情報処理学会論文誌　Vol. 0　1–8 (??? 1959)

仮想マシンを用いた既存 IDS のオフロード

飯田貴大[†1]　光来健一[†1,†2]

侵入検知システム（IDS）を安全に実行できるようにするために、仮想マシンを用いて IDS をオフロードするという手法が提案されている。この手法は監視対象のシステムを仮想マシン上で動作させ、IDS だけ別の仮想マシン上で動作させる手法である。しかし、IDS をオフロードして動作させられるようにするためには多大な労力を必要とすることが多い。本稿ではオフロードした IDS に修正を加えることなく動作させられるようにする *Transcall* を提案する。Transcall はオフロード元の仮想マシンを監視するための実行環境である *VM* **シャドウ**を提供する。Transcall は VM シャドウ内のプロセスに対して、システムコールのエミュレーションを行い、**シャドウファイルシステム** を提供する。VM シャドウ内で IDS を動作させることで、オフロード元の仮想マシンを監視させることができる。

Offloading Existing IDSes Using Virtual Machines

Takahiro Iida [†1] and Kennichi Kourai[†1,†2]

To execute intrusion detection systems (IDSes) securely, offloading IDSes with virtual machines (VMs) has been proposed. This technique runs a monitored system on a VM and executes only IDSes on another VM. However, the proper execution of offloaded IDSes often requires great efforts. This paper proposes *Transcall*, which enables offloaded IDSes to be executed without any modification. Transcall provides an execution environment for monitoring a VM, called a *VM shadow*. Transcall emulates system calls and provides a *shadow filesystem* for processes in a VM shadow. Executing IDSes in a VM shadow enables monitoring the corresponding VM.

†1 九州工業大学
Kyushu Institute of Technology
†2 独立行政法人 科学技術振興機構, CREST
Japan Science and Technology Agency, CREST

1. はじめに

インターネットに接続されたサーバへの攻撃は年々増加しており、攻撃を検出するための侵入検知システム（IDS）[5),8),9)] の重要性が増している。IDS は攻撃の兆候を検出すると管理者に警告するシステムである。近年、IDS を安全に実行できるようにするために、仮想マシンを用いて IDS をオフロードするという手法が提案されている。[2),4)] この手法では監視対象のシステムを**サーバ** *VM* と呼ばれる仮想マシンを用いて動作させ、IDS だけを *IDS-VM* と呼ばれる別の仮想マシンで動作させる。これにより、侵入を検知する前に攻撃者によって IDS を無力化されてしまうことを防ぐことができる。

しかし、IDS を IDS-VM にオフロードして動作させられるようにするには多大な労力が必要とされることが多い。IDS は監視対象のシステムのプロセスに関する情報やファイルシステムに関する情報を取得して、攻撃の兆候の検出を行う。IDS を監視対象のサーバ VM から IDS-VM に移動させると、そのままでは IDS-VM 上で動いているプロセスや IDS-VM で使われているファイルシステムを監視することになってしまう。IDS-VM 上の IDS からサーバ VM を監視できるようにするには、仮想マシンモニタの機能を利用するように IDS 本体やそこから呼び出される外部コマンドに大幅な変更を加える必要があり、既存の IDS をそのまま使うことができないことが多かった。

この問題を解決するために、本稿ではオフロードした IDS に修正を加えることなく動作させられるようにする *Transcall* を提案する。Transall は IDS-VM に対して、サーバ VM を監視するための実行環境である *VM* **シャドウ**を提供する。IDS を VM シャドウの中で動作させることで、IDS 本体や外部コマンドは IDS-VM 上に置かれた既存のものをそのまま使いつつ、サーバ VM から情報を取得することが可能になる。Transcall は IDS がサーバ VM の OS の情報を取得できるようにシステムコールのエミュレーションを行う。また、IDS がサーバ VM のファイルシステムから情報を取得できるように、**シャドウファイルシステム**を提供する。サーバ VM への侵入者に OS カーネルを改ざんされない限りは正確な情報を取得できるようにするために、Transcall はサーバ VM のカーネルの内部構造を解析することによって必要な情報を取得する。

我々は Transcall を Xen[1)] のドメイン 0 上に実装した。Xen のドメイン U をサーバ VM とし、ドメイン 0 を IDS-VM として IDS のオフロードを実現した。仮想マシンモニタやドメイン 0 およびドメイン U の OS カーネルへの変更を行わないようにするために、Transcall によるシステムコールのエミュレーションは ptrace システムコールを用いて実装した。ま

た、シャドウファイルシステムの一部であるシャドウ proc ファイルシステムは FUSE[7] を用いて実装した。現在の実装では、VM シャドウ内で ps コマンドやファイルシステム関連のコマンドを実行することができている。Transcall を用いることで、サーバ VM で実行した ps コマンドの結果と VM シャドウ内で実行した ps コマンドの結果を比較して隠しプロセスを発見するシステム[4] を、既存の ps コマンドだけを用いて構築することができた。

以下、2 章で仮想マシンを用いた IDS のオフロードの問題点について述べ、3 章で VM シャドウを提供する Transcall について述べる。4 章で Transcall の実装について述べ、5 章で Transcall を用いて行った実験について述べる。6 章で関連研究に触れ、7 章で本稿をまとめる。

2. 仮想マシンを用いた IDS のオフロード

攻撃者の侵入を検知するために IDS[5],[8],[9] がよく用いられているが、検知までの時間差を利用して検知前に IDS を無力化される危険性がある。例えば、攻撃者がインストールしたルートキットを検出する IDS として chkrootkit[8] がよく用いられている。攻撃者は chkrootkit 本体を改ざんすることで、ルートキットの検出を容易に回避することができる。chkrootkit 本体を改ざんしなくても、chkrootkit が利用する外部コマンドを改ざんすることでも検出を無効化できる。chkrootkit は sshd の改ざんを検出するために、ps コマンドを用いて動作している sshd のパスを取得し、そのパスにある sshd を調べる。攻撃者は ps コマンドを改ざんすることで、実際に動作している sshd とは異なるパスを返すようにすることができる。その結果、実際には`/tmp` にある sshd が動作していたとしても、chkrootkit に正規のパスである`/usr/sbin` にある sshd を調べさせることができ、改ざんの検出を回避することができる。

このような IDS の無効化を防ぐために、仮想マシンを用いて IDS をオフロードするという手法が提案されている。[2],[4] IDS のオフロードとは、監視対象のシステムを**サーバ** *VM* と呼ばれる仮想マシン上で動作させ、IDS だけを *IDS-VM* と呼ばれる別の仮想マシンで動作させる手法である。IDS をオフロードすることにより、サーバ VM に侵入した攻撃者が IDS を無効化するのは難しくなる。例えば、chkrootkit 本体は IDS-VM に置かれることになるため、攻撃者に改ざんされることはない。また、chkrootkit が用いる ps コマンドも IDS-VM に置かれているものを使うことができるため、攻撃者が ps コマンドを改ざんして chkrootkit に実際に動作しているものと異なる sshd を調べさせることもできない。さらに、IDS の設定ファイルやデータベース、検出結果を記録するログなどの改ざんも防ぐことができるようになる。

しかし、IDS をオフロードする際にはしばしば IDS の修正が必要になるため、オフロードを行うのは容易ではなかった。例えば、chkrootkit が利用する ps コマンドとして IDS-VM の標準の ps コマンドをそのまま用いることはできない。オフロードした chkrootkit が必要とするのはサーバ VM のプロセスの一覧であるが、IDS-VM の ps コマンドは IDS-VM のプロセスの一覧を返してしまうためである。IDS への変更を最小限に抑えるために、リモートプロシージャコール（RPC）を用いてサーバ VM 上で ps コマンドを実行し、実行結果を取得するようにすることができる。ps コマンドを実行している部分を ssh などを使ったサーバ VM でのリモート実行に置き換えるだけでよい。この方法では IDS 本体を安全に動作させることが可能だが、ps コマンドはサーバ VM 上で実行されるため、実行結果を信用することができない。

ps コマンド等の改ざんの影響を受けずに、IDS-VM からサーバ VM の情報を取得することができる VIX ツール群[3] も提案されている。例えば、プロセス一覧を取得するために vix-ps コマンドが提供されており、サーバ VM の OS カーネルの内部構造を解析することでプロセスの一覧を取得することができる。VIX ツール群はサーバ VM の情報を取得するために ps や netstat などに対応するコマンドを提供しており、これらのコマンドを呼び出すだけの IDS はコマンドの実行部分を修正するだけで容易にオフロードすることができる。しかし、IDS 本体がシステムの情報を取得している場合、サーバ VM から情報を取得するように IDS を修正する必要がある。IDS には多種多様なものが存在しているため、個別に対応するのは多大な労力を必要とする。

3. Transcall

本稿では、オフロードした IDS に修正を加えることなく動作させることを可能にする *Transcall* を提案する。Transcall は IDS-VM に対してサーバ VM を監視するための実行環境である *VM* **シャドウ**を提供する。VM シャドウはあたかもサーバ VM にリモートログインしたかのような実行環境であり、IDS を VM シャドウの中で動作させることでサーバ VM から情報を取得させることができる。また、IDS に対する改ざんの影響を受けないように、IDS 本体や IDS から呼び出されるプログラムは IDS-VM 上に置かれたものを使う。Transcall のシステム構成は図 1 のようになっている。Transcall はサーバ VM の OS のある種のエミュレータであり、シャドウ VM 内で動かす IDS に対してシステムコールとファイルシステムのエミュレーションを行う。

3　仮想マシンを用いた既存 IDS のオフロード

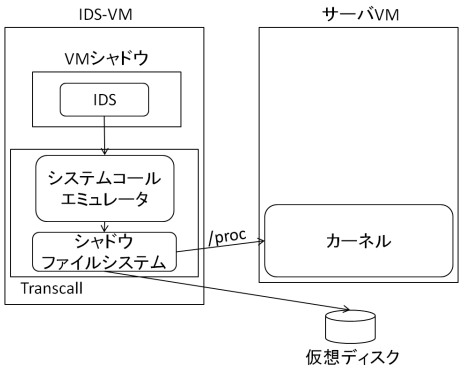

図 1　Transcall のシステム構成

Transcall はシステムコールをエミュレートすることにより、VM シャドウ内のプロセスがサーバ VM の OS の情報を取得することを可能にする。IDS は OS に対してシステムコールを発行することで OS の情報を取得して監視を行っている。例えば、uname システムコールを発行することでカーネルのバージョンなどの情報を取得する。Transcall はいくつかのシステムコールについて、VM シャドウの中では IDS-VM の OS の情報ではなく、監視しているサーバ VM の情報を返す。Transcall はサーバ VM のカーネル内部から直接情報を取得することで、VM シャドウの中のプロセスが発行したシステムコールをサーバ VM で実行したかのようにエミュレートする。これにより、攻撃者にサーバ VM のカーネルを改ざんされない限りは正確な情報を取得することができる。

Transcall は VM シャドウ内のプロセスがサーバ VM のファイルシステムから情報を取得できるように、**シャドウファイルシステム**を提供する。IDS はファイルの内容やパーミッションなどのメタデータを参照することで監視を行っている。例えば、chkrootkit は sshd の実行ファイル内に特定の文字列があった場合に改ざんと判定する。シャドウファイルシステムはサーバ VM のファイルシステムの内容を参照することを可能にする。通常のファイルを提供するファイルシステムに関しては、サーバ VM が使っている仮想ディスクを IDS-VM にもマウントすることで情報を取得する。ファイルアクセスに関するシステムコール全体をエミュレートする方法も考えられるが、複雑なファイルシステムのエミュレーションも必要になるため、IDS-VM の OS 内のファイルシステムを利用する。一方、Linux で使われている proc ファイルシステムに関しては、サーバ VM のカーネル内の情報を取得することで

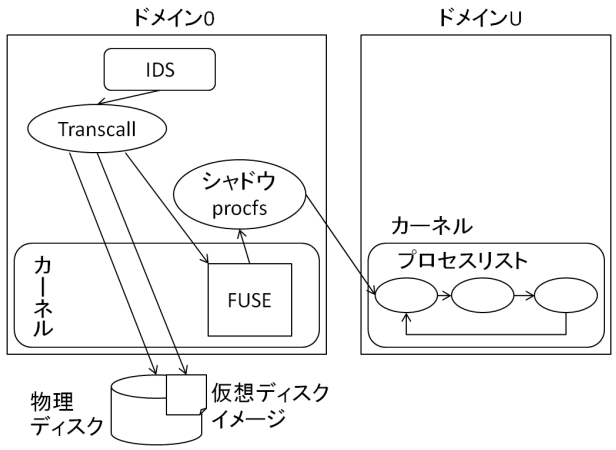

図 2　Transcall の実装

構成する。proc ファイルシステムは ps コマンドがプロセス情報を取得する際などに使われている。

また、Transcall はポリシファイルを記述することで、VM シャドウ内のプロセスが IDS-VM 内のファイルを使用することも可能にする。IDS のオフロードを行う際には、サーバ VM のファイルを参照するだけではなく、IDS-VM のファイルが必要になる場合がある。例えば、IDS 本体やそこから呼び出される外部コマンドは改ざんを防ぐために IDS-VM に置かれているものを使用する必要がある。また、IDS が使う設定ファイルや IDS が書き出すログなども改ざんを防ぐために IDS-VM に置く必要がある。Transcall ではポリシファイルに書かれたファイルについては、対応する IDS-VM 上のファイルへのアクセスに変換する。

4. 実　　装

我々は Transcall を Xen 3.4.0[1)] のドメイン 0 上に実装した。通常の仮想マシンであるドメイン U を監視対象のサーバ VM とし、特権を持った仮想マシンであるドメイン 0 を IDS-VM とした。我々は仮想マシンモニタやドメイン 0 およびドメイン U のカーネルへの変更を行わない方針で実装を行った。Transcall の実装は図 2 のようになっている。

4.1　システムコールのエミュレーション

Transcall は IDS が発行するシステムコールをエミュレートするために、ptrace システ

ムコールを利用してシステムコールをトラップする。Transcall は子プロセスを作った後、ptrace システムコールを使って PTRACE_TRACEME リクエストを実行させてから、IDS プログラムを実行させる。このリクエストは子プロセス自身が親プロセスにトレースされることを宣言するためのものである。子プロセスが明示的に宣言することで、その直後からトレースを行うことができる。プロセスのトレースを行う方法として、ptrace システムコールを使って親プロセスで PTRACE_ATTACH リクエストを実行する方法も考えられる。このリクエストは子プロセスに SIGSTOP シグナルを送って停止させてからトレースを開始する。しかし、この方法では子プロセスの最初のシステムコールを取りこぼす可能性がある。

子プロセスのトレースを開始したら、Transcall は waitpid システムコールを実行して子プロセスがシステムコールを発行するのを待つ。子プロセスがシステムコールを発行すると子プロセスに SIGTRAP シグナルが送られ、waitpid システムコールが終了する。Transcall は waitpid システムコールが返す子プロセスのステータスを調べ、SIGTRAP シグナルの配送によって停止されていればシステムコールのエミュレーションを行う。

Transcall は ptrace システムコールを使って PTRACE_GETREGS リクエストを実行して、子プロセスが発行したシステムコールに関する情報を取得する。このリクエストを実行すると子プロセスのレジスタ一覧を取得することができ、user_regs_struct 構造体にシステムコール番号や引数などが格納される。エミュレートすべきシステムコールだった場合、必要に応じて引数を取得する。引数が文字列だった場合、ptrace システムコールを使って PTRACE_PEEKTEXT リクエストを実行して 1 ワードずつ読み出すことで文字列全体を取得する。

Transcall は ptrace システムコールを使って PTRACE_SYSCALL リクエストを実行することで、子プロセスがシステムコールを実行した後で再度トラップする。この時点で Transcall はシステムコールのエミュレートを行う。現在、エミュレートしているシステムコールは uname のみである。Transcall はドメイン U のカーネルから uname が返す OS やアーキテクチャの情報を取得する。これらの情報は task_struct 構造体からたどれるメンバに格納されている。uname は utsname 構造体を引数に取るため、この構造体のメンバに取得した情報を書き込む。この場合、引数がポインタなので、ptrace システムコールを使って PTRACE_POKETEXT リクエストを実行し、子プロセスのメモリに 1 ワードずつ書き込む。user_regs_struct 構造体の値を変更した場合には、ptrace システムコールを使って PTRACE_SETREGS リクエストを実行し、レジスタへの変更を子プロセスに反映させる。

```
% losetup -a
/dev/loop1: [fd00]:12624259 (/var/lib/xen/images/fedora10vm.img)
% kpartx -a /dev/loop1
% pvscan
PV /dev/dm-3 VG VolGroupXen00 lvm2 [19.78 GB / 32.00 MB free]
  :
% lvscan
inactive '/dev/VolGroupXen00/LogVol00' [18.75 GB] inherit
  :
% vgchange -ay
2 logical volume(s) in volume group "VolGroupXen00" now active
% mount -o ro /dev/VolGroupXen00/LogVol00 /mnt/xvdb
```

図 3　仮想ディスクイメージのマウント手順

4.2　シャドウファイルシステム

ドメイン U の通常ファイルを参照できるようにするために、ドメイン U の仮想ディスクとして使われているディスクイメージを図 3 のようにしてドメイン 0 にマウントする。まず、losetup コマンドを実行して仮想ディスク用に使われているループバックデバイスを調べ、kpartx コマンドを実行して見つかったループバックデバイスに対してデバイスマップを作成する。LVM でなければこのデバイスを直接マウントすることができる。LVM の場合は、pvscan コマンドを実行して物理ボリュームを探し、lvscan コマンドを実行して論理ボリュームを探す。最後に、vgchange コマンドで論理ボリュームをアクティブに変更してからマウントする。ドメイン 0 に同じ物理ボリューム名があると論理ボリュームを探すのに失敗するため、名前を変更しておく必要がある。1 つのディスクイメージを複数箇所でマウントするとファイルシステムの整合性が破壊されてしまうため、ドメイン 0 からは読み込み専用でマウントする。

通常のファイルシステム以外の特殊なファイルシステムとして、Transcall はドメイン U の proc ファイルシステムを参照するための**シャドウ** *proc* **ファイルシステム** を提供している。proc ファイルシステムはプロセスやシステムの情報を取得するために Linux 等で使われているファイルシステムである。我々は FUSE[7] を用いてシャドウ proc ファイルシステ

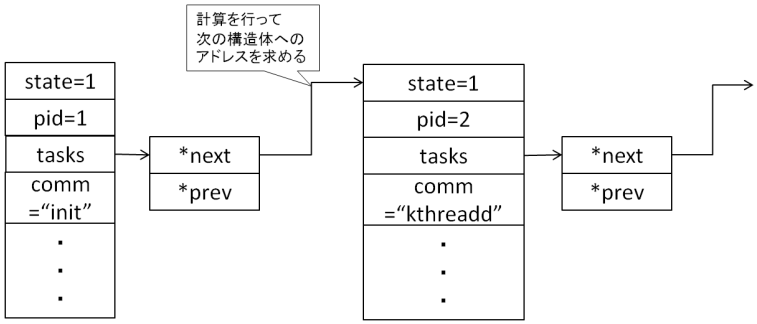

図 4　プロセスリストの構造

ムの実装を行った。FUSE はカーネルモジュールおよびライブラリで構成され、新しいファイルシステムをユーザプロセスとして構築することを可能にする。

シャドウ proc ファイルシステムは、proc ファイルシステムと同様に、ドメイン U で動いているプロセスの ID を名前として持つディレクトリをトップディレクトリ以下に作成する。その下に stat と status という名前のファイルを作成する。stat と status にはプロセス名やプロセス ID など、プロセスに関する情報が格納されている。ドメイン U で動いているプロセスの情報は、あらかじめ取得しておいたカーネルの型情報を基にプロセスリストを追跡することによって取得している。各プロセスに関する task_struct 構造体は図 4 のようにすべてポインタでつながっており、init_task 変数からたどることができる。

この他にトップディレクトリに存在するファイルとして、現在実行中のプロセスを指す self がある。IDS を実行している間、self はその IDS プロセスを指すのが意味的には正しいが、ドメイン U には該当するプロセスが存在しない。IDS はドメイン 0 の VM シャドウ内で実行されているためである。そこで、self についてはドメイン 0 の IDS プロセスの情報をコピーして作成している。

これらのシャドウファイルシステムは VM シャドウを作成するのと同時にマウントする。特に、シャドウ proc ファイルシステムについては、マウント時にドメイン U のカーネルを参照して上記のディレクトリのファイルをすべて作成する。これはアクセスされるたびにドメイン U を参照するのはオーバヘッドが大きいためである。ドメイン U のメモリを参照するオーバヘッドに加え、特定のプロセスの情報を取得するにはプロセスリストを先頭からたどり直す必要があるためである。このため、シャドウ proc ファイルシステムは VM シャドウを作成した時のスナップショットになる。IDS の実行に時間がかかる場合は、定期的にシャドウ proc ファイルシステムを作り直すなどの対処が必要となるが、これについては今後の課題である。

```
/etc/tripwire/tw.pol   /domU/etc/tripwire/tw.pol
/etc/tripwire/         /etc/tripwire/
/var/lib/tripwire/     /domU/var/lib/tripwire/
```

図 5　Tripwire のためのポリシファイル例

シャドウファイルシステムへのアクセスはシステムコールのエミュレーションと同様の方法を用いて、システムコールの引数を置換することで実現する。open システムコールと stat システムコールの第 1 引数が/proc で始まっていればシャドウ proc ファイルシステムをマウントしたディレクトリ名で置換し、それ以外であれば仮想ディスクイメージをマウントしたディレクトリ名で置換する。

4.3　ポリシファイル

VM シャドウ内のプロセスはデフォルトではドメイン U のファイルシステムを参照するため、ドメイン 0 のファイルシステムを参照したい場合にはポリシファイルに記述する。ポリシファイルには置換対象のパス名と置換後のパス名の対応を記述する。図 5 は Tripwire を動かす場合のポリシファイルの例である。Tripwire のポリシファイルやデータベースファイル、レポート等には専用のディレクトリを使用し、暗号鍵等は共通のディレクトリを使用している。

Transcall は open システムコールや stat システムコールをトラップした時に、ポリシファイルを上から順番に調べ、置換対象のパス名にマッチすればシステムコールのエミュレーションと同様にしてパス名を置換する。ただし、外部コマンド等の実行ファイルを execve システムコールで実行する場合には、安全のために常にドメイン 0 のファイルを使用する。異なるパスにある実行ファイルを使いたい場合だけポリシファイルに対応を記述する。

4.4　ドメイン U のカーネル情報の取得

Transcall は図 6 のようにして、ドメイン 0 からドメイン U のカーネルメモリを参照する。まず、ドメイン U のページテーブルを参照してドメイン U のカーネルが使っている仮想アドレスを疑似物理アドレスに変換する。次に、仮想マシンモニタ内のテーブルを参照して、疑似物理アドレスから仮想マシンモニタが使っているマシンアドレスを求める。このマ

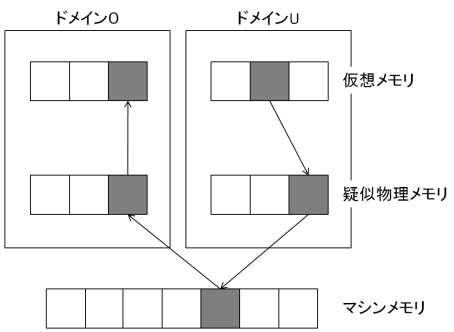

図 6　ドメイン U のメモリマップ

シンアドレスを含むメモリページをドメイン 0 にマップすることで、ドメイン 0 からドメイン U の指定したメモリアドレスを参照することができる。

Transcall はドメイン U のカーネルの型情報を用いてカーネルのデータ構造を解析する。型情報を取得するには、あらかじめカーネルをデバッグオプション付きでコンパイルしておく。そのカーネルに対して GDB の ptype コマンドを使って型情報を取得する。Transcall は事前に必要な情報をすべて取得しておく。

5. 実　　験

chkrootkit は外部コマンドを複数使用しているが、その中の ps コマンドについて実験を行った。実験に使用したマシンは、Intel Quad Core 2.83 GHz を 1 基、メモリを 4GB 搭載していた。Xen 3.4.0 を用い、ドメイン 0 で Linux 2.6.18.8 を、ドメイン U で Linux 2.6.27.35 を動作させた。

5.1　隠しプロセスの発見

Transcall を用いて、ルートキットによって隠されたプロセス情報が取得できることを確かめる実験を行った。ルートキットの模倣として、ドメイン U の ps コマンドを置き換え、init プロセスを表示しないようにした。図 7 がドメイン U で ps コマンドを実行した結果である。実行結果から、init プロセスが表示されていないことが分かる。一方、図 8 はドメイン 0 の ps コマンドを VM シャドウ内で実行した結果である。ドメイン 0 の ps コマンドは置き換えられていないため、正しくドメイン U 内の init プロセスが表示されていることが分かる。これらの結果を比較することで、特定のプロセスが隠されていたとしても発見することができる。

```
[taka@yone-vm program]$ ps -A
  PID TTY          TIME CMD
    2 ?        00:00:00 kthreadd
    3 ?        00:00:00 migration/0
    4 ?        00:00:00 ksoftirqd/0
    5 ?        00:00:00 watchdog/0
    6 ?        00:00:00 events/0
    7 ?        00:00:00 khelper
```

図 7　ドメイン U での ps の実行結果

```
[domU /] ps -A
  PID TTY          TIME CMD
    1 ?        00:00:00 init
    2 ?        00:00:00 kthreadd
    3 ?        00:00:00 migration/0
    4 ?        00:00:00 ksoftirqd/0
    5 ?        00:00:00 watchdog/0
    6 ?        00:00:00 events/0
    7 ?        00:00:00 khelper
```

図 8　VM シャドウ内での ps の実行結果

表 1　ps コマンドの実行時間

	実行時間 (μs)
ドメイン U での実行	19.9
VM シャドウ内での実行	65.6

5.2　ps コマンドの実行時間

ps コマンドをドメイン U で実行した場合と、ドメイン 0 の VM シャドウ内で実行した場合とで実行時間の比較を行った。それぞれ 1000 回繰り返して実行した時にかかった時間を測定し、10 回測定を行った。ps コマンド 1 回あたりの実行時間の平均を表 1 に示す。VM シャドウ内で実行した場合、ドメイン U で実行した場合の約 3.3 倍の実行時間がかかっていることが分かる。この原因は、ptrace ですべてのシステムコールをトラップするオーバヘッドと、シャドウ proc ファイルシステムに FUSE を用いていることによるオーバヘッドである。現在の実装では、ドメイン U のカーネルの解析はシャドウ proc ファイルシステムをマウントする際にだけ行っているため、このオーバヘッドは含まれていない。

6. 関 連 研 究

Livewire[2] や VMwatcher[4] は仮想マシンの外部で IDS を動作させて、仮想マシン内の OS の監視を行うことができる。仮想マシンの外部から OS の状態を調べるために、Transcall と同様に OS の内部構造に関する情報を用いている。これらのシステムでは専用の IDS を開発する必要があるが、従来の IDS とは異なる IDS を作成することができる。一方、VMwatcher ではアンチウィルスのような既存の IDS を仮想マシンの外側で動作させることもできる。しかし、これらの IDS はファイルシステムを参照するのみであり、仮想ディスクイメージをマウントできれば容易に動作させることができる。さらに、Transcall と違い、マウントした

ディレクトリに対して IDS を実行するように IDS の設定ファイルを変更する必要がある。

HyperSpector[6),11)] は Transcall と同様に、サーバと IDS を別々の仮想マシンで動作させる。HyperSpector でもサーバ VM のファイルシステムはシャドウファイルシステムとして IDS-VM に提供され、IDS-VM からサーバ VM のファイルシステムの監視を行うことができる。また、サーバ VM のプロセスはシャドウプロセスとして IDS-VM からもアクセスすることができる。さらに、サーバ VM が送受信するネットワークパケットを IDS-VM から監視することもできる。

Transcall との最も大きな違いは、HyperSpector は OS レベルの仮想化を前提としていることである。HyperSpector ではサーバ VM と IDS-VM は 1 つの OS を共有し、共通の OS が名前空間の制御を行うことにより IDS-VM からサーバ VM の監視を可能にしている。IDS-VM とサーバ VM は chroot のような機能を用いて別々のディレクトリを使うが、IDS-VM はサーバ VM が使っているディレクトリを参照することができる。また、サーバ VM に対しては参照できるプロセスを限定するが、IDS-VM はすべてのプロセスを参照することができる。

それに対して、Transcall はシステムレベルの仮想化を前提としているため、IDS-VM からサーバ VM を参照するのは容易ではない。サーバ VM のファイルシステムを参照するために、サーバ VM が使っている仮想ディスクイメージをマウントし、そのファイルシステムを解釈する。また、サーバ VM の proc ファイルシステムを参照できるようにするために、サーバ VM の OS カーネル内の情報を基に proc ファイルシステムを構築している。さらに、サーバ VM と IDS-VM の OS が異なるため、uname などのシステムコールもエミュレートする必要がある。

Proxos[10)] は既存のシステムをコモディティ VM で動かし、センシティブなデータを扱うアプリケーションを別のプライベート VM で動かすことを可能にしている。プライベート VM で動くアプリケーションはシステムコールごとにプライベート VM で実行するかコモディティ VM の OS に実行させるかを指定することができる。RPC を使ってコモディティ VM の OS にシステムコールを実行させるため、Proxos では OS を書き換える必要がある。RPC サーバをユーザプロセスで実現すれば OS への変更が不要になると考えられるが、RPC サーバが攻撃を受けると実行結果を容易に改ざんされてしまう。それに対して、Transcall では OS 内の情報を直接参照することで OS への変更を不要にしている。

7. ま と め

本稿では、既存の IDS に修正を加えることなくオフロードすることを可能にする Transcall を提案した。Transcall は IDS-VM からサーバ VM を監視するための VM シャドウと呼ばれる実行環境を提供する。Transcall によるシステムコールのエミュレーションとシャドウファイルシステムにより、VM シャドウの中で動作する既存の IDS にサーバ VM を監視させることができる。Transcall を用いて IDS-VM の ps コマンドをそのまま実行して、サーバ VM のプロセス情報を取得できた。

今後の課題は、シャドウ proc ファイルシステムを完成させることである。現在のところ、ps コマンドが返す実行結果がまだ不完全なため、proc ファイルシステムから必要な情報を取得できるようにして完全な実行結果を返せるようにする必要がある。また、ポリシファイルについても詳細な仕様を決定して実装する。さらに、chkrootkit を完全にオフロードできるようにするには、netstat コマンドも VM シャドウ内で実行できるようにする必要があり、そのためには様々なシステムコールのエミュレーションが必要になると考えられる。

参 考 文 献

1) 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. Symp. Operating Systems Principles*, pp.164–177 (2003).
2) Garfinkel, T. and Rosenblum, M.: A Virtual Machine Introspection Based Architecture for Intrusion Detection, *Proc. Network and Distributed Systems Security Symp.*, pp.191–206 (2003).
3) Hay, B. and Nance, K.: Forensics Examination of Volatile System Data Using Virtual Introspection, *SIGOPS Operating System Review*, Vol.42, No.3, pp.74–82 (2008).
4) Jiang, X., Wang, X. and Xu, D.: Stealthy Malware Detection through VMM-based "Out-of-the-box" Semantic View Reconstruction, *Proc. Conf. Computer and Communications Security*, pp.128–138 (2007).
5) Kim, G. and Spafford, E.: The Design and Implementation of Tripwire: A File System Integrity Checker, *Proc. Conf. Computer and Communications Security*, pp.18–29 (1994).
6) Kourai, K. and Chiba, S.: HyperSpector: Virtual Distributed Monitoring Environments for Secure Intrusion Detection, *In Proc. Int. Conf. Virtual Execution Environments*, pp.197–207 (2005).

7) M.Szeredi: Filesystem in Userspace, `http://fuse.sourceforge.net/`.
8) Murilo, N. and Steding-Jessen, K.: chkrootkit – Locally Checks for Signs of a Rootkit, `http://www.chkrootkit.org/`.
9) Roesch, M.: Snort – Lightweight Intrusion Detection for Networks, *Proc. USENIX System Administration Conf.* (1999).
10) Ta-Min, R., Litty, L. and Lie, D.: Splitting Interfaces: Making Trust between Applications and Operating Systems Configurable, *Proc. Symp. Operating Systems Design and Implementation*, pp.279–292 (2006).
11) 光来健一，　千葉滋：仮想的な分散監視環境による安全な侵入検知アーキテクチャ，情報処理学会論文誌：コンピューティングシステム，Vol.46, No.SIG 16, pp.108–118 (2005).