得する．その後，ログバッファに蓄積したログのログ保存 VM への転送が完了すると，VMM はログの取得を再開する．VMM は AP

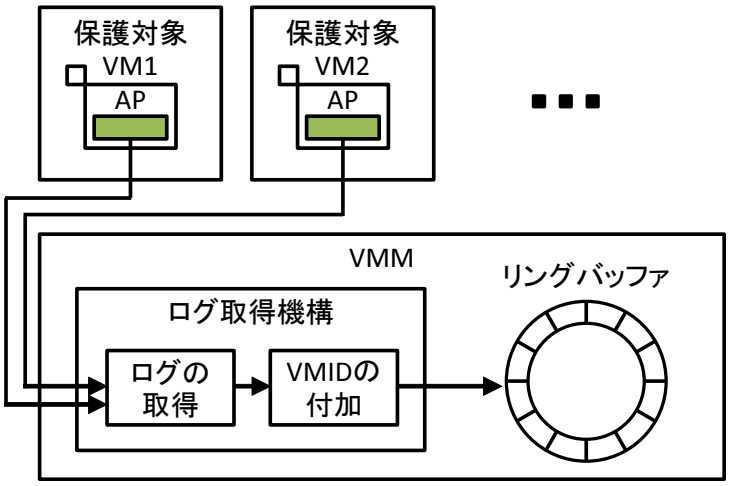

図 3.4 ログ取得機構によるログの取得とリングバッファへの格納

上にログが残っている限り，同様の手順を繰り返す．このため，AP が VMM 内のログバッファよりも長大なログの転送を依頼しても，ログを消失させることなく，VMM はすべてのログを取得し，ログ保存 VM に転送できる．保護対象 VM はログ転送時にサスペンドされるため，長大なログの転送依頼を大量に送信することによるログ消失の可能性はない．しかし，長大なログやログの大量の転送依頼を送信された場合には，VM の性能は大きく低下する．

3.3.4 ログ保存 VM へのログのコピー

ログ保存 VM へのログのコピーは，ログのコピー回数をできるだけ削減するため，非同期に行う．VMM は，ログがログバッファに蓄積されたことをログ保存 VM に通知する．ログ保存 VM 上のログ保存 AP は，通知を受け取ると，ログを受信する準備ができたことを VMM に通知する．VMM は，通知を受け取ると，VMM はログバッファに蓄積されたログをログ保存 AP に送信する．ログ保存 AP は，ログを syslog 関数を用いてログファイルに書き出す．ログのフォーマットは，ログの各エントリに VMID が付加されている以外は，syslog による通常のログと同じである．このため，既存のログ解析ツールを利用できる．さらに，ログを安全に保存する既存手法やログファイルの暗号化手法を改変なしに提案手法に適用できる．

3.4　分析と考察

3.4.1　提案手法の限界

攻撃者が提案手法の存在を前提として攻撃する場合，攻撃者はログを出力しないような AP を作成して既存の AP と置き換える可能性がある．また，攻撃者の作成した AP が提案手法を回避する目的ではなく，別の目的を持った AP であった場合は，libc を動的リンクする場合には，提案手法を適用した libc を利用するため，提案手法によるログ取得が可能である．しかし，攻撃者の作成した AP が libc を静的リンクしていた場合には，提案手法を適用できない．しかし，ログを転送しないような AP や libc を静的リンクした AP に置き換えられた場合には，提案手法により取得するログの量が極端に減少することが予想される．このため，提案手法で取得するログの量を定期的に比較することで，攻撃者による AP の置き換えを検出できる可能性がある．

3.4.2　ログの転送経路における安全性

syslog によるログの保存経路は，AP によるログの生成，AP から syslog デーモンへのログの転送，syslog デーモンによるログの受信，ログファイルまたはリモート計算機の syslog デーモンへログを転送する間，およびログファイルへ書き込み完了後の 5 段階に分けられる．このことから，ログは，(1) プロセスによるログ生成時，(2)AP によるログ送信から syslog によるログ受信時，(3)syslog デーモンがログを受信してファイルに書き出すかリモートの syslog デーモンへ転送するまでの間，および (4) ログファイルへ出力後の 4 箇所で攻撃により改ざんされる可能性がある．

adore-ng[67] のようなカーネルレベルマルウェアは，(2) と (3) の契機でログを改ざんできる．tuxkit[66] のような syslog デーモンへの攻撃では，(3) の契機でログを改ざんできる．また，攻撃者は，攻撃対象の計算機における管理者権限を取得した場合には，(4) の契機でログを改ざんできる．

提案手法では，保護対象 VM 上のログは，AP から syslog デーモンへ転送される直前で VM 外部へ転送される．一旦ログが VM 外部に転送されると，保護対象 VM 上の攻撃者は，転送後のログを改ざんできない．このことから，提案手法手法は，(2),(3), および (4) の契機におけるログの改ざんを回避できる．保護対象 VM 上の攻撃者がログを改ざんしたとしても，VM 外部へ転送したログには一切影響はない．

3.4.3　ログの改ざん実験

提案手法によりログの改ざんを防止できるかを確認するために，ログの改ざん実験を行った．はじめに，ログを改ざんする機能を持つカーネルレベルルートキットである adore-ng [67] を用いて，カーネル空間におけるログの改ざんを提案手法により防止できるか実験した．adore-ng は，メモリ上のカーネルコードにパッチを適用し，ソケット通信によるプロセス間通信を監視する．プロセス間通信において，攻撃者に都合の悪い文字列を検出した場合には，そのメッセージを削除する．この実験の結果，保護対象 VM 上で adore-ng によりログが改ざんされた場合でも，VMM に転送されたログは改ざんの影響を受けないことを確認した．これにより，カーネル空間におけるログの改ざんを提案手法により防止できることを確認した．また，ログが改ざんされた場合，保護対象 VM 上のログファイルとログ保存 VM 上のログファイルを比較することで改ざんの有無や改ざん箇所を特定できることを確認した．

次に，syslog デーモンの設定ファイルを編集してログの書き込みを制限した．具体的には，ログの書き出しポリシを編集して，ログを一切書き出さないようにした．この場合，syslog デーモンが走行していても，ログはログファイルには一切出力されない．このような状況においても，提案手法では，ログを漏れなく取得できた．この結果より，提案手法は，ログの書き出しポリシを編集する攻撃に耐性があることを確認した．また，この結果は，syslog デーモンを置き換えるような攻撃を受けたとしても，提案手法により取得したログには影響がないことも示している．このため，提案手法は，tuxkit [66] による攻撃に対処できる．

次に，保護対象 VM 上の syslog デーモンを停止した．この場合，syslog デーモンによるログ出力は発生しない．このような状況でも，提案手法は，ログを取得できた．しかし，これは，ログの転送処理に依存する．`GNU libc` では，syslog 関数は，syslog デーモンとの接続を確立できない場合には，ログの転送を中止する．我々のプロトタイプでは，コネクション確立前に，ログの整形が完了した直後に，ログの転送依頼を送信するように，`libc` を修正している．これにより，提案手法では，syslog デーモンが停止した状態でもログを取得できる．つまり，syslog デーモンとの接続確立後にログの転送依頼を送信するように `libc` を修正した場合には，syslog デーモンが停止すると，提案手法ではログを取得できない．

最後に，保護対象 VM 上のログファイルを改ざんし，提案手法で取得したログに影響があるか確認した．ログファイルを改ざんするマルウェアとして LastDoor [82] がある．LastDoor は，指定した文字列を含むログを削除する機能を持つ．提案手法は，ログファイルに書き込まれる前にログを VM 外部に転送するため，LastDoor により保護対象 VM 上のログが改ざんされても，提案手法で取得したログには影響しない．

これらの結果より，提案手法は，多くの場合でログを取得でき，取得したログは攻撃の影

響を受けないことを確認できた．提案手法では，ログを送信しないような AP に置き換えるような攻撃には対処できない．しかし，攻撃者は，ログを改ざんする場合には，それぞれの AP を攻撃するのではなく，すべてのログが通過する箇所にログを改ざんするマルウェアを挿入することが多い．たとえば，adore-ng は，カーネル内の関数に挿入される．また，tuxkit は syslog デーモンを置き換える．syslog 関数により送信されたログはカーネルや syslog デーモンを経由するため，少ないコストで syslog を用いるすべてのログを監視できる．このことから，ログの生成元の AP への直接攻撃は稀であると推測できる．

3.4.4　修正したライブラリの保護

保護手法

提案手法への攻撃方法として，修正したライブラリのファイルを置き換える攻撃とライブラリのロードされたメモリ領域を改ざんする攻撃が考えられる．

ライブラリのファイルを置き換える攻撃に対しては，ライブラリのファイルの完全性検証が有効である．プロセスがライブラリを読み込む際に，読み込むファイルの完全性を検証することで，ライブラリの置き換えの有無を検知できる．ライブラリのファイルを検証するためには，事前にファイルのハッシュ値を取得しておく必要がある．ファイルがメモリに読み込まれる際に，VMM はシステムコールを検知し，ライブラリのファイルのハッシュ値を測定し，VMM 内に保存しておいたハッシュ値と比較する．この際，ハッシュ値が異なっていれば，ライブラリのファイルの改ざんを検知できる．

メモリ上にロードされたライブラリの完全性の保証には，走行中プロセスのメモリの完全性を VMM により保証する手法 [83] が有効である．カーネルレベルマルウェアは，ユーザプロセスのメモリ空間にアクセスできるため，メモリ上にロードされたライブラリを改ざんできる．文献 [83] の手法では，指定したプロセスのメモリ空間はカーネルからは書き換え不可能になる．VM の利用する仮想メモリは実メモリには直接マッピングされず，シャドウページテーブルにより，ゲスト仮想アドレスと実アドレスの対応付けが維持される．そこで，指定したプロセスとそれ以外でシャドウページテーブルを分割し，指定したプロセスのメモリへのアクセスで例外を発生させる．指定したプロセスからは自身の仮想メモリ空間を参照できる．しかし，カーネルやその他のプロセスからは，保護されたプロセスのメモリにはアクセスできない．この手法を提案手法と併用することで，提案手法は，メモリ上にロードされたライブラリを改ざんする攻撃に対処できる．

性能への影響の予測

ライブラリの保護による性能低下が予測できる．特に，ライブラリのファイルの検証では，ファイルの完全性検証を行うため，性能低下が大きいことが予想できる．一方で，メモリにロードされたライブラリの保護による性能低下は小さいと予想できる．改変したライブラリをロードしたメモリ領域への書き込みを禁止した場合でも，その領域の読み込みや実行は可能である．書き込みの禁止による性能低下は，書き込みを禁止した領域への書き込みによってのみ発生する．しかし，通常は，ライブラリのメモリを書き換えるような処理は発生せず，攻撃によりライブラリを書き換えようとした場合にのみ性能低下が発生する．このため，ライブラリをロードしたメモリ領域への書き込みの禁止による性能低下は非常に小さいと推察できる．

性能低下の予測において，改変したライブラリの検証における処理時間は，ライブラリの保護における処理時間において支配的である．このため，性能低下を予測するために，ファイルの完全性を検証する機能のプロトタイプを作成した．プロトタイプは，ログ保存 VM 上の AP から保護対象 VM 上のライブラリのファイルのハッシュ値を md5 により取得し，完全性を検証する．ハッシュ値の計算には，mhash ライブラリを用いた．保護対象 VM 上のライブラリのファイルの読み込みには，guestmount を用いた．性能低下を予測するために，保護対象 VM 上のライブラリのファイルを読み込み，ハッシュ値を作成する処理にかかる時間を測定した．測定に用いたライブラリは，`libc-2.13.so` であり，ファイルサイズは，1,595,408 バイトである．実験環境は，他の性能測定を実施した表 3.2 の環境と同じである．

測定の結果，libc ライブラリのハッシュ値の計算に必要な時間は，48 ミリ秒であった．この処理時間大半は，ディスクアクセスによるものだと推測できる．48 ミリ秒の処理時間は十分大きいと予想できる．しかし，この処理時間は，プログラムの起動時のみ発生するものであることから，VM 上の AP の処理時間への影響は小さいと予測できる．

3.4.5　提案手法を用いた DoS 攻撃

3.4.4 項で述べた方法は，ログを転送するプロセスを認証していない．このため，不正なプロセスが大量にログの転送依頼を発行し，計算機へ高負荷を与える攻撃が可能である．この攻撃では，ログの転送に VMM が関わることから，同一計算機上で動作するすべての VM の性能に影響を与える．このため，大量にログを送るような攻撃を防止する必要がある．

この問題への対処として以下の方法がある．

(1) ログの転送依頼を発行可能なプロセスを認証する方法

(2) 一定期間内に大量の転送依頼を発行するプロセスを不正なプロセスとみなし，ログの転送依頼を禁止する方法

(1) の方法では，認証により，不正なプロセスによる攻撃を防止できる．しかし，どのプロセスを正常なプロセスとし，どのプロセスを不正なプロセスとするかの判定基準を設ける必要がある．

(2) の方法では，他の VM の性能へ高負荷を与えるようなログの転送依頼を発行するプロセスを特定するために，期間 T とログの転送依頼の発行回数 N を用いる．期間 T 中に N を超える回数のログの転送依頼が発行された場合は，そのプロセスを不正なプロセスと判断する．

不正なプロセスとして判定した後の対処として，ログの転送依頼を受け付けないように VMM が対処する方法と AP を終了させる方法がある．ログの転送依頼を受け付けないようにした場合，一定時間経過後にログの転送依頼の発行回数が減少していた場合には，転送依頼の受付を再開するなど，柔軟に対処できる．しかし，すべてのプロセスによるログの転送依頼を監視し，管理する必要があるため，監視のコストが増大する．一方，プロセスを終了させる方法は，プロセスを終了させた後に AP の状態を管理する必要がないため，管理コストは小さい．しかし，期間 T とログの転送依頼の発行回数 N の設定が不適切な場合，誤って正常なプロセスを不正なプロセスと判断した場合に，重要なプロセスを終了させてしまう危険がある．このため，プロセスの認証は，今後の課題である．

3.4.6　様々なログ転送契機への提案手法の適用

提案手法は，syslog 関数の発行に着目している．この手法は，AP から VMM への転送依頼に依存しているため，syslog に限らず適用できる．たとえば，Apache Web サーバは，ログを syslog を用いずに独自のログファイルに保存する．そこで，Apache によるログ出力のインタフェースに `cpuid` 命令を埋め込むことで，ログの転送依頼を発行できる．Web サーバにおけるログ出力は，Web サーバの性能低下の大きな原因の一つであるため，提案手法の適用は有効であると考えられる．ただし，提案手法を様々なイベントに適用すると，維持管理コストが増大する．このため，提案手法の適用は，維持管理コストとのトレードオフの関係にある．

3.5　評価

3.5.1　評価の目的と評価環境

以下の観点で提案手法を評価した．

- ログ取得の完全性
  ログ取得の完全性を評価するために，大量のログ転送依頼を送信することで，高負荷な環境を作成し，提案手法を評価した．

- 多種の OS への適用
  多種の OS への適用の容易さを評価した．

- 性能
  syslog 関数，システムコール，およびデータベース管理システムにおいて，提案手法を適用した場合の性能を評価した．

- 複数の VM が走行する環境における性能
  複数の VM が走行する環境における Web サーバの性能を評価した．また，同時に走行する VM 数を変化させた場合の性能の変化を測定し，提案手法の VM 数に対するスケーラビリティを評価した．

- メモリ使用量
  提案手法の適用によるメモリ使用量の増加を評価した．

評価に用いたソフトウェア環境を表 3.2 に示す．提案手法のプロトタイプは，Xen[16] を用いて実現した．

3.5.2　ログ取得の完全性

保護対象 VM 上で出力されたログを提案手法により漏れなく取得できることを確認するために，高負荷な環境でログを取得できるか実験した．実験では，10,000 回連続してログの転送依頼を発行した．転送依頼の開始から終了までの時間は，0.26 秒であった．1 回の転送依頼におけるログの長さは 30 バイトとした．実験の結果，すべてのログは漏れなくログ保存 VM に転送できたことを確認した．この結果より，提案手法は，高負荷な環境においても，十分に運用可能であることを確認できた．

表 3.2 評価に用いたソフトウェア

VMM	Xen 4.2.0
OS (保護対象 VM)	FreeBSD 9.0.0 64-bit
	Debian (Linux 2.6.32 64-bit)
OS (ログ保存 VM)	Debian (Linux 3.5.0 64-bit)
Web サーバ	thttpd 2.25b
データベース管理システム	PostgreSQL 9.2.4
syslog デーモン	rsyslogd 4.6.4
ベンチマークソフト	ApacheBench 2.3
	pgbench 9.2.4
	LMbench version 3

```
void
  __vsyslog_chk(int pri, int flag, const char *fmt, va_list ap)
  {
            int saved_errno = errno;
            char failbuf[3 * sizeof (pid_t) + sizeof "out of memory []"];

+           reg_t regs;
+           regs.rax = 0xffff;
+
  #define   INTERNALLOG LOG_ERR|LOG_CONS|LOG_PERROR|LOG_PID
            /* Check for invalid bits. */
            if (pri & ~(LOG_PRIMASK|LOG_FACMASK)) {
***************
*** 278,283 ****
--- 297,308 ----
            if (LogType == SOCK_STREAM)
              ++bufsize;

+           regs.rbx = (unsigned long)buf;
+           regs.rcx = bufsize;
+
+           cpuid_logxfer(regs.rax, &regs);
+           regs.rax = regs.rbx = regs.rcx = 0;
+
            if (!connected || __send(LogFile, buf, bufsize, send_flags) < 0)
              {
                if (connected)
```

図 3.5 提案手法の適用前と適用後のライブラリのソースコードの差分

3.5.3 多種の OS への提案手法の適用

修正工数として，プログラムを修正するために追加した行数と修正箇所を決定するコストがある．プロトタイプでは，FreeBSD 9.0.0 上の libc.so.7 内の syslog 関数に提案手法を適用

```
void cpuid_logxfer(unsigned long rax, reg_t *reg)
{
        __asm__ __volatile__(
                "cpuid"
                :"=a"(reg->rax)
                :"a"(rax), "b"(reg->rbx), "c"(reg->rcx)
        );
        return;
}
```

図 3.6 cpuid_logxfer() 関数の定義

した．追加した処理は，CPUID 命令を実行する処理と CPUID 命令の実行前にレジスタに適切な値を設定する処理である．プログラムに追加した行数は 20 行である．

libc の修正箇所を図 3.5 に示す．また，ログの転送依頼を発行するために定義した cpuid_logxfer 関数の定義を図 3.6 に示す．

追加内容は，cpuid_logxfer 関数の定義，reg_t 構造体の定義，rax, rbx, rcx レジスタへの値の設定，および cpuid_logxfer 関数の呼び出しである．これらのうち，改変するライブラリ中の syslog 関数の構造に依存しているのは，cpuid_logxfer 関数の呼び出し箇所だけであり，既存のライブラリへの依存は，十分小さいといえる．このため，提案手法の多種の OS への適用は，容易であるといえる．修正箇所は，ログを送信する前に文字列を成型する処理の直後とした．UNIX 系 OS では syslog 関数によりログを転送するため，対応する関数名をライブラリから探索することでライブラリの修正箇所を特定した．

以上のことから，FreeBSD 9.0.0 上の libc.so.7 へ提案手法を適用するためのライブラリへの修正箇所は 20 行である．このことから，提案手法を多種の OS へ適用する場合において，ライブラリの修正工数は小さく，適用は容易であるといえる．

3.5.4　OS 構造への依存

3.5.3 項で述べたように，提案手法は，libc に依存している．提案手法は，OS の構造に依存していないものの，各 OS は独立したライブラリを保持している．本章では，提案手法と既存手法において，OS 構造への依存度を定量的に比較する．

はじめに，提案手法が libc に依存するコード量を分析する．提案手法は libc に 20 行のコードを追加する必要がある．しかし，追加コードの多くは，cpuid_logxfer() 関数と regs 構造体の定義である．このため，libc に依存しているコードは，レジスタに適切な値を設

定する処理と cpuid 命令の実行である．これらの処理を syslog 関数に追加する箇所のみが提案手法の実現において必要な処理である．このことから，OS やライブラリへの依存箇所は非常に少ないといえる．

次に，既存手法の OS への依存度を分析する．VMI[45] や CloudSec[48] は，カーネル内で利用されるデータ構造にしたがって VM の利用するメモリを解析する必要があるため，OS のデータ構造に非常に強く依存している．このため，カーネルのバージョンが更新された場合には，これらの手法が利用するデータ構造の定義を更新する必要がある．このため，これらの手法は，OS の構造に非常に強く依存している．VM 内にエージェントを挿入する手法 [46] は，VMI や CloudSec よりも OS のデータ構造への依存度は低い．SIM は，エージェントを VM 上のカーネルに挿入する．エージェントは，カーネルのベンダにより提供された手順に従って挿入する．VMM の改変により VM 上のログを取得する手法 [84] は，VM 上のカーネルにフックポイントを挿入するために，システムコールのエントリポイントのアドレスが必要である．また，カーネルログバッファのメモリ領域を VMM が把握しておく必要がある．このため，VMM は，OS の種類やバージョンと判別し，対応する必要がある．この手法は，AP や OS の修正が必要ないものの，OS のデータ構造へ依存している．

OS カーネルのデータ構造への依存は，手法の適用を困難にする．一般的に，OS カーネルやデバイスドライバを改変するのは，AP を改変するよりも困難である．このため，既存手法は OS カーネルに依存していることから，適用は困難である．一方で，提案手法は，ライブラリの改変のみが必要である．また，改変したライブラリを事前に作成しておくことで，VM の管理者は，ライブラリを置き換えるだけで提案手法を適用できる．以上のことから，提案手法の適用は，既存手法よりも容易であるといえる．

3.5.5　性能評価

評価項目と評価環境

提案手法による性能への影響を評価するために，syslog 関数，システムコール，および AP において，性能を評価した．また，rsyslogd による VM 外部へのログ転送と提案手法による VM 外部へのログ転送の性能を比較した．AP の性能評価では，複数 VM が走行する環境における AP についても，性能への影響を評価した．syslog 関数と AP の性能評価では，提案手法の適用による性能低下を明らかにする．システムコールの性能評価では，提案手法は syslog 関数でのみ性能低下が発生することから，提案手法の適用によるシステムコールの性能への影響はないことを確認する．

表 3.3 syslog 関数の性能

	Time (μs)	Overhead (μs (%))
Xen	31.47	–
提案手法	33.38	1.91 (6.08%)

評価は，Core i7-2600 (3.40 GHz, 4 コア) と 16 GB のメモリを持つ計算機で行った．評価では，各 VM には，一つの仮想 CPU と 1 GB のメモリを割り当てた．利用する仮想 CPU による性能の変化を抑制するために，ハイパースレッディングは無効とした．各仮想 CPU は，物理 CPU に固定的に割り当てた．これは，複数の仮想 CPU が特定の物理 CPU を利用することで，その仮想 CPU を利用する VM の性能が低下するのを避けるためである．

syslog 関数とシステムコール

提案手法では，AP が置き換えたライブラリの syslog 関数を呼び出した際に，ログの転送依頼を発行する．提案手法による性能低下を明らかにするために，提案手法を適用した Xen 上と提案手法を適用していない Xen 上に VM を作成し，各 VM 上での syslog 関数の性能を測定した．表 3.3 に測定結果を示す．提案手法による性能低下は，約 1.91 マイクロ秒であった．これは，syslog 関数の処理時間の 6.08%にあたる．syslog 関数は，頻繁に呼び出される関数ではないため，syslog 関数における 6.08%の性能低下が AP の性能へ与える影響は小さい．

提案手法による syslog 関数の性能低下が AP の性能に与える影響を予測するために，thttpd における syslog 関数の呼び出し頻度を測定した．測定は，Ubuntu 13.04 上で行なった．表 3.4 は，thttpd に 100 回 Web ページを要求した際のライブラリ関数の呼び出し回数の測定結果である．測定には，`ltrace` を用いた．表 3.4 より，ライブラリ関数呼び出し全体に占める syslog 関数の呼び出しの割合は，1%程度である．また，呼び出し回数だけでなく，性能への影響を測定するために，ライブラリ関数の処理時間について，呼び出し回数の測定時と同様に測定した．測定結果を表 3.5 に示す．表中の `__syslog_chk` 関数は，syslog 関数に相当する．表 3.5 に示すように，syslog 関数がライブラリ関数全体の処理時間に占める割合は，0.18%である．このことから，syslog 関数における 6.08%の性能低下が AP の性能へ与える影響は，十分小さいものと推測できる．

また，提案手法の適用によるシステムコールへの影響を評価した．評価には，LMbench[85] を用いた．LMbench は，ファイルの生成と削除，プロセスの生成，およびシステムコールの性能を含む様々な処理の性能を測定するベンチマークソフトである．提案手法は，syslog 関

表 3.4 thttpd Web サーバに Web ページを要求した際のライブラリ関数の呼び出し頻度

関数名	呼び出し回数	頻度 (%)	関数名	呼び出し回数	頻度 (%)
strncasecmp	1600	17.77	strftime	200	2.22
strlen	1400	15.55	accept	200	2.22
strcpy	800	8.89	gmtime	200	2.22
vsnprintf	600	6.67	__errno_location	200	2.22
memmove	400	4.44	time	100	1.11
strchr	400	4.44	close	100	1.11
select	301	3.34	read	100	1.11
gettimeofday	301	3.34	getnameinfo	100	1.11
strstr	300	3.33	strcat	100	1.11
fcntl	300	3.33	readlink	100	1.11
strpbrk	300	3.33	strrchr	100	1.11
strcasecmp	200	2.22	syslog	100	1.11
__xstat	200	2.22	writev	100	1.11
strspn	200	2.22			

数の発行時のみ性能低下が発生するため，LMbench による測定対象の処理では，性能は低下しないと推察した．測定結果より，LMbench による測定対象の処理では，性能は低下しないことを確認した．

rsyslogd によるログ転送との比較

ログ転送処理性能を測定するために，提案手法によりログを転送した場合，rsyslogd により UDP を用いて転送した場合，および rsyslogd により TCP を用いて転送した場合におけるログの転送に必要な処理時間を測定した．また，ログの転送による性能低下を測定するために，rsyslogd によりファイルに書き出す際の性能を測定し，転送に要する時間と比較した．rsyslogd の標準の設定でログを書き出した場合，ログの書き出し時刻は，秒単位でしか出力されない．このため，syslog の書き出しポリシを変更し，すべてのログに詳細な書き出し時刻を出力するようにした．評価では，ログを 1,000 回出力した．それぞれのログの長さは 300 バイトである．この長さは，Apache Web サーバにより出力されるアクセスログと同等の長さである．評価は，VM 上で行い，VM 上の rsyslogd によるログの転送先は，同じ計算機上で動作する異なる VM とした．

表 3.5 thttpd におけるライブラリ関数の処理時間の比率

関数名	処理時間の比率 (%)	関数名	処理時間の比率 (%)
writev	76.90	memmove	0.13
poll	17.71	gmtime	0.10
strncasecmp	0.82	strcasecmp	0.10
strlen	0.81	strftime	0.10
strcpy	0.51	strspn	0.09
close	0.30	strcat	0.09
__vsnprintf_chk	0.30	read	0.08
__xstat	0.23	getnameinfo	0.05
strchr	0.22	memcpy	0.05
fcntl	0.22	time	0.05
__syslog_chk	0.18	strrchr	0.05
accept	0.17	__strcpy_chk	0.04
readlink	0.15	malloc	0.02
strpbrk	0.14	mmap	0.00
strstr	0.14	open	0.00
__errno_location	0.14	realloc	0.00
gettimeofday	0.14		

表 3.6 rsyslogd により TCP および UDP でログを転送した場合と提案手法により転送した場合のレイテンシの比較

	ログの転送依頼を 1,000 回実行した際の処理時間 (ms)	ryslsogd によるファイルへの出力における処理時間との差 (ms)
ファイルへの出力	29.98	—
UDP による転送	57.13	27.16
TCP による転送	71.12	41.14
提案手法	37.76	7.78

表 3.6 に測定結果を示す．評価結果より，提案手法によるログの転送は，rsyslogd により UDP を用いた場合や TCP を用いた場合よりも高速であることが分かる．UDP を用いてログを転送した場合は TCP を用いた場合よりも高速であるものの，提案手法よりも低速であ

表 3.7 PostgreSQL における性能比較

tmpfs	VMM	TPS	相対性能
無効	Xen	400.37	–
	提案手法	395.76	0.99
有効	Xen	1,448.80	–
	提案手法	1,372.60	0.95

る．高負荷な環境を想定した場合，より高速なログ転送機構が好ましい．提案手法は，高負荷な環境でも高速にログを転送できる．

データベース管理システム

pgbench を用い，PostgreSQL により 1 秒間に処理されたトランザクション数（TPS）を測定した．測定時のクライアント数は 1 とした．提案方式が性能に影響を与えるのは，syslog 関数を用いた場合である．このため，測定では，1 回のトランザクションごとに処理時間をログとして syslog へ出力するよう設定した．また，ディスクアクセスによる性能のばらつきを抑えるために，tmpfs 上にファイルを作成し，性能を測定した．tmpfs はメモリ上にファイルを作成するため，ファイルアクセス時にディスクへアクセスしない．このため，ディスクのシーク待ち時間の差による性能のばらつきを抑制できる．また，処理時間に占める CPU 処理の割合が増加するため，CPU 処理の性能への影響を評価できる．

表 3.7 に測定結果を示す．表では，TPS の値が大きいほど性能が高い．tmpfs を用いた場合の結果より，PostgreSQL における CPU オーバヘッドは約 5%である．これは，PostgreSQL において，トランザクション処理の処理時間に対し，syslog 関数の処理時間の占める割合が低いためである．測定結果より，提案方式が PostgreSQL の性能に与える影響は小さいといえる．

また，表 3.7 より，tmpfs を用いない場合には，I/O 待ちにより CPU オーバヘッドが隠ぺいされるため，相対性能が向上しており，提案手法の導入にようる性能低下は 1%未満である．PostgreSQL を実際に運用する際は，ディスクアクセスなどにより I/O 待ちが生じる．このため，トランザクションごとの処理時間が増加する一方で，提案方式の導入による性能への影響は小さくなる．

PostgreSQL における評価では，提案方式の導入による性能低下は最大でも 5% 程度である．また，本評価では，提案手法の導入による性能低下を明らかにするために，トランザク

表 3.8 複数 VM を走行させた場合の Web サーバのスループット (要求数/s)

ファイルサイズ	VMM	VM 数						
		0	2	4	6	8	10	12
1 KB	Xen	1396.9	1329.27	1295.61	1225.22	1171.51	1231.72	1172.15
	提案手法	1231.06	1150.54	1057.95	1017.53	987.24	1015.69	946.61
	相対性能	0.88	0.87	0.81	0.83	0.84	0.82	0.8
10 KB	Xen	680.61	658.15	639.76	627.9	628.56	609.45	615.64
	提案手法	664.48	626.12	612.93	559.02	582.24	578.89	589.58
	相対性能	0.98	0.95	0.89	0.92	0.93	1.00	0.96
1,000 KB	Xen	11.41	11.41	11.4	11.39	11.38	11.39	11.39
	提案手法	11.41	11.41	11.4	11.39	11.37	11.39	11.06
	相対性能	1.00	1.00	1.00	1.00	1.00	1.00	0.98

ションごとにログを出力するように設定したものの，通常の DBMS の運用では，ログ出力の頻度は小さい．このことから，通常の DBMS の運用における提案手法の導入による性能低下は，より小さくなると推察できる．

複数 VM が動作する環境における性能への影響

複数 VM が同時に走行する場合の AP の性能への影響を評価するために，単一計算機上で複数 VM が走行する環境において，Web サーバの性能を測定した．これらの VM 上では，syslog 関数を毎秒 1 回呼び出すプロセスが走行する．本評価で用いた計算機には，4 つのコアを持つ CPU が搭載されている．各 VM のコアの割り当てでは，測定対象の VM の性能のばらつきを抑えるために，ログ保存 VM をコア 0 に，Web サーバを走行させる VM をコア 1 に，その他の VM をコア 2 とコア 3 に配置した．また，VM 数の増加による性能への影響を明らかにするために，コア 2 とコア 3 上の合計 VM 数を 0, 2, 4, 6, 8, 10, 12 の順に増加させ，各環境において性能を測定した．コア 2 とコア 3 の VM 数は等しくなるように VM を配置した．

表 3.8 に各環境における測定結果を示す．また，図 3.7 に各環境における性能の変化の様子を示す．同時に走行する VM 数が増加すると，Web サーバの性能も低下している．ファイルサイズが 10 KB 以上の場合，提案方式の導入による性能低下は 10%以下である．特に，ファイルサイズが 1,000 KB の場合，性能低下はほとんどない．これは，ログ取得の処理時間がファイルサイズに関係なく一定であるのに対し，Web サーバは，ファイルサイズが大きい程，処理時間が増加するためである．また，提案方式を導入した場合において，同時に走行する VM 数の増加による Web サーバの性能低下の度合いは，十分小さいといえる．同時

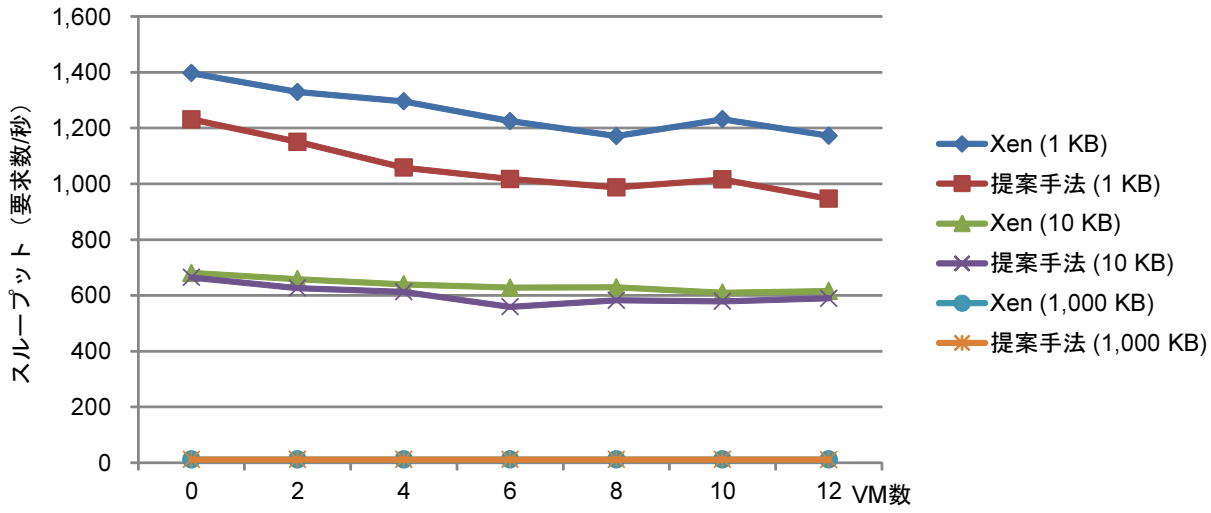

図　3.7　複数 VM が走行する環境における thttpd Web サーバの性能の比較

に走行する VM 数が変化した場合においても，性能低下の度合いはほぼ同じである．このため，提案方式は，複数 VM が同時に走行する環境においても，十分有用であるといえる．

メモリ使用量

提案手法によるメモリ使用量の増加を評価するために，修正なしの Xen と修正した Xen のメモリ使用量を比較した．実際のメモリ使用量を直接測定するのは難しい．Xen が認識するメモリ総量，Xen が認識する空きメモリ量，および Domain0 用に確保されたメモリ量は，Xen のメモリ使用量の推定に利用できる．メモリ総量と空きメモリ量は，`xl info` コマンドにより確認できる．Domain0 用に確保されたメモリは，`xl list` コマンドにより確認できる．`xl list` コマンドは，VM 名，ID，VM に割り当てられたメモリ量，VM の利用する VCPU 数，および VM が消費した CPU 時間を確認できる．これらの情報から，Xen と提案手法の利用するメモリ量を推定した．

表 3.9 に計算結果を示す．VMM によりメモリ使用量は，（メモリ総量）−（空きメモリ）−（Domain0 用に確保されたメモリ）により計算できる．測定結果からは，メモリ使用量の増加は観測できなかった．計算精度を考慮すると，メモリ使用量の増加は，1 MB 未満であると考えられる．このため，提案手法によるメモリ使用量の増加は軽微であるといえる．

さらに，提案手法の適用前と適用後の VMM の実行ファイルのサイズを比較した．修正なしの Xen の実行ファイルサイズは，805,133 バイトであった．一方，修正後の Xen の実行ファ

表 3.9 提案手法の適用前と適用後の VMM によるメモリ使用量の比較 (MB).

	Xen	提案手法
メモリ総量	16,288	16,288
空きメモリ	15,082	15,082
Domain-0 用に確保されたメモリ	1,024	1,024
VMM のメモリ使用量	182	182

イルサイズは，806,517 バイトであった．この比較より，提案手法の適用によって Xen の実行ファイルサイズは 1,384 バイト増加したことが分かる．

3.6　関連研究

3.6.1　安全なログ保存

Accorsi は，セキュアロギングに関するプロトコルを文献 [8] においてまとめている．この調査では，syslog の拡張 [28, 29, 26, 27] は，ログの保存ではなく，ログメッセージの転送経路における安全性を提供していると述べられている．本研究では，ログの転送経路における安全性に着目しており，これらの syslog 拡張と強く関連している．Accorsi によると，syslog 拡張の中でも，reliable syslog [29] のみが監査に利用する証跡の要件を満たす．ただし，これらのプロトコルは，ログの完全性検証が可能であるものの，ログの改ざんを防止できない．この点において，本研究と異なる．ログの完全性検証に関する研究は多くあるものの，改ざんを防止する研究は少ない．ただし，本研究は完全性検証の仕組みがないため，既存の完全性検証手法と本研究を組み合わせることで，より信頼性の高いログの保存が可能になる．

3.6.2　VM へのエージェント挿入による情報取得

SIM [46] は，VM 上にカーネルモードドライバとして情報取得のためのエージェントを挿入し，エージェントへの攻撃を VMM により防止している．本研究は，エージェントを新規に作成することなく，既存のライブラリを改変したライブラリと置き換えることで情報を取得している．また，本研究で取得する情報は syslog の扱うログに限定している．

3.7　まとめ

本章では，ライブラリの置き換えによるVM外部への低オーバヘッドなログ転送手法について述べた．本手法は，VM上のAPがログを発行する際に，カーネルに到達する前にVM外部へ転送する．このため，VM上のカーネルレベルで動作するマルウェアがログを改ざんしたとしても，VM外部に転送したログに影響しない．また，本手法をVMMにより実現することで，VM上のマルウェアから攻撃の困難な機構を実現した．さらに，本手法は，OSの構造に依存せず，ライブラリの置き換えによりログのVM外部への転送を実現することで，多種のOSへ容易に適用できる．さらに，VMMによりVM上のAPやOSの動作を監視するのではなく，VM 上の置き換えたライブラリからVMMへログの転送を依頼するため，VM上のAPやOSの動作を監視する処理が不要となり，監視により生じるオーバヘッドを削減することで，低オーバヘッドなログの転送を実現した．ただし，本手法は，低オーバヘッドでログを転送できる一方で，置き換えたライブラリを改変される可能性がある．このため，高負荷な環境では，本手法を用いることで低オーバヘッドでログを転送し，VM上のライブラリを置き換えない方が望ましい場合や，ログの転送機構の安全性を要求する場合は，ログの改ざんや消失を防止するシステムを利用するのが適している．

考察より，本手法でログを転送するよりも前にログを改ざんするのは難しいことを確認した．評価結果より，多種のOSへの適用では，ライブラリに20行のコードを追加するだけで適用できることから，適用の容易さを示した．性能評価では，本手法の導入によるsyslog関数の性能低下は6%であることを示した．また，データベース管理システムを用いた評価では，性能低下は1%未満であることを示した．このことから，入出力処理の負荷が高いAPにおいては，本手法の適用による性能低下は非常に小さいことを示した．また，複数VMが走行する環境における性能評価では，VM数の増加による性能低下への本手法の影響は小さいことを示した．

第 4 章

プロセス情報の不可視化によるプロセス特定の困難化

4.1　概要

本章では，プロセス情報の不可視化による攻撃回避手法について述べる．セキュリティソフトウェアは，攻撃者により無効化される可能性がある．セキュリティソフトウェアが攻撃され，停止または機能を無効化されると，攻撃の検知が遅れ，被害が拡大する可能性がある．VM 上のセキュリティソフトウェアへの攻撃を防止する手法として，VM 外部にセキュリティソフトウェアを実現する手法や VM 外部に既存のセキュリティソフトウェアを移植する方法がある．しかし，これらの手法は，移植や実現の工数や適用できるソフトウェアが限定される問題がある．この問題への対処として，攻撃者によるセキュリティソフトウェアへの攻撃をより困難にするために，VMM を利用し，プロセス情報の置換による攻撃回避手法を提案する．攻撃者は，プロセス情報をもとに攻撃対象のプロセスを特定するため，提案手法によりプロセス情報を偽の置換することで，攻撃者からの攻撃対象プロセスの特定を困難にし，攻撃を回避する．プロセス情報の置換では，VMM の改変のみで実現することで，VM 上の既存のセキュリティソフトウェアを改変することなく，攻撃を回避する．本章では，提案手法の適用により，プロセス情報をもとにした攻撃対象のプロセスの特定を困難にできることを示す．

4.2　想定する攻撃

想定する攻撃者は，管理者権限を取得可能であり，カーネル空間とユーザ空間のメモリに自由にアクセスできるものとする．このような攻撃者がプロセスを停止するような攻撃を想定する．管理者権限を取得した攻撃者は，攻撃者に不利益なプロセスを任意に停止できる．たとえば，攻撃を検知されないように，セキュリティソフトウェアを停止する攻撃が考えられる．また，ログイン履歴や操作履歴を記録されないように，ログ収集プログラムを停止する攻撃が考えられる．

このような攻撃の例として，Agobot[12] を用いた攻撃がある．Agobot は，セキュリティソフトウェアを停止する機能を持つ．セキュリティソフトウェアが攻撃により停止した場合，攻撃の検知や対処が遅れる問題やシステムを管理不能になる問題がある．このため，本章では，重要サービスとしてセキュリティソフトウェアを想定し，重要サービスへの攻撃への対処を検討する．

4.3　プロセスの特定を困難にする攻撃回避手法

4.3.1　目的

本研究の目的は以下の二つである．

(目的 1) 重要サービスへの攻撃の回避

(目的 2) 既存ソフトウェアの改変なしの利用

攻撃防止のために多くの手法が提案されているものの，未知の攻撃への対処の難しさや攻撃の様々な手法を組み合わせた高度な攻撃により，完全に攻撃を防止するのは難しい．このため，攻撃を前提とし，攻撃による被害の抑制を目的とする．本研究では，攻撃の防止ではなく，重要サービスへの攻撃を回避することを目的とする．また，既存ソフトウェアの機能を VMM に移植する工数は大きい．このため，改変なしの既存ソフトウェアの利用を目的とする．

4.3.2　基本方式

本研究では，攻撃者が攻撃対象のサービスを攻撃する際にサービスの存在を特定することに着目し，重要サービスを提供するプロセスに関する情報（以降，プロセス情報）を不可視

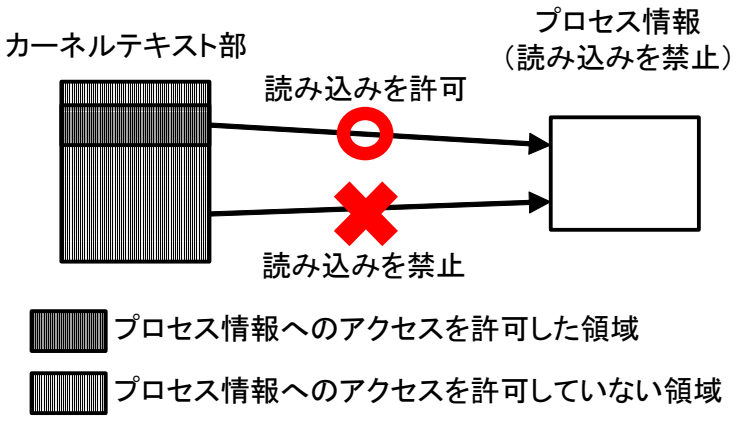

図　4.1　プロセス情報へのアクセス制御

化することにより，当該プロセスの特定を困難にし，攻撃を回避する．具体的には，攻撃者が重要サービスのプロセス情報を参照した場合には，偽の情報を返すことで，本来のプロセス情報を攻撃者から隠す．また，この機能を提供する機構自体を VMM 内に実現することで，機能への攻撃を困難にする．

VMM は，VM を提供するための機能に限定して開発されており，ゲスト OS からアクセスするインタフェースも少なく，ソースコードの規模も OS と比べて小さい．このため，VMM への攻撃は OS への攻撃よりも困難であるといえる．また，提案手法は，ゲスト OS や重要サービスのソースコードの変更や機能の追加は行わないで実現できる．VMM のみの改変により提案手法を実現することで，既存のソフトウェアを改変なしに利用できる．これらの理由から，提案手法を VMM の改変により実現する．

4.3.3　重要サービスを提供するプロセスに関する情報の不可視化

プロセスの特定を困難にする手法は，以下の二つの機能からなる．

(1) プロセス情報へのアクセス制御

(2) プロセス情報の置換

プロセス情報へのアクセス制御について，図 4.1 に概略図を示す．この方式では，事前にプロセス情報へのアクセスを許可するカーネルテキスト部の領域を設定しておき，また，プロセス情報が存在するページの読み込みを禁止しておく．読み込みを禁止したページへのアクセス違反が発生した際に，アクセス元の命令が配置されているアドレスに応じて返却する

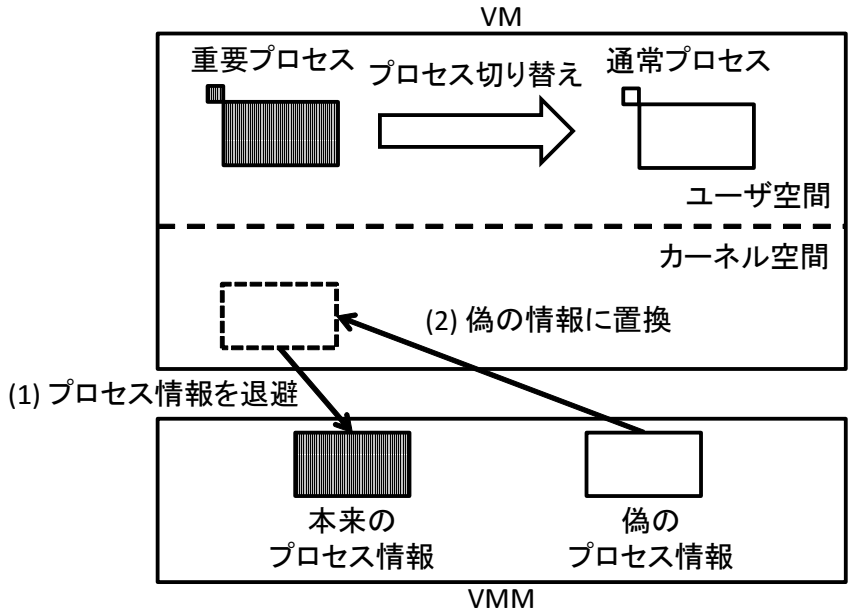

図 4.2 重要プロセスから通常プロセスへの切り替えにおけるプロセス情報の置換

値を変更し，アクセスを許可するカーネルテキスト部以外からのアクセスに対しては，偽の情報を返却する．これにより，本来のプロセス情報を攻撃者から不可視化できる．

プロセス情報の置換では，重要プロセスの走行中は，本来のプロセス情報を利用し，重要プロセス以外のプロセス（以降，通常プロセス）の走行中は，偽の情報に置換する．これにより，重要プロセスの走行中以外は，攻撃者からは，偽のプロセス情報しか参照できない．重要プロセスから通常プロセスへのプロセス切り替えが発生すると，図 4.2 に示すように，重要プロセスのプロセス情報を退避し，偽の情報に置換する．また，通常プロセスから重要プロセスへのプロセス切り替えが発生すると，図 4.3 に示すように，重要プロセスのプロセス情報を復元する．このため，重要プロセスの走行を妨ることなく，攻撃者には偽のプロセス情報を見せることで，攻撃対象プロセスの特定を困難にする．プロセス情報の置換については，4.4 節で詳述する．

4.3.4　不可視化対象プロセスの特定方法と対処

不可視化対象プロセスの特定方法

攻撃者は，提案手法の存在を検知し，不可視化対象のプロセスを特定できた場合には，そのプロセスを停止または無力化できる．このため，あるプロセスについて，そのプロセスが

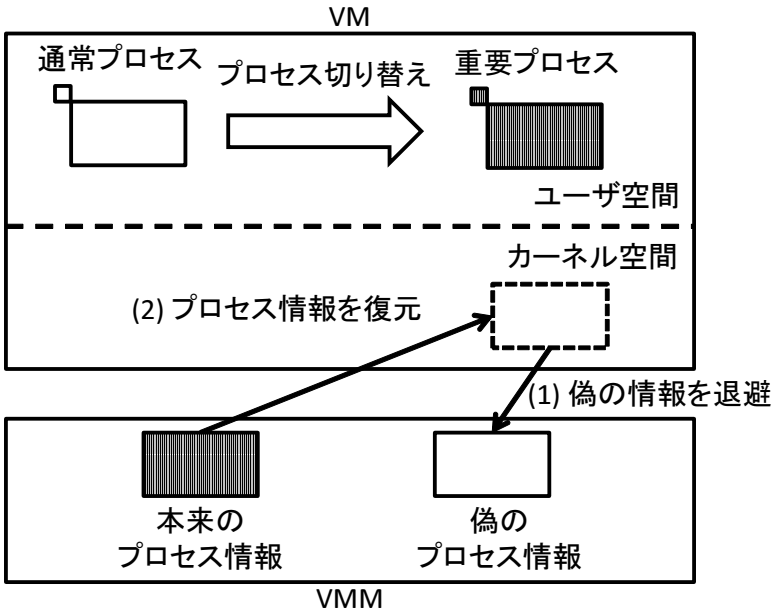

図 4.3 通常プロセスから重要プロセスへの切り替えにおけるプロセス情報の置換

不可視化対象か否かの攻撃者による判定を困難にする必要がある．

VM 上の AP やカーネルから不可視化対象プロセスを特定する方法として以下がある．

(1) プロセス切り替えの処理時間の差異の検知

(2) プロセス情報の継続的な監視

不可視化対象プロセスは，プロセス切り替え発生時に，提案手法によりプロセス情報の不可視化処理が実施される．このため，プロセス切り替えの処理時間が他のプロセスよりも長い．

また，攻撃者がカーネルレベルで動作するソフトウェアを VM 上で動作させていた場合，攻撃者は，あるプロセスのプロセス情報を継続的に監視し，プロセス切り替えの度に検査することで，不可視化対象プロセスか否かを検出できる．不可視化対象プロセスはプロセス切り替えの度にプロセス情報が置換される一方，他のプロセスは走行状態などを示す変数以外は変化しない．このため，プロセス切り替え時に本来変化しない変数が変化していた場合，攻撃者は，そのプロセスを不可視化対象プロセスと特定できる．

不可視化対象プロセスの特定方法への対処

(1) の方法による検知には，プロセス切り替え時に不可視化対象プロセスに対して行っている処理と同等の処理を通常プロセスに対しても実施することで対処できる．ただし，すべてのプロセスのプロセス切り替えにおいて性能低下が生じるため，システム全体の性能は低下する．

または，文献 [86] でマルウェア解析機能の一部として提案されているタイムコントロール機能の利用が考えられる．この機能は，特定の関数の実行にかかる時間をあらかじめ計測しておき，その関数実行時に，予め計測した時間との実実行時間の差異によりデバッガの存在を検知し，動作を停止するマルウェアを対象としている．タイムコントロール機能は，解析対象の実行を一時停止させる場合に，ゲスト OS 内の CPU クロックを完全に停止し，すべての仮想ハードウェアの動作を一時停止させる．この機能を応用し，提案方式においてゲスト OS から VMM に処理が移行した際に，ゲスト OS 内の CPU クロックを停止させることで，提案手法によるプロセス情報の置換処理にかかるオーバヘッドを隠ぺいできる．これにより，提案手法の導入によるオーバヘッドをもとにした不可視化対象プロセスの検知を回避できる．

(2) の方法による検知には，プロセス情報へのアクセス制御とプロセス情報の置換を連携させることで対処できる．ここでは，ゲスト OS として Linux を想定し，攻撃者が Loadable Kenrel Module (以降，LKM）を用いてプロセス情報の継続的な監視を行なっているものとする．まず，準備として，重要プロセスのプロセス情報のカーネルからの読み込みを禁止し，カーネルモジュールを含まないカーネルコードの仮想アドレス範囲を決定しておく．その後，重要プロセスのプロセス情報へのアクセス違反により VMM へ処理が移行した際に，図 4.4 に示す手順に従い，メモリアクセスを許可するか否かを決定する．まず，VM 上の命令ポインタが事前に決定した範囲か否かを判定する．範囲外であった場合は，偽のプロセス情報を返却する．範囲内であった場合は，カーネルスタックを遡り，カーネル内関数の呼び出し元の関数すべての仮想アドレスを取得し，そのすべてが決定した範囲内であった場合のみ，メモリアクセスをエミュレートする．そうでない場合は，プロセス情報を置換する．

上記のアクセス制御モデルは，ゲスト OS のカーネルの完全性に依存している．このため，カーネルコードを上書きする攻撃を考慮する必要がある．カーネルのテキスト領域を上書きする攻撃の例として，DKSM[87] がある．DKSM では，カーネルのテキスト領域を上書きするために，CR0 レジスタを操作する．カーネルのテキスト領域は書き込みが禁止されているため，攻撃者は，カーネルのテキスト領域の書き込み禁止を無効化するために，CR0 レジスタを操作する．しかし，Intel VT-x により仮想化された環境では，CR0 レジスタへのアクセ

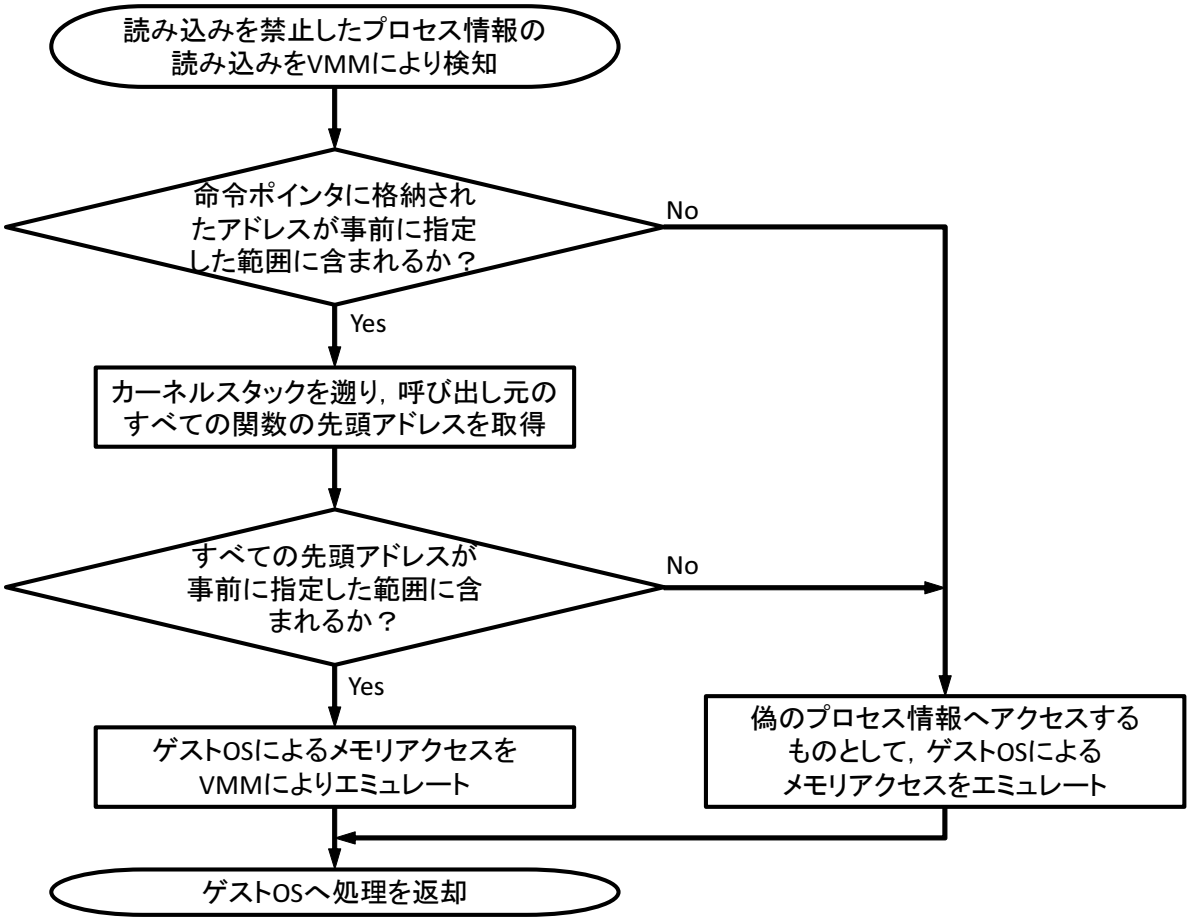

図 4.4 重要プロセスのプロセス情報へのアクセスを許可するか否かの判定

スにより VM exit が発生し，VMM へ処理が遷移する．このため，VMM は，カーネルテキスト領域への書き込みを目的とした CR0 レジスタへのアクセスを検知できる．このように，CR0 レジスタへの書き込みを VMM により監視することで，上記のアクセス制御モデルを適用できる．

以上の手順により, 事前に決定した範囲外からのプロセス情報の読み込み時には偽のプロセス情報を返却することで，継続的なプロセス情報の監視による不可視化対象プロセスの特定を回避できる．

4.3.5　提案手法の構成

以上の対処を実現した場合の提案手法の全体像を図 4.5 に示す．提案手法は，プロセス情報へのアクセス制御とプロセス情報の置換を行うプロセス情報制御部を VMM 内に追加する

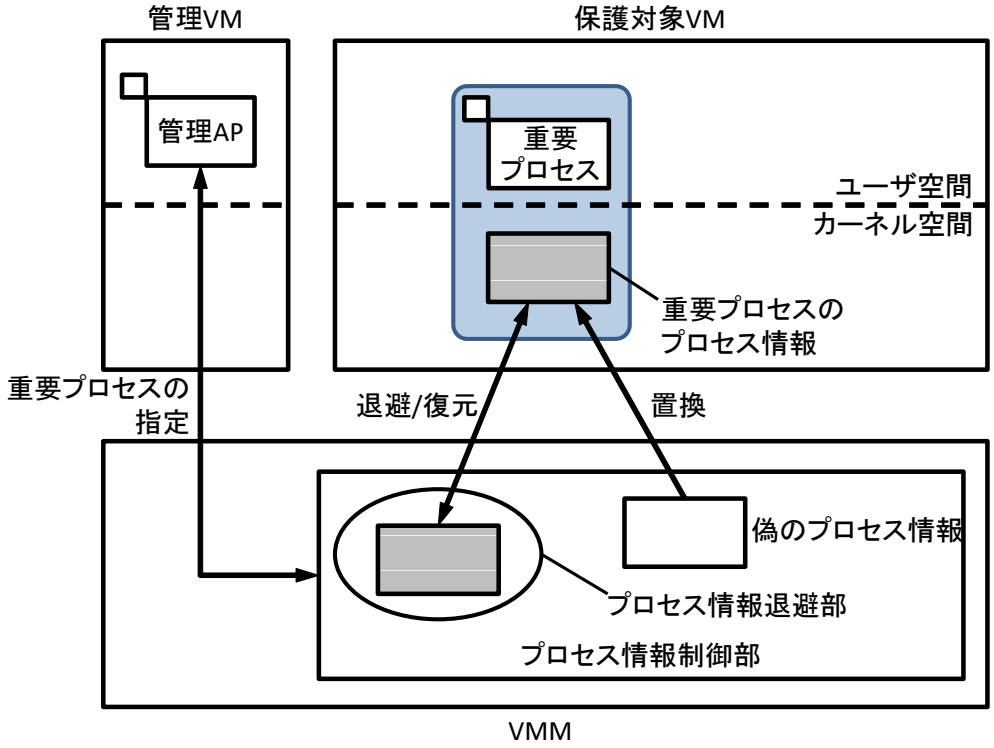

図　4.5　提案手法の全体像

ことで実現する．プロセス情報制御部は，保護対象 VM 上で発生するプロセス切り替えを監視し，プロセス切り替えが発生した際には，重要プロセスのプロセス情報の退避，偽の情報への置換，および本来の情報への復元を行う．プロセス情報は，VMM 内に確保したプロセス情報退避部に退避する．プロセス情報退避部は，保護対象 VM ごとに VMM 内に確保し，管理する．偽のプロセス情報は，あらかじめ複数種類を用意しておき，プロセス情報の置換の際にどの情報を用いるかを決定する．

提案手法において，保護対象 VM 上のどのプロセスを重要プロセスとするかは，管理 VM で動作する制御 AP により指定する．このため，提案手法を利用する場合には，保護対象 VM の管理者は，事前に，どのプロセスを重要プロセスとするかを VMM の管理者に連絡し，VMM の管理者が管理 AP を用いて保護対象プロセスを VMM に通知し，指定する．

4.3.6　提案手法の限界

提案手法は，重要プロセスの攻撃者による特定を困難にするのみであり，プロセス自体を不可視にはしない．このため，偽のプロセス情報に置換した状態のプロセスは，攻撃者から

認識できる．このことから，攻撃者が無作為にプロセスを停止させた場合には，重要プロセスを停止される可能性がある．

重要プロセスへの攻撃を防止するには，4.3.3 項で述べたプロセス情報へのアクセス制御が有用である．ただし，この攻撃への対処は，重要サービスへの攻撃の回避という本研究の目的とは異なる．このため，本章では，詳細な検討はしない．

また，本手法では，正規のプロセス情報を偽のプロセス情報に置換するため，保護対象 VM の管理者は，重要プロセスを制御できない．ただし，重要プロセスとして動作させるプログラムは，セキュリティソフトウェアなどの，常駐型のプログラムである．このため，本手法では，常駐型以外のプログラムで，利用者から頻繁に実行されるようなプログラムを重要プロセスとして走行させることは想定していない．

4.4　プロセス情報の置換手法

4.4.1　置換対象のプロセス情報

プロセス情報の定義

ゲスト OS として x86 または x64 アーキテクチャを対象とした Linux を用いた場合，プロセス情報として以下がある．

(1) プロセス制御ブロック

(2) カーネルスタックと `thread_info` 構造体

(3) ハードウェアコンテキスト

(4) ページテーブル

(5) プロセスの利用しているメモリ

プロセスを完全に不可視化するには，上記のプロセス情報をすべて不可視化する必要がある．しかし，(3)，(4)，および（5）の情報をもとにプロセスが何かを特定するのは困難である．一方，(1）と（2）の情報は，プロセスを特定するために有用な情報を多く含む．このため，本章における実現方式では，上記のプロセス情報のうち，(1）と（2）の二つの情報を扱う．

以下にそれぞれのプロセス情報の詳細を示す．

(1) プロセス制御ブロック (task_struct 構造体)
プロセス制御ブロックには，プロセス ID（PID），スレッドグループ ID（TGID），プロセスの実行ファイル名，および親プロセスの PID など，プロセスの特定に利用できる情報が多く含まれている．Linux においては，プロセス制御ブロックは，task_struct 構造体として定義されており，プロセスやスレッドごとに作成される．

(2) カーネルスタックと thread_info 構造体
カーネルスタックと thread_info 構造体は，共用体としてカーネルにより確保されている．カーネルスタックは task_struct 構造体ごとに確保される．カーネルスタックには，カーネル内で呼び出したカーネル内関数のアドレス，引数，および戻り値などが格納されている．thread_info 構造体は，task_struct 構造体と相互にリンクされている．これらの情報はプロセスの特定に利用できる．

置換対象

プロセス情報として定義した情報は，プロセススケジューリングやシグナルの配送など，プロセスが走行中でない場合に利用される情報を含む．このため，プロセス情報のすべてを偽の情報に置換した場合，ゲスト OS におけるプロセススケジューリングやシグナルの配送を妨害する問題がある．これらのことから，置換対象とするプロセス情報を選択する基準として，プロセスの特定が可能か否か，またはプロセスの走行を妨害しないかの二つがある．

そこで，プロセス情報を置換する方針として，以下の 2 方針がある．

(置換方針 1) プロセスの動作に最低限必要な情報のみ本来の情報を設定し，その他のできるだけ多くの情報を置換

(置換方針 2) プロセスの動作を妨害しない情報のうち，攻撃者によるプロセスの特定を困難にできる情報のみを置換

(置換方針 1) は，(置換方針 2) よりもプロセスの特定が困難になる．しかし，多くの情報を置換する必要があるため，メモリコピーによる性能低下が大きい．一方，(置換方針 2) は，プロセスの特定に利用される少数の情報のみを置換するため，性能低下は小さい．ただし，(置換方針 2) は，実在するマルウェアがどのような情報から攻撃対象のプロセスを特定しているのかを調査する必要がある．

攻撃への対策の観点からは，(置換方針 1) が望ましい．しかし，実運用を考慮すると，できるだけ性能低下を抑えながら攻撃対象プロセスの特定をある程度困難にできる (置換方針

2) の方が実用的であると考えられる．このため，本章における実現では，(置換方針 2) を採用する．

攻撃対象プロセスの特定に利用される情報

マルウェアが攻撃対象プロセスの特定に利用する情報について考察する．Agobot[12] は，Windows を対象としており，走行中プロセスの一覧からプログラムの実行ファイル名を探索し，あらかじめ作成しておいたプログラム名の一覧に一致するエントリがあった場合，TerminateProcess 関数により，そのプロセスとプロセスが所属するすべてのスレッドを終了させる．t0rnkit[88] やその亜種である dica[89] は，Linux を対象としており，killall コマンドにより syslogd を停止させる．killall コマンドは，proc ファイルシステムを利用し，走行中のプロセスのうちプログラム名に引数で渡された文字列を含むプロセスの PID を取得し，kill システムコールにより当該プロセスを終了させる．

この他にも，プロセスを停止させるマルウェアは多く存在するものの，その多くは，プログラム名をもとにプロセスを終了させる．このため，プロセス情報の置換において，プロセスのプログラム名の置換が効果的である．

偽のプロセス情報

本章における実現方式では Linux を対象としている．上で述べたように，Linux を対象としたルートキットである dica は killall コマンドによりプロセスを停止させる．このため，Linux におけるプロセス情報のうち，プロセスのプログラム名が格納されている情報を置換する．

killall コマンドがプログラム名を取得する際には，`/proc/[PID]/stat` を読み込む．ここで，`[PID]` には，走行中のプロセスの PID が割り当てられる．`/proc/[PID]/stat` をプロセスが参照すると，カーネルは，プロセスが用意したバッファに，プロセスに関する情報をスペース区切りで格納する．この情報には，`task_struct` 構造体のうち，`pid`，`comm`，および `state` などが含まれる．`comm` メンバには，プロセスのプログラム名が格納されている．このため，`comm` メンバを置換することで，プログラム名をもとにプロセスを停止する攻撃を回避できる．

プログラム名を置換する場合に，固定の文字列に置換すると，攻撃者は，攻撃対象とするプログラム名に文字列を追加するだけで対処できる．このため，プログラム名を置換する場合は，他のプロセスから実行されることの多いプログラムのプログラム名や複数走行するこ

との多いプロセスのプログラム名に置換する．他のプログラムから実行されることの多いプログラム名に置換した場合，攻撃者がそのプロセスを停止すると，他のプロセスの動作に問題が生じ，システム管理者が異変に気付きやすくなる．また，複数走行することの多いプロセスのプログラム名に置換した場合，同時に走行するプロセス数が増加しても，そのプロセスが本来の動作として複数走行しているのか提案手法により複数走行しているように見えるのかを区別できないため，攻撃対象の特定が難しい．このようなプログラムの例として，udevd や apache2 がある．

4.4.2　プロセス情報を置換する契機

プロセス情報の置換では，プロセス切り替えの発生時に，切り替え元プロセスと切り替え先プロセスのそれぞれが不可視化対象のプロセスか否かを判断し，プロセス情報を置換する．このためには，VMM からゲスト OS におけるプロセス切り替えを検知する必要がある．

完全仮想化環境では，ゲスト OS は VMX non-root モードで，VMM は VMX root モードで動作する．VMX non-root モードで許可されていない操作を行うと，VM exit が発生し，VMX root モードで動作するソフトウェアに処理が移る．許可されていない操作の中には，CR3 レジスタへの書き込みがある．多重仮想記憶を利用している OS では，プロセス切り替え時にメモリ空間を切り替えるために，ページディレクトリの先頭アドレスが格納されている CR3 レジスタを書き換える．このため，CR3 レジスタへの書き込みによる VM exit を契機とすることで，プロセス切り替えを VMM により検知できる．

4.4.3　ゲスト OS のプロセス情報の取得

切り替え元プロセスのプロセス情報の取得

VM exit 発生時に VMM から参照できるのは，主に，VM の利用しているレジスタである．ゲスト OS のプロセス情報である `thread_info` 構造体と `task_struct` 構造体は，図 4.6 に示すように，カーネルスタックのスタックポインタである rsp レジスタより計算できる．カーネルスタックの最大サイズはカーネルコンパイル時に確定するため，カーネルスタックの最大サイズをもとに，`thread_info` 構造体の先頭アドレスを rsp レジスタより取得できる．また，`thread_info` 構造体の `task` メンバは，そのプロセスの `task_struct` 構造体の先頭アドレスを保持する．これにより，`task_struct` 構造体を取得できる．以上のことより，VMM は，VM の rsp レジスタの値をもとに，ゲスト OS のプロセス情報を取得できる．ただし，VMM

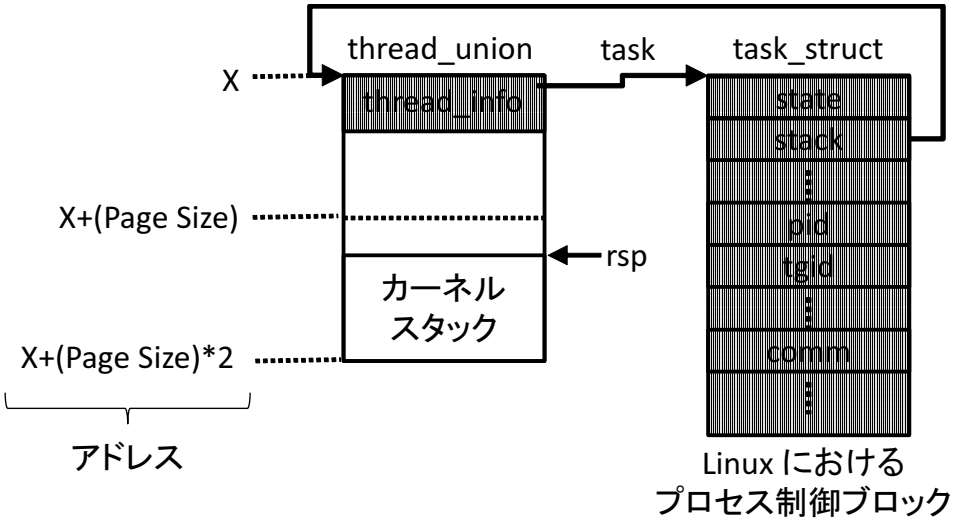

図 4.6 カーネルスタックとプロセス情報の関係

は，事前にゲスト OS の thread_info 構造体と task_struct 構造体の定義を保持しておく必要がある．

切り替え先プロセスのプロセス情報の取得

上で述べた方法は，プロセス情報の取得に rsp レジスタを用いているため，走行中プロセスのプロセス情報の取得にしか利用できない．このため，切り替え先プロセスのプロセス情報の取得方法を検討する必要がある．

取得方法

プロセス切り替え発生時に VMM から取得できる VM の情報のうち，切り替え先プロセスの特定に利用できる情報として，CR3 レジスタへ書き込まれようとしている値がある．この情報をもとに切り替え先プロセスのプロセス情報を取得する方法として，以下の方式が考えられる．

(走査方式) ゲスト OS 上のプロセスリストを走査する方式

(リスト管理方式) 可視化対象のプロセスの CR3 レジスタの値と task_struct 構造体の先頭アドレスの組を VMM 内に保持する方式

(契機追加方式) 切り替え先プロセスを特定できるような箇所に VM exit を発生させる契機を追加する方法

(走査方式) は，プロセス切り替え時の CR3 レジスタへのアクセスによる VM exit を検知し，ゲスト OS のプロセスリストを走査する方式である．Linux において，すべての `task_struct` 構造体は，双方向環状のプロセスリストで連結されている．プロセスリストの先頭は，カーネルスレッドである `init_task` であり，カーネルの終了まで開放されない．また，プロセスとページディレクトリの対応は，`task_struct` 構造体の `mm` メンバを参照することで確認できる．このため，`init_task` から順にプロセスリストを走査し，VM exit 発生時に CR3 レジスタに書き込まれようとしている値をゲスト OS のプロセスリストから探索することで，切り替え先プロセスを特定できる．

(リスト管理方式) は，プロセス切り替え時に切り替え元プロセスが不可視化対象プロセスか否かを常に確認しておき，不可視化対象プロセスであった場合には，CR3 レジスタの値を VMM 内に確保した領域に保存する．また，CR3 レジスタの値だけでなく，`task_struct` 構造体の先頭アドレスも保存する．これにより，切り替え先プロセスの CR3 の値を取得した後に，VMM 内に保存されているものと比較することで，切り替え先プロセスが不可視化対象プロセスか否かを識別できる．また，`task_struct` 構造体の先頭アドレスと組にして保存しておくことで，切り替え先プロセスのプロセス名や PID をゲスト OS が利用しているメモリから即座に取得できる．不可視化対象プロセスが複数存在する場合には，これらの要素をメンバに持つ構造体のリストを作成する．これにより，CR3 へ書き込まれようとしている値を参照することで切り替え先プロセスを特定できる．

(契機追加方式) は，切り替え先プロセスを特定できるような契機を新たに追加する．たとえば，プロセス切り替えを行う際に切り替え元プロセスと切り替え先プロセスの両方の `task_struct` 構造体の先頭アドレスを引数にとる関数にブレークポイントを設定して VM exit を発生させる方法が考えられる．この場合は，引数の情報をもとに切り替え先プロセスを特定できる．ブレークポイントの設定には，ゲスト OS のメモリを書き換えて `INT3` 命令を埋め込む方法やハードウェアブレークポイントを設定する方法を利用できる．既存の VM exit を用いるのではなく，追加の VM exit を発生させるという点で上記の 2 方式と異なる．

取得方法の比較

それぞれの方式の利点と欠点を表 4.1 に示す．(走査方式) は，VM exit 発生時にゲスト OS のメモリを解析するだけで実現できる．このため，(リスト管理方式) のようにゲスト OS 上

表 4.1 切り替え先プロセスの特定方法の比較

方式	利点	欠点
(走査方式)	ゲスト OS のメモリの解析のみで実現可能	ゲスト OS のプロセスリスト探索の所要時間大
(リスト管理方式)	VMM 内のリスト探索の所要時間小	VMM のメモリ使用量増加
(契機追加方式)	切り替え先プロセスを即座に特定可能であることから，性能低下小	設定できるブレークポイント数に制限有

のプロセスの状態を VMM 内で把握する必要や (契機追加方式) のようにゲスト OS にブレークポイントを追加する必要はない．しかし，プロセス切り替えによる VM exit が発生する度にゲスト OS のプロセスリストを走査する，このため，プロセスリストの長さを N とした場合，VMM は平均で $N/2$ 回プロセス情報を取得し，切り替え先プロセスか否かを判定する必要があり，性能低下が大きいと予測できる．これに比べ，(リスト管理方式) と (契機追加方式) は，十分に小さい性能低下で実現できる．(リスト管理方式) は，完全仮想化環境において必ず生じる VM exit のみを用いて切り替え先プロセスを特定できるため，追加の VM exit は発生せず，最も性能低下が小さい．(契機追加方式) は，切り替え先プロセスを特定するために追加で VM exit を発生させるため，(リスト管理方式) よりも性能低下が大きいと予測できる．(リスト管理方式) の欠点として VMM 内に重要プロセスのリストを作成するため，メモリ使用量が増加する問題がある．しかし，重要プロセスとして想定するプロセスは，ウイルス対策ソフトウェアやシステム管理ツールであることから，不可視化対象となるプロセスは少数である．このことから，増加するメモリ使用量は，VMM 内のリストの 1 エントリのサイズが数 10 バイトだとしても，100 バイト程度であると予測できる．代表的な VMM の一つである Xen のメモリ使用量は，Xen 4.2.0 において 182 メガバイトであることから，100 バイトの増加は十分小さい．以上のことから，切り替え先プロセスの特定には，(リスト管理方式) を用いる．

4.4.4　重要プロセスの指定

提案手法では，プロセス情報を置換する前に，管理 AP から重要プロセスを指定する必要がある．提案手法では，保護対象 VM から VMM へ重要プロセスを通知するインタフェースを持たないため，保護対象 VM の管理者は，事前に，管理 VM の管理者に重要プロセスを通知する必要がある．保護対象 VM か VMM へ重要プロセスを通知するインタフェースを持たないのは，VM から VMM へ直接処理依頼をするインタフェースを追加すると，VMM の脆弱性の増加に繋がるためである．このため，重要プロセスの起動前に，起動するプログラムの実行ファイルのフルパスやコマンド名を管理 VM の管理者に通知しておく必要がある．管理 VM の管理者は，通知を受けた情報をもとに，管理 AP を用いて VMM に通知する．管理 AP から VMM への通知には，Xen におけるハイパーコールを用いる．プロセス情報制御部は，通知を受けた情報をもとにプロセス切り替えを監視し，重要プロセスを発見すると，プロセス情報を偽のプロセス情報に置換する．以上の手順により，保護対象 VM の管理者は重要プロセスを指定する．

4.4.5　マルチコアプロセッサへの対応

マルチコア CPU を用いる環境を想定した場合，VM に複数の物理 CPU コアが割り当てられているとすると，あるコアで重要プロセスの走行中に，別のコアで他のプロセスが同時に走行する状況が考えられる．この場合，重要プロセスのプロセス情報は復元されており，本来の情報を利用している．このため，別のコアのプロセスやカーネルから重要プロセスのプロセス情報を参照できてしまう．

この問題へ対処するために，重要プロセスの走行中は，通常プロセスが並列に走行することを禁止する．重要プロセスの走行中は，重要プロセスが走行している CPU コア以外のコアを停止する．これは，VM に提供している仮想 CPU を VMM により一時的に停止することで実現する．これにより，通常プロセスによる保護対象プロセスのプロセス情報の参照を制限する．

表 4.2 評価環境

VMM	Xen 4.2.0
OS（管理 VM）	Debian 7.3 (Linux 3.2.0 64-bit)
OS（保護対象 VM）	Debian 7.3 (Linux 3.2.0 64-bit)

4.5 評価

4.5.1 評価環境

評価環境を表 4.2 に示す．CPU には，Intel Core i7-2600 を用いた．保護対象 VM は，Intel VT-x の機能を用いて Xen により完全仮想化した．

4.5.2 評価の目的と評価方法

評価の目的は，攻撃によるプロセスの停止を提案手法により回避できることの確認である．あるプログラムについてプロセス情報を置換した際に，プログラム名をもとにしたそのプロセスの停止を回避できるか否かを確認する．

攻撃によるプロセスの停止として，killall コマンドの利用を想定する．killall コマンドは，プロセスのプログラム名を proc ファイルシステムから取得する．このため，本評価では，提案手法により不可視化対象プロセスのプログラム名を置換した場合に，本来のプログラム名が ps コマンドにより表示されないことを確認する．ps コマンドは，`task_struct` 構造体の `comm` メンバを参照する．proc ファイルシステムにおいてプログラム名を取得する際にも，同様に `comm` メンバを参照するため，ps コマンドにより，proc ファイルシステムで取得できるプログラム名と同じ名前を取得できる．

4.5.3 評価結果

提案手法により rsyslogd のプログラム名を変更した．ここでは，実験として，rsyslogd の名前を apache2 に変更した．これは，ディストリビューションの標準の設定でインストールした場合，apache2 が複数プロセスで走行するためである．本来一つしか走行しないプロセスが複数走行していた場合は，攻撃者は，容易に異常を検知し，攻撃対象プロセスを特定できる．そこで，標準の設定で複数プロセスで走行するプロセスのプロセス名を rsyslogd の偽のプロセス名として設定した．

保護対象 VM において ps コマンドの利用によりすべてのプロセスのプログラム名を確認した．その結果，プロセスの一覧に rsyslogd は表示されず，rsyslogd のプロセスのプログラム名として apache2 という文字列が表示された．このことから，プロセスのプログラム名を置換できていることを確認できた．

4.6　まとめ

本章では，VMM により VM 上のプロセス情報を置換することでプロセスの特定を困難にする攻撃回避手法について述べた．攻撃者は，攻撃対象のプロセスをプロセス情報をもとに特定することから，プロセス情報を偽の情報に置換することで本来のプロセス情報を不可視化し，攻撃対象のプロセスの特定を困難にすることで攻撃を回避する．本手法は，VM 上の OS や AP の改変なしに実現する．プロトタイプを用いた実験では，攻撃対象のプロセスの特定に多く利用されるプロセス名を異なるプロセス名に置換し，他のプロセスからは偽のプロセス名しか参照できないことを確認した．

第 5 章

結論

5.1 成果

第 1 章では，本研究の背景と課題について述べた．攻撃の高度化や未知の攻撃への対処の困難さから，攻撃者による侵入や情報の取得が行われることを前提としたセキュリティ機構が必要である．その中でも，ログ保全と攻撃難化が重要であり，将来的に仮想化技術の利用を前提としたソフトウェア構成が一般的になることが予想されるため，ログ保全と攻撃難化のために，仮想化技術を利用することが有効であることを述べた．以上のことから，課題として，ログの改ざんや消失の防止，ログ保全処理のオーバヘッド削減，およびプロセス情報の不可視化によるプロセス特定の困難化を課題として述べた．

第 2 章では，監視対象 OS のログの改ざんや消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．計算機の動作を把握するためには，ログが重要である．しかし，攻撃者により，攻撃の痕跡を消すために，ログを改ざんされる可能性がある．また，Linux におけるカーネルログは，大量のログ出力により古いログが消失する問題がある．これらの問題へ対処するために，VMM によりログの改ざんや消失を防止するシステムを提案した．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他の VM 上に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 のログ送信要求終了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消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．また，VMM が取得したログを取得対象とは異なる VM に隔離して保存する．これにより，取得したログの安全性を高めている．さらに，隔離したログと取得対象の VM 上のログを比較することで，ログの改ざん箇所を検出できる．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，実現した機構について，ログの改ざんや消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消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消失する問題について，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，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 2–5μs 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，測定した処理のオーバヘッドは，約 5–50μs であり，その影響は小さいことを示した．

第 3 章では，VM 外部への低オーバヘッドなログ転送手法について述べた．VMM を用いることで，VM 上のログを取得し，安全に保存する方式は第 2 章で述べた．しかし，この方式は，AP によるログ出力を VMM により監視する必要があり，性能低下が問題となる．そこで，VM 上のライブラリにログの転送契機を VMM へ通知する機構を追加することで，VM 外部への低オーバヘッドなログ転送手法を提案した．本手法は，VM 上の AP がログを発行する際に，カーネルに到達する前に VMM へログの転送を依頼することで，VM 外部へログを転送する．このため，VM 上のカーネルレベルで動作するマルウェアがログを改ざんしたとしても，VM 外部に転送したログに影響しない．また，本手法を VMM により実現することで，VM 上のマルウェアから攻撃の困難な機構を実現した．さらに，本手法は，OS の構造に依存せず，ライブラリの置き換えによりログの VM 外部への転送を実現することで，多種の OS へ容易に適用できる．VMM により VM 上の AP や OS の動作を監視するのではなく，置き換えたライブラリから VMM へログの転送を依頼するため，VM 上の AP や OS を監視する処理が不要となり，監視により生じるオーバヘッドを削減し，低オーバヘッドなログの転送を実現した．

評価結果より，本手法でログを転送するよりも前にログを改ざんするのは難しいことを示した．また，多種の OS への適用では，ライブラリに 20 行のコードを追加するだけで適用できることから，適用の容易さを示した．性能評価では，本手法の導入による syslog 関数の性能低下は 6%であることを示した．また，データベース管理システムを用いた評価では，性能低下は 1%未満であることを示した．このことから，入出力処理の負荷が高い AP において

は，本手法の適用による性能低下は非常に小さいことを示した．複数 VM が走行する環境における性能評価では，VM 数の増加による性能低下への本手法の影響は小さいことを示した．

第 4 章では，VMM により VM 上のプロセス情報を置換することでプロセスの特定を困難にする攻撃回避手法について述べた．セキュリティソフトウェアは，攻撃者により無効化される可能性がある．セキュリティソフトウェアが停止または無効化されると，攻撃の検知が遅れ，被害が拡大する可能性がある．攻撃者は，攻撃対象のプロセスをプロセス情報をもとに特定する．このことから，プロセス情報へのアクセス制御とプロセス情報の置換について検討した．このうち，プロセス情報の置換について，プロセス情報を偽の情報に置換することにより，本来のプロセス情報を不可視化し，攻撃対象のプロセスの特定を困難にすることで攻撃を回避する手法の設計と VMM を用いた実現方式について述べた．

プロセス情報へのアクセス制御では，プロセス情報へアクセス可能なコードを事前に設定しておき，それ以外のコードによるプロセス情報の読み込みを禁止する．プロセス情報へのアクセス元が事前に設定したコードの場合，本来の情報を返却し，そうでない場合は偽の情報を返却することで，事前に設定したコード以外からは，本来のプロセス情報にアクセスできない．また，プロセス情報の置換では，セキュリティソフトウェアなどの重要プロセスについて，走行中は本来のプロセス情報を利用し，他のプロセスの走行中は偽の情報に置換する．これにより，重要プロセスの走行中以外は，重要プロセスのプロセス情報をマルウェアに参照されても，偽の情報を攻撃者に見せる．

プロセス情報へのアクセス制御は，プロセス情報を格納するメモリ領域へのアクセスで例外を発生させるため，性能低下が大きい．このため，第 4 章では，プロセス情報へのアクセス制御とプロセス情報の置換の設計について述べ，プロセス情報の置換手法の実現方式とプロトタイプを用いた評価について述べた．評価より，重要プロセスの走行中は，重要プロセスのプロセス情報は偽の情報に置換できていることを確認した．

以上より，将来的に更なる普及が予想される仮想化技術の利用により，攻撃の痕跡の削除を困難にし，かつセキュリティソフトウェアへの攻撃を回避することで，計算機利用環境の安全性を向上できることを示した．その具体例として，VM 上のソフトウェアが出力するログの保全と VM 上で動作するセキュリティソフトウェアへの攻撃難化を実現する手法を示した．

5.2　今後の課題

今後の課題として，以下が挙げられる．

(1) syslog を用いずに出力されたログの保全の検討
本論文で示したログの保全手法は，主に，Linux で広く利用されている syslog を用いたログ出力を対象としている．しかし，AP によっては，syslog を用いず，独自にログを出力するものがある．本論文で示したログの保全手法は，syslog を用いたログ出力を想定しているため，AP が独自に出力したログを保全できない．このため，本論文で示したログの保全手法を AP が独自に出力したログへ利用する手法を検討する必要がある．

(2) プロセス名以外に置換するプロセス情報の検討
プロセス情報の不可視化によるプロセス特定の困難化は，プロセス名を別の名前に置換することで，攻撃対象プロセスの特定を困難化し，攻撃を回避する．しかし，プロセス名のみの置換では，他のプロセス情報をもとに攻撃対象プロセスを特定される場合が考えられる．このため，プロセス名以外に，攻撃者から不可視化するプロセス情報を検討する必要がある．

(3) プログラムの設定ファイルへの攻撃回避手法の検討
プロセス情報の不可視化によるプロセス特定の困難化は，攻撃者がプロセス情報をもとに攻撃対象プロセスを特定することに着目している．しかし，攻撃者は，プロセス情報だけでなく，プログラムの設定ファイルをもとに攻撃対象プロセスの存在を検知する場合や，プログラムの設定ファイルを攻撃対象とする場合が考えられる．このため，プロセス情報の不可視化のみでは，プロセスへの攻撃回避は十分でない．このため，プログラムの設定ファイルへの攻撃回避手法を検討する必要がある．

近年，スマートフォン等の携帯端末の高度化や通信回線の高速化により，計算機は，利用者にとってより身近なものとなっている．したがって，今後，これまで以上に，計算機の扱う情報の増加と多様化が予想される．計算機や計算機の扱う情報が利用者にとって身近になることで，サイバー攻撃が利用者に与える影響は，より大きくなる．このため，計算機が利用者に身近になり，計算機の扱う情報の増加と多様化が進むにつれ，情報セキュリティの重要性が増加していくと考えられる．また，情報システムの更なる普及により，計算機資源を有効に利用するための仮想化技術の利用は，より一般的になることが予想される．このような環境において，仮想化技術の利用を前提とした情報セキュリティ技術が必要となる．本研究は，仮想化技術の利用を前提とした環境において，利用者が計算機を安全に利用するための基盤技術の構築に貢献するものである．このため，仮想化技術の更なる発展と普及が期待される将来においても，本研究は，情報セキュリティ向上における有効性を期待できる．

謝辞

本論文をまとめるにあたり，ご査読いただき，かつご助言を賜りました岡山大学 大学院自然科学研究科 産業創成工学専攻 山内利宏 准教授，谷口秀夫 教授，名古屋彰 教授に深く感謝いたします．

特に，山内利宏 准教授には，本研究を進めるにあたり，非常に丁寧なご指導を頂いたのみならず，岡山大学 工学部 情報工学科四年生から今日に至るまで，約五年半もの間お世話になり，研究はもちろんのこと，様々なことをご指導いただきました．心より感謝いたします．

また，谷口秀夫 教授には，研究に関する貴重なご助言だけでなく，様々なことをご指導いただきました．深く感謝いたします．

さらに，岡山大学 大学院自然科学研究科 産業創成工学専攻 乃村能成 准教授，ならびに後藤佑介 助教（現在，准教授）には，研究に関する貴重なご助言を頂きました．深く感謝いたします．

本論文は，岡山大学における約五年半の研究成果をまとめたものです．研究のみならず，様々なことにご協力いただきました山内研究室，谷口研究室，および乃村研究室の学生の皆様に感謝いたします．

最後に，常に支えてくれた家族に心より感謝いたします．

参考文献

[1] 総務省：電子政府，入手先 ⟨http://www.soumu.go.jp/main_sosiki/gyoukan/kanri/a_01.htm⟩（参照 2014-06-14）.

[2] 総務省:平成 17 年版 情報通信白書，入手先 ⟨http://www.soumu.go.jp/johotsusintokei/whitepaper/h17.html⟩（参照 2014-06-14）.

[3] 情報処理推進機構:「2013 年版 10 大脅威　身近に忍び寄る脅威」を公開，入手先 ⟨https://www.ipa.go.jp/security/vuln/10threats2013.html⟩（参照 2014-06-10）.

[4] McAfee, Center for Strategic and International Studies: The Economic Impact of Cybercrime and Cyber Espionage, http://www.mcafee.com/sg/resources/reports/rp-economic-impact-cybercrime.pdf (2013).

[5] 特定非営利活動法人デジタル・フォレンジック研究会：証拠保全ガイドライン第 3 版，http://www.digitalforensic.jp/20130930gijutsu.pdf?attredirects=0 (2013).

[6] Kent, K. and Souppaya, M.: Guide to Computer Security Log Management, Special Publication 800-92, http://csrc.nist.gov/publications/nistpubs/800-92/SP800-92.pdf (2006).

[7] 情報処理推進機構:組織における内部不正防止ガイドライン，入手先 ⟨http://www.ipa.go.jp/security/fy24/reports/insider/index.html⟩（参照 2014-06-12）.

[8] Accorsi, R.: Log Data as Digital Evidence: What Secure Logging Protocols Have to Offer?, *Proc. 33rd Annual IEEE International Computer Software and Applications Conference* (*COMPSAC '09*), Vol.2, pp.398–403 (2009).

[9] Saxena, M., kumar singh, N., Thakur, S.S. and kumar, P.: A Review of Computer forensic & Logging System, *International Journal of Advanced Research in Computer Science and Software Engineering*, Vol.2, No.1 (2012).

[10] Rutkowska, J.: Introducing stealth malware taxonomy, COSEINC Advanced Malware Labs (2006).

[11] Shields, T.: Survey of Rootkit Technologies and Their Impact on Digital Forensics, http://www.donkeyonawaffle.org/misc/txs-rootkits_and_digital_forensics.pdf (2008).

[12] F-Secure: Agobot, available from ⟨http://www.f-secure.com/v-descs/agobot.shtml⟩ (accessed 2014-06-08).

[13] Wheeler, A. and Mehta, N.: 0wning antivirus, http://www.blackhat.com/presentations/bh-europe-05/bh-eu-05-wheeler-mehta-up.pdf (2005).

[14] Xue, F.: Attacking Antivirus, http://blackhat.com/presentations/bh-europe-08/Feng-Xue/Presentation/bh-eu-08-xue.pdf (2008).

[15] Hsu, F.-H., Wu, M.-H., Tso, C.-K., Hsu, C.-H. and Chen, C.-W.: Antivirus Software Shield Against Antivirus Terminators, *IEEE Transactions on Information Forensics and Security*, Vol.7, No.5, pp.1439–1447 (2012).

[16] 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* (*SOSP '03*), pp.164–177 (2003).

[17] Kivity, A., Kamay, Y., Laor, D., Lublin, U. and Liguori, A.: kvm: the Linux virtual machine monitor, *Proc. Linux Symposium*, Vol.1, pp.225–230 (2007).

[18] VMware: vSphere Hypervisor, available from ⟨http://www.vmware.com/products/vsphere-hypervisor/⟩ (accessed 2014-06-16).

[19] VMware: VMware Workstation, available from ⟨http://www.vmware.com/products/workstation/⟩ (accessed 2014-06-16).

[20] Oracle: VirtualBox, available from ⟨https://www.virtualbox.org/⟩ (accessed 2014-06-16).

[21] Popek, G.J. and Goldberg, R.P.: Formal Requirements for Virtualizable Third Generation Architectures, *Comm. ACM*, Vol.17, No.7, pp.412–421 (1974).

[22] Corbet, J., Kroah-Hartman, G. and McPherson, A.: Linux Kernel Development: How Fast it is Going, Who is Doing It, What They are Doing, and Who is Sponsoring It, Linux Foundation Publications (2012).

[23] Xen Project, A Linux Foundation Collaborative Project: Xen Project Software Overview, available from ⟨http://wiki.xen.org/wiki/Xen_Project_Software_Overview⟩ (accessed 2014-06-20).

[24] Wojtczuk, R.: Subverting the Xen Hypervisor, ftp://document.eidologic.de/Linux/Xen/Xen_0wning_Trilogy/part1.pdf (2008).

[25] Rutkowska, J. and Wojtczuk, R.: Preventing and Detecting Xen Hypervisor Subversions, http://www.invisiblethingslab.com/resources/bh08/part2-full.pdf.

[26] Adiscon: Adiscon's rsyslog: The enhanced syslogd for Linux and Unix rsyslog, available from ⟨http://www.rsyslog.com/⟩ (accessed 2013-11-15).

[27] Balabit IT Security: The free software BalaBit: Syslog Server — syslog-ng Logging System, available from ⟨http://www.balabit.com/network-security/syslog-ng/⟩ (accessed 2013-11-15).

[28] Kelsey, J., Callas, J. and Clemm, A.: Signed syslog messages, available from ⟨http://tools.ietf.org/html/rfc5848⟩ (accessed 2013-11-12).

[29] New, D. and Rose, M.: Reliable delivery for syslog, available from ⟨http://www.ietf.org/rfc/rfc3195.txt⟩ (accessed 2013-11-12).

[30] 芦野佑樹，佐々木良一：セキュリティデバイスとヒステリシス署名を用いたデジタルフォレンジックシステムの提案と評価，情報処理学会論文誌，Vol.49, No.2, pp.999–1009 (2008).

[31] Schneier, B. and Kelsey, J.: Secure audit logs to support computer forensics, *ACM Transactions on Information and System Security (TISSEC)*, Vol.2, No.2, pp.159–176 (1999).

[32] Holt, J.E.: Logcrypt: Forward Security and Public Verification for Secure Audit Logs, *Proc. 4th Australasian Workshops on Grid Computing and e-Research*, Vol.54, pp.203–211 (2006).

[33] Ma, D. and Tsudik, G.: A new approach to secure logging, *ACM Transactions on Storage (TOS)*, Vol.5, No.1, pp.2:1–2:21 (2009).

[34] Takada, T. and Koike, H.: NIGELOG: Protecting Logging Information by Hiding Multiple Backups in Directories, *International Workshop on Database and Expert Systems Applications*, pp.874–878 (1999).

[35] Konishi, R., Amagai, Y., Sato, K., Hifumi, H., Kihara, S. and Moriai, S.: The Linux Implementation of a Log-structured File System, *SIGOPS Oper. Syst. Rev.*, Vol.40, No.3, pp.102–107 (2006).

[36] Microsoft: アクセス制御，入手先 ⟨http://technet.microsoft.com/ja-jp/library/cc732699(v=ws.10).aspx⟩ （参照 2014-07-02）.

[37] NSA: Security-Enhanced Linux, available from ⟨http://www.nsa.gov/research/selinux/⟩ (accessed 2014-07-02).

[38] 原田季栄，半田哲夫，橋本正樹，田中英彦：アプリケーションの実行状況に基づく強制アクセス制御方式，情報処理学会論文誌，Vol.53, No.9, pp.2130–2147 (2012).

[39] LIDS, available from ⟨http://www.lids.org/⟩ (accessed 2014-07-02).

[40] Cowan, C., Beattie, S., Kroah-Hartman, G., Pu, C., Wagle, P. and Gligor, V.D.: SubDomain: Parsimonious Server Security, *Proc. 14th Annual Large Installation Systems Administration Conference* (*LISA 2000*), pp.355–368 (2000).

[41] Microsoft: 暗号化ファイルシステム，入手先 ⟨http://technet.microsoft.com/ja-jp/library/cc749610(v=ws.10).aspx⟩ （参照 2014-07-02）.

[42] Microsoft: BitLocker Drive Encryption Overview, available from ⟨http://technet.microsoft.com/en-us/library/cc732774.aspx⟩ (accessed 2014-07-02).

[43] 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).

[44] Boeck, B., Huemer, D. and Tjoa, A.M.: Towards More Trustable Log Files for Digital Forensics by Means of "Trusted Computing", *Proc. International Conference on Advanced Information Networking and Applications*, pp.1020–1027 (2010).

[45] Garfinkel, T. and Rosenblum, M.: A Virtual Machine Introspection Based Architecture for Intrusion Detection, *Proc. Network and Distributed Systems Security (NDSS) Symposium*, pp.191–206 (2003).

[46] Sharif, M.I., Lee, W., Cui, W. and Lanzi, A.: Secure In-VM Monitoring Using Hardware Virtualization, *Proc. 16th ACM Conference on Computer and Communications Security* (*CCS '09*), pp.477–487 (2009).

[47] 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, *Proc. 5th Symposium on Operating Systems Design and Implementation* (*OSDI '02*), pp.211–224 (2002).

[48] Ibrahim, A., Hamlyn-Harris, J., Grundy, J. and Almorsy, M.: CloudSec: A Security Monitoring Appliance for Virtual Machines in the IaaS Cloud Model, *Proc. 5th International Conference on Network and System Security*, (*NSS '11*), pp.113–120 (2011).

[49] Isohara, T., Takemori, K., Miyake, Y., Qu, N. and Perrig, A.: LSM-Based Secure System Monitoring Using Kernel Protection Schemes, *Proc. International Conference on Availability, Reliability, and Security* (*ARES '10*), pp.591–596 (2010).

[50] Zhao, S., Chen, K. and Zheng, W.: Secure Logging for Auditable File System Using Separate Virtual Machines, *Proc. 2009 IEEE International Symposium on Parallel and Distributed Processing with Applications* (*ISPA09*), pp.153–160 (2009).

[51] Kim, G.H. and Spafford, E.H.: The Design and Implementation of Tripwire: A File System Integrity Checker, *Proc. 2nd ACM Conference on Computer and Communications Security* (*CCS '94*), pp.18–29 (1994).

[52] AIDE, available from ⟨`http://aide.sourceforge.net/`⟩ (accessed 2014-07-23).

[53] Hansen, S.E. and Atkins, E.T.: Automated System Monitoring and Notification with Swatch, *Proc. 7th Annual Large Installation System Administration Conference* (*LISA '93*), pp.145–152 (1993).

[54] Roesch, M.: Snort: Lightweight Intrusion Detection for Networks, *Proc. 13th Annual Large Installation Systems Administration Conference* (*LISA '99*), Vol.99, pp.229–238 (1999).

[55] Patil, S., Kashyap, A., Sivathanu, G. and Zadok, E.: I3FS: An In-Kernel Integrity Checker and Intrusion Detection File System, *Proc. 18th Annual Large Installation System Administration Conference* (*LISA '04*), Vol.4, pp.67–78 (2004).

[56] Kourai, K. and Chiba, S.: HyperSpector: Virtual Distributed Monitoring Environments for Secure Intrusion Detection, *Proc. 1st ACM/USENIX International Conference on Virtual Execution Environments* (*VEE '05*), pp.197–207 (2005).

[57] Joshi, A., King, S.T., Dunlap, G.W. and Chen, P.M.: Detecting Past and Present Intrusions Through Vulnerability-specific Predicates, *Proc. 12th ACM Symposium on Operating Systems Principles* (*SOSP '05*), pp.91–104 (2005).

[58] Jones, S.T., Arpaci-Dusseau, A.C. and Arpaci-Dusseau, R.H.: Antfarm: Tracking Processes in a Virtual Machine Environment, *Proc. USENIX 2006 Annual Technical Conference*, pp.1–14 (2006).

[59] Payne, B., Carbone, M., Sharif, M. and Lee, W.: Lares: An Architecture for Secure Active Monitoring Using Virtualization, *Proc. 2008 IEEE Symposium on Security and Privacy* (*SP '08*), pp.233–247 (2008).

[60] 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* (*CCS '08*), pp.51–62 (2008).

[61] Lanzi, A., Sharif, M.I. and Lee, W.: K-Tracer: A System for Extracting Kernel Malware Behavior, *Proc. Network and Distributed System Security (NDSS) Symposium* (2009).

[62] Jiang, X., Wang, X. and Xu, D.: Stealthy Malware Detection Through Vmm-based "Out-of-the-box" Semantic View Reconstruction, *Proc. 14th ACM Conference on Computer and Communications Security* (*CCS '07*), pp.128–138 (2007).

[63] Garfinkel, T., Pfaff, B., Chow, J., Rosenblum, M. and Boneh, D.: Terra: A Virtual Machine-based Platform for Trusted Computing, *Proc. 19th ACM Symposium on Operating Systems Principles* (*SOSP '03*), pp.193–206 (2003).

[64] Rutkowska, J. and Wojtczuk, R.: Qubes OS Architecture, Invisible Things Lab Tech Rep (2010).

[65] Xiong, X., Tian, D. and Liu, P.: Practical Protection of Kernel Integrity for Commodity OS from Untrusted Extensions, *Proc. Network and Distributed System Security (NDSS) Symposium* (2011).

[66] spoonfork: Analysis of a rootkit: Tuxkit, available from ⟨`http://www.ossec.net/doc/rootcheck/analysis-tuxkit.html`⟩ (accessed 2013-11-15).

[67] stealth: A new Adore root kit, available from ⟨`http://lwn.net/Articles/75990/`⟩ (accessed 2013-11-15).

[68] 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).

[69] 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* (*SOSP '07*), pp.335–350 (2007).

[70] Chow, J., Garfinkel, T. and Chen, P.M.: Decoupling dynamic program analysis from execution in virtual environments, *Proc. USENIX 2008 Annual Technical Conference*, pp.1–14 (2008).

[71] Riley, R., Jiang, X. and Xu, D.: Multi-Aspect Profiling of Kernel Rootkit Behavior, *Proc. 4th ACM European Conference on Computer Systems*, pp.47–60 (2009).

[72] Snow, K.Z., Krishnan, S., Monrose, F. and Provos, N.: SHELLOS: Enabling Fast Detection and Forensic Analysis of Code Injection Attacks, *Proc. USENIX Security Symposium* (2011).

[73] Hu, Y., Nanda, A. and Yang, Q.: Measurement, analysis and performance improvement of the Apache Web server, *Proc. 1999 IEEE International Performance, Computing and Communications Conference*, pp.261–267 (1999).

[74] 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).

[75] Kim, G.H. and Spafford, E.H.: The design and implementation of tripwire: a file system integrity checker, *Proc. 2nd ACM Conference on Computer and Communications Security*, pp.18–29 (1994).

[76] Neo: vanish.c, available from ⟨`http://packetstormsecurity.com/files/10486/vanish.c.html`⟩ (accessed 2014-07-17).

[77] 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).

[78] Wang, Z. and Jiang, X.: HyperSafe: A Lightweight Approach to Provide Lifetime Hypervisor Control-flow Integrity, *Proc. 2010 IEEE Symposium on Security and Privacy* (*SP '10*), pp.380–395 (2010).

[79] Wang, Z., Wu, C., Grace, M. and Jiang, X.: Isolating commodity hosted hypervisors with hyperlock, *Proc. 7th ACM European Conference on Computer Systems*, pp.127–140 (2012).

[80] Chen, P.M. and Noble, B.D.: When virtual is better than real, *Proc. 8th Workshop on Hot Topics in Operating Systems* (*HOTOS '01*), pp.133–138 (2001).

[81] 佐藤将也，山内利宏：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，情報処理学会論文誌，Vol.53, No.2, pp.847–856 (2012).

[82] Symantec: Backdoor.lastdoor, available from ⟨`http://www.symantec.com/security_response/writeup.jsp?docid=2002-090517-3251-99`⟩ (accessed 2013-11-12).

[83] Dewan, P., Durham, D., Khosravi, H., Long, M. and Nagabhushan, G.: A hypervisor-based system for protecting software runtime memory and persistent storage, *Proc. 2008 Spring simulation multiconference* (*SpringSim '08*), pp.828–835 (2008).

[84] Sato, M. and Yamauchi, T.: VMM-Based Log-Tampering and Loss Detection Scheme, *Journal of Internet Technology*, Vol.13, No.4, pp.655–666 (2012).

[85] McVoy, L.W., Staelin, C. et al.: lmbench: Portable Tools for Performance Analysis, *Proc. USENIX 1996 Annual Technical Conference*, pp.279–294 (1996).

[86] 川古谷裕平，岩村　誠，伊藤光恭：ステルスデバッガを利用したマルウェア解析手法の提案，マルウェア対策人材育成ワークショップ 2008 (MWS2008) (2008).

[87] Bahram, S., Jiang, X., Wang, Z., Grace, M., Li, J., Srinivasan, D., Rhee, J. and Xu, D.: DKSM: Subverting Virtual Machine Introspection for Fun and Profit, *Proc. 29th IEEE Symposium on Reliable Distributed Systems* (*SRDS 2010*), pp.82–91 (2010).

[88] F-Secure: Tornkit, available from ⟨`http://www.f-secure.com/v-descs/torn.shtml`⟩ (accessed 2014-06-08).

[89] Packetstorm: dica.tgz, available from ⟨`http://packetstormsecurity.com/files/26243/dica.tgz.html`⟩ (accessed 2014-06-08).