Vol.2011-OS-119 No.6
Vol.2011-EMB-23 No.6
2011/11/29

VM Shadow: 既存 IDS をオフロードするための実行環境

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実行環境である *VM Shadow* を提案する。VM Shadow はシステムコールのエミュレーションを行うことで、IDS にオフロード元の仮想マシンの OS の情報を返す。また、proc ファイルシステムも含め、オフロード元と同一のファイルシステムを提供する。ただし、IDS の実行ファイル等については安全のためにオフロード先の仮想マシンのファイルを提供する。現在の実装では、VM Shadow 内で chkrootkit や Tripwire などを動作させることができた。

VM Shadow: An Execution Environment for Offloading Existing IDSes

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, called a *server VM*, and executes only IDSes on another VM, called an *IDS-VM*. However, the proper execution of offloaded IDSes often requires great efforts. This paper proposes a *VM Shadow*, which is an execution environment for running offloaded IDSes without any modification. A VM Shadow emulates system calls and returns information of a server VM. It also provides the same file system as a server VM, including the proc file system. Note that it provides executable files in an IDS-VM for security. In the current implementation, we can run chkrootkit and Tripwire in VM shadows.

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）[6),9),10)] 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),5)] 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事態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 IDS への修正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 を安全に監視できるようにするには、サーバ VM のカーネルメモリから必要な情報を取得するように 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ための実行環境である *VM Shadow* を提案する。IDS を VM Shadow の中で動作させることで、実行ファイルや共有ライブラリ、設定ファイルなどは 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。VM Shadow は IDS にサーバ VM の OS の情報を取得させるため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 のファイル情報を取得できるように、VM Shadow はサーバ VM と同一のファイルシステムを提供する。特に、サーバ VM のプロセスやネットワークの情報を提供する proc ファイルシステムも提供する。

我々は VM Shadow を提供するシステム *Transcall* を Xen[1)] のドメイン 0 上に実装した。Transcall は 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

†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

Vol.2011-OS-119 No.6
Vol.2011-EMB-23 No.6
2011/11/29

オフロードを実現する。Transcall のシステムコール・エミュレータは VM Shadow 内のプロセスが発行するシステムコールをトラップし、必要に応じてサーバ VM のカーネルメモリを解析して情報を返す。また、Shadow ファイルシステムはサーバ VM のファイルシステムをマウントして名前空間をサーバ VM と一致させることで実現し、実行ファイルや共有ライブラリを自動判別して IDS-VM のファイルを使わせる。Shadow proc ファイルシステムはサーバ VM のカーネルメモリを解析することで proc ファイルシステムをエミュレートする。現在のところ、VM Shadow 内で chkrootkit や Tripwire などを動作させることができている。

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 Shadow について述べる。4 章で VM Shadow を提供する Transcall の実装について述べ、5 章で VM Shadow 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[6),9),10)] 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。攻撃者は IDS 自体を置き換えてしまうことで攻撃が検出されないようにすることができる。chkrootkit のように ps コマンドや netstat コマンドなどの外部コマンドを使っている場合には、これらを置き換えることでも検出が回避できる。例えば、sshd を隠蔽する ps コマンドに置き換えれば、chkrootkit は sshd のチェックを行わなくなるため、改ざんした sshd を動かし続けることができる。また、IDS のポリシファイルやデータベースを改ざんすることでも、IDS を無力化することができる。例えば、Tripwire は検査するファイルをポリシファイルに記述し、正常時のファイルの情報をデータベースに記録している。これらのファイルを改ざんされると、Tripwire はファイルの改ざんを検出できなくなる。

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),5)] 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。IDS 本体が IDS-VM に置かれることになるため、サーバ VM に侵入した攻撃者に改ざんされる恐れはない。IDS から呼び出される外部コマンドも IDS-VM に置かれているものが使われるため、IDS は外部コマンドの実行結果を信頼することができる。さらに、IDS の設定ファイルやデータベース、検査結果を記録したログなどの改ざんも防ぐことができる。

しかし、IDS をオフロードする際には 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、IDS-VM にオフロードした chkrootkit はサーバ VM のプロセスを調べる必要があるが、chkrootkit をそのまま実行すると IDS-VM のプロセスを調べてしまう。サーバ VM のプロセスに関する情報を安全に取得するにはサーバ VM と通信して情報を取得するのではなく、サーバ VM のカーネルメモリを直接、解析する必要がある。この解析にはサーバ VM のカーネルデータの型情報やシンボル情報等を用いて、アドレス変換を行いながら情報を取得するという複雑なプログラミングが必要になる。一方、Tripwire のようにディスクの検査を行う IDS の場合サーバ VM のディスクをマウントすれば安全にアクセスすることができるが、設定ファイルに書かれた検査するファイルのパス名をマウント先に合わせて書き換える必要があり、この作業を手で行うと書き換えミスが発生しやすい。また、IDS プログラムにパス名が埋め込まれている場合にはプログラムへの修正も必要になる。

オフロードを容易にするために、よく使われるコマンドについては、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)] を使うこともできる。例えば、サーバ VM のプロセス一覧を取得するための vix-ps コマンドや、ネットワーク情報を取得するための vix-netstat などが提供されている。VIX ツール群はサーバ VM のカーネルメモリを解析することで情報を取得している。これらのコマンドを呼び出すだけの IDS はコマンド名の部分を修正すればオフロードを行うことができる。しかし、IDS 本体がシステムコールや proc ファイルシステムなどを用いてシステムの情報を取得している場合は IDS への修正が必要となる。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. VM Shadow

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 *VM Shadow* を提案する。VM Shadow は図 1 のように IDS-VM 上のプロセスがサーバ VM を透過的に監視するための実行環境である。VM Shadow は監視に関してサーバ VM にリモートログインしたかのような実行環境を提供し、既存の IDS を VM Shadow の中で動作させることでサーバ VM の情報へのアクセスを可能とする。VM Shadow は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。

Vol.2011-OS-119 No.6
Vol.2011-EMB-23 No.6
2011/11/29

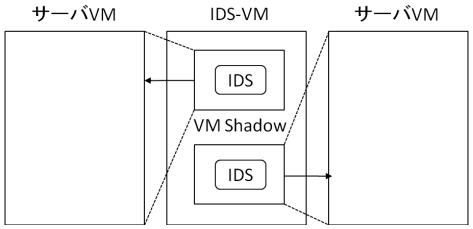

図 1 VM Shadow を用いた監視

VM Shadow はその中で動作するプロセスが発行するシステムコールをエミュレートすることにより、サーバ VM の OS の情報を返す。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。VM Shadow はいくつかのシステムコールについて、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。この際に、サーバ VM のカーネルメモリから直接、情報を取得することで、VM Shadow 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。ただし、メモリ管理やネットワークに関するシステムコールについては IDS-VM の機能を利用することができるため、IDS-VM の OS に対して発行する。

VM Shadow はプロセスがサーバ VM のファイル情報を取得できるように、サーバ VM と同一の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、Tripwire はファイルの内容のハッシュ値が正常値と異なっていた場合に改ざんと判定する。サーバ VM とファイルシステムの名前空間も同じであるため、IDS はサーバ VM で動作する時と同じパス名を用いることができる。そのため、IDS にハードコードされたパス名や設定ファイルに書かれているパス名を変更する必要がない。また、特殊なファイルシステムとして、サーバ VM の proc ファイルシステムも提供している。proc ファイルシステムはプロセスやネットワークの情報を提供しており、ps コマンドや netstat コマンドによって使われている。VM Shadow はシステムコールのエミュレーションと同様に、サーバ VM のカーネルメモリから直接、情報を取得することで proc ファイルシステムをエミュレートする。VM Shadow は実行ファイルや共有ライブラリなど、IDS-VM 上で実行されるファイルを自動的に判別し、サーバ VM

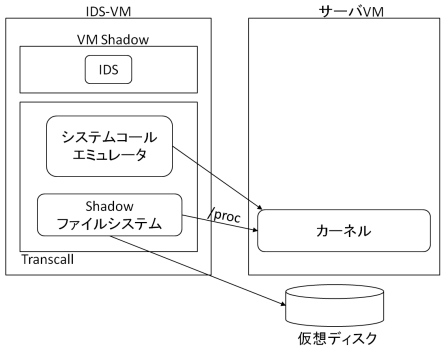

図 2 Transcall のシステム構成

上のファイルではなく、IDS-VM 上のファイルを用いる。これはサーバ VM に置かれている実行ファイルや共有ライブラリは攻撃者によって改ざんされている可能性があり信頼できないためである。IDS-VM 上のプログラムだけを実行に用いることにより、安全に実行することができる。ただし、Tripwire 等を用いてサーバ VM 上の実行ファイルや共有ライブラリを検査する際には、IDS-VM 上のファイルではなくサーバ VM 上のファイルを読み込む。これにより、正しくサーバ VM 上のファイルを検査することができる。

VM Shadow 内の IDS が用いる設定ファイルやデータベースなどに関しては、マッピングファイルを記述することで IDS-VM 内のファイルを使用させることができる。IDS が用いるこれらのファイルをサーバ VM 上のファイルシステムから読み込むと、改ざんされている恐れがあるためである。このようなファイルは実行ファイルと異なり、IDS の動作に影響を与える重要なファイルであるか、検査の対象として読み込むファイルであるかを区別することが難しいため、マッピングファイルを用いてユーザが指定する。VM Shadow はマッピングファイルに記述さ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 VM Shadow を提供するシステム *Transcall* を開発した。Transcall のシステム構成は図 2 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、システムコール・エミュレータと Shadow ファイルシステムからなる。我々は Transcall を

Vol.2011-OS-119 No.6
Vol.2011-EMB-23 No.6
2011/11/29

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。Transcall の現在の実装は、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必要としない。

4.1 システムコール・エミュレータ

Transcall は IDS が発行するシステムコールにドメイン U の OS の情報を返させるために、IDS を子プロセスとして実行し、ptrace システムコールを使ってシステムコールをトラップしてエミュレートする。

現在の実装では uname システムコールだけをエミュレートしている。Transcall が uname システムコールをトラップするとドメイン U のカーネルメモリから uname に返させる OS やアーキテクチャの情報を取得する。これらの情報はプロセスを表す task_struct 構造体からたどれるメンバに格納されている。uname システムコールは引数で渡される utsname 構造体に情報を格納して返す仕様になっているため、Transcall はこの構造体にドメイン U から取得した情報を書き込む。この引数はポインタなので、ptrace システムコールを使い、子プロセスである IDS のメモリに 1 ワードずつ書き込む。

4.2 Shadow ファイルシステム

Shadow ファイルシステムは VM Shadow に対してドメイン U と同一のファイルシステムを提供するためのファイルシステムである。Shadow ファイルシステムを実現するために、Transcall は VM Shadow を作成した時にドメイン 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ドメイン 0 にマウントする。1 つのディスクイメージをドメイン U とドメイン 0 の両方で書き込み可能にしてマウントするとファイルシステムの整合性が失われてしまうため、ドメイン 0 では読み込み専用でマウントする。Shadow ファイルシステムはドメイン U と同じ名前空間を提供させるために、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、ファイル関連のシステムコールの引数のパス名を置換する。例えば、ディスクイメージを/vm1 にマウントしたとすると、open システムコールの第 1 引数が絶対パス名だった場合には先頭に/vm1 を付加する。これにより、VM Shadow 内ではサーバ VM と同じパス名でファイルにアクセスすることができる。

VM Shadow 内のプロセスは基本的にはドメイン U のファイルシステムを参照する。が、ファイルを実行する場合には安全のためにドメイン 0 のファイルを用いる。実行のためにファイルを読み込む際には open システムコールではなく、execve システムコールが用いられるため、Shadow ファイルシステムはシステムコールによって実行ファイルかどうかを区別する。execve システムコール経由でファイルにアクセスされた場合には、第 1 引数のパス名を書き換えることでドメイン 0 のファイルを使わせる。一方、実行ファイルであっても open システムコールを用いて読み込まれる場合には、通常のファイルと同様にドメイン U から読み込む。

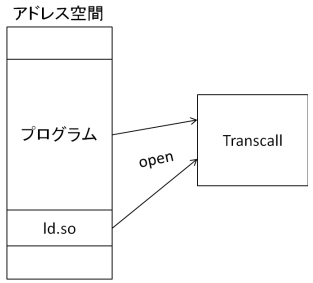

図 3 呼び出し元の判別

共有ライブラリについても、実行ファイルによって読み込まれて実行されるため、ドメイン 0 のファイルを用いる。共有ライブラリが dlopen システムコールによって明示的に読み込まれる場合は、実行ファイルと同様にしてドメイン 0 のファイルを読み込む。一方、暗黙的に読み込まれる場合には、通常のファイルと同じように読み込まれるため、システムコールレベルでは実行されるかどうかを区別することができない。そこで、プログラム本体から読み込まれるか、プログラムに埋め込まれたダイナミックリンカ（ld.so）から読み込まれるかを判別することで、共有ライブラリが実行されるかどうかを区別する。(図 3) ダイナミックリンカはプログラムとは異なるアドレスに配置されるため、open システムコールを実行した時の eip レジスタの値を基に判別することができる。

これら以外のファイルについてドメイン 0 のファイルシステムを参照したい場合にはマッピングファイルに記述する。マッピング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 4 は Tripwire のマッピングファイルの例である。1 行目では、仮想マシンごとに設定ファイルを変えている。2 行目では、特定のディレクトリ以下へのアクセスをすべてドメイン 0 から行うようにしている。3 行目では仮想マシンごとにレポートの出力先を変えている。Transcall は open システムコールなどをトラップした時に、マッピングファイルを上から順番に調べ、置換対象のパス名に一致すればパス名を置換する。

Vol.2011-OS-119 No.6
Vol.2011-EMB-23 No.6
2011/11/29

```
/etc/tripwire/tw.pol  /etc/tripwire/vm1/tw.pol
/etc/tripwire/        /etc/tripwire/
/var/lib/tripwire/    /var/lib/tripwire/vm1/
```

図 4 Tripwire のためのマッピングファイル例

4.3 Shadow proc ファイルシステム

Shadow proc **ファイルシステム**は Shadow ファイルシステムの一部であり、ドメイン U の proc ファイルシステムの情報を提供する。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。VM Shadow を作成した時に Transcall はドメイン U のカーネルメモリを参照して Shadow proc ファイルシステムのすべてのファイルを作成する。そのため、Shadow proc ファイルシステムは VM Shadow を作成した時のスナップショットになる。これはアクセスされるたびにドメイン U のカーネルメモリから情報を取得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。IDS の実行に時間がかかる場合は、定期的に Shadow 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。我々は FUSE[8] を用いて Shadow proc ファイルシステムの実装を行った。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。

Shadow proc ファイルシステムを構築するために、Transcall はドメイン 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し、ドメイン U のプロセス名やプロセス ID などのプロセスに関する情報を格納する。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ドメイン U のカーネルメモリ中のプロセスリストを追跡することによって取得する。各プロセスに関する task_struct 構造体は図 5 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。また、task という名前のディレクトリを作成し、その下にスレッド ID を名前として持つディレクトリを作成する。その下にプロセスと同様に stat と status という名前のファイルを作成する。スレッドは task_struct 構造体の thread_group というリストからたどることができる。

Transcall は各プロセスのディレクトリの下に実行ファイルのパス名が格納された cmdline

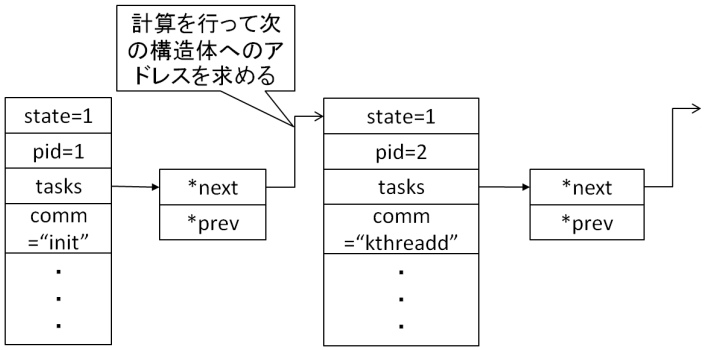

図 5 プロセスリストの構造

という名前のファイルを作成する。この情報は task_struct 構造体ではなく、プロセスのメモリ上に置かれている。そのため、カーネルメモリを解析するだけでは取得することができない。Transcall は task_struct 構造体のメンバの mm_struct 構造体からプロセスのページテーブルの情報を取得し、そのページテーブルを使ってアドレス変換を行いながらプロセスのメモリを解析することで実行ファイルのパス名を取得する。ただし、プロセスの当該メモリがページアウトされていた場合には現在のところ、パス名を取得することはできない。

さらに、Transcall はトップディレクトリに現在実行中のプロセスを指す self というディレクトリを作成する。IDS を実行している間、self はその IDS プロセスを指すべきだが、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 Shadow 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する。

また、Transcall は net というディレクトリを作成し、その中にドメイン U のネットワーク情報が格納された tcp、udp、unix というファイルを作成する。ネットワークに関する情報もドメイン U のカーネルメモリを解析することによって取得する。この情報は sock 構造体から取得することができ、それぞれ tcp_hashinfo、udp_hash、unix_socket_table という変数から追跡することができる。これらの変数はすべてハッシュになっており、先頭から調べていくことでネットワーク情報をすべて取得することができる。

4.4 ドメイン U のカーネル情報の取得

Transcall はドメイン 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

Vol.2011-OS-119 No.6
Vol.2011-EMB-23 No.6
2011/11/29

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マシンメモリフレーム番号に変換する。マシンメモリフレーム番号は物理メモリ全体にページ単位でつけられた番号である。このマ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 の指定したアドレスのメモリ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き、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。

GDB を用いて型情報を全て手作業で取得するのは多大な労力を必要とするため、Transcall は型情報を自動取得するツールを提供している。このツールは取得したい型名を指定すると再帰的に型情報を取得する。例えば、task_struct 構造体の型情報を取得しようとすると、それに属するメンバの型情報も取得することができる。ただし、ポインタの型情報も再帰的に取得するとアクセスしない限り必要にならないにも関わらず、型情報が大きくなりすぎてしまうため、ポインタの型情報は取得しない。ポインタの型情報が必要な場合にはその型を指定して再度ツールを実行すれば、必要な型情報を全て取得することができる。

5. 実　　験

VM Shadow の有用性を確かめるために、既存の chkrootkit と Tripwire をオフロードする実験を行った。また応用として、ps コマンドの実行結果を比較して隠しプロセスを見つける実験を行った。実験に使用したのは、Intel Quad Core 2.83 GHz を 1 基、メモリを 4GB 搭載したマシンである。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 オフロードした IDS の動作テスト

VM Shadow を用いて chkrootkit のオフロードが行えるか確かめる実験を行った。chkrootkit が VM Shadow 内で正しく動作しているか確かめるために、ドメイン 0 の VM Shadow 内とドメイン U で同じ chkrootkit を実行し、その結果、実行結果を比較したところ、ほぼ同じ出力結果が得られていることが確かめられた。出力結果が異なっていたのは、ネットワークインターフェースを検査する箇所である。VM Shadow はドメイン U のネットワークインターフェース情報の取得にはまだ対応できておらず、ドメイン 0 のネットワークインターフェースを検査していた。

次に、VM Shadow を用いて Tripwire をオフロードする実験を行った。Tripwire をオフロードするためには、図 4 のようなマッピングファイルが必要であった。比較のため、Tripwire をドメイン 0 の VM Shadow 内とドメイン U で実行した。実験の結果、VM Shadow 内とドメイン U で同じ結果が得られ、Tripwire をオフロードして正しく実行できていることを確認した。実際にドメイン U で新しくファイルを作成し、VM Shadow で Tripwire を動作させたところ追加ファイルを検出できた。

```
[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 6 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 7 VM Shadow 内での ps の実行結果

5.2 隠しプロセスの発見

VM Shadow 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。ルートキットを模倣して、ドメイン U の ps コマンドを置き換え、init プロセスを出力しないようにした。図 6 はドメイン 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 7 は VM Shadow 内で ps コマンドを実行した結果である。ドメイン 0 の ps コマンドが実行されるため、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。

5.3 オフロードした IDS の実行時間

chkrootkit をドメイン 0 の VM Shadow 内で実行した場合とドメイン U で実行した場合とで実行時間の比較を行った。OS 内のキャッシュの影響を排除するために実行ごとにマシンを再起動しながら、実行時間を 5 回測定した。chkrootkit1 回あたりの実行時間の平均を表 3 に示す。VM Shadow 内で実行した場合、ドメイン U で実行した場合の約 1.5 倍の実行時間がかかっていることが分かる。chkrootkit はそれほど頻繁に実行する IDS ではないため、この程度の性能であれば許容範囲内であると考えられる。現在の実装では ptrace や FUSE を用いているため、カーネル内で実装すればこの性能差は縮まるはずである。

chkrootkit の実行時間には VM Shadow を作成した時に Shadow ファイルシステムを作成する時間も含まれている。Shadow ファイルシステムの作成時にはドメイン U の仮想ディ

Vol.2011-OS-119 No.6
Vol.2011-EMB-23 No.6
2011/11/29

表 1 Shadow ファイルシステム作成時間

	時間 (s)
仮想ディスクイメージのマウント	2.4
Shadow proc ファイルシステムの作成	4.7
合計	7.1

表 2 各コマンドの実行時間

	ドメイン U での実行 (ms)	VM Shadow 内での実行 (ms)
ps	59.2	137.3
netstat	180.4	190.2

表 3 各 IDS の実行時間

	ドメイン U での実行 (s)	VM Shadow 内での実行 (s)
chkrootkit	38.6	58.5
Tripwire	120.8	97.1

スクイメージをマウントする必要があり、Shadow proc ファイルシステムの作成時にはドメイン U のカーネルメモリを解析する必要がある。Shadow ファイルシステムの作成にかかる時間を測定したところ 7.1 秒であった。ドメイン U の仮想ディスクには LVM を使用しているとマウントにため、マウントを行うために多くの手段が必要となり 2.4 秒かかっている。また、Shadow proc ファイルシステムはドメイン U のプロセス数やソケット数に依存するが、プロセス数が 132 個、ソケット数が 627 個の時に 4.7 秒であった。

また、chkrootkit から呼び出され、Shadow proc ファイルシステムを利用する ps コマンドと netstat コマンドについて、ドメイン 0 の VM Shadow 内で実行した場合とドメイン U で実行した場合とで実行時間の比較を行った。それぞれ 100 回繰り返して実行した時にかかった時間を測定し、この実験を 10 回行った。それぞれのコマンドの実行時間の平均を表 2 に示す。VM Shadow 内で実行した場合、ドメイン U で実行した場合と比べて、ps コマンドで約 2.3 倍、netstat コマンドで約 1.1 倍の実行時間がかかっていることが分かる。

次に、Tripwire について chkrootkit と同様の実験を行った。検査対象のファイル数は 30755 個であった。Tripwire の実行時間の平均を表 3 に示す。VM Shadow 内で実行した場合、chkrootkit とは逆に約 1.2 倍速くなっている。ドメイン U の仮想ディスクはドメイン 0 上に置かれているため、ドメイン U からアクセスするより、ドメイン 0 の VM Shadow からアクセスした方が高速であるためである。Tripwire の実行はシステムコールを ptrace でトラップする分だけ遅くなるが、ディスクアクセスの高速化のほうが効いている。

6. 関連研究

Livewire[2] や VMwatcher[5] 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、VM Shadow 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。ただし、VM Shadow 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[7],[12] は VM Shadow 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 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 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。

VM Shadow 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、VM Shadow 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 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[11] 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、VM Shadow 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 VM Shadow を提案した。VM Shadow は IDS-VM からサーバ VM を透過的に監視することができる実行環境である。VM Shadow はシステムコールをエミュレーションすることにより、その中で動作するプロセスにサーバ VM の情報を返す。また、proc ファイルシステムも含めてサーバ VM と同一のファイルシステムを提供するが、安全のために実行ファイルや共有ライブラリ、設定ファイル等については IDS-VM のファイルを使わせる。我々は VM Shadow を提供するシステム Transcall を開発し、既存の chkrootkit や Tripwire を動作させることができた。

今後の課題は、VM Shadow 内で chkrootkit を動作させた時に正しくルートキットを検出できることを確認することである。現状ではまだ、様々な検出のそれぞれが正しく機能するかどうかについては検証できていない。また、OSSEC[4] などの他の IDS の動作を確認し、より多くの既存の IDS を動作させられるようにする必要もある。

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) Inc., T.M.: OSSEC, `http://www.ossec.net/`.
5) 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).
6) 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).
7) 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).
8) M.Szeredi: Filesystem in Userspace, `http://fuse.sourceforge.net/`.
9) Murilo, N. and Steding-Jessen, K.: chkrootkit – Locally Checks for Signs of a Rootkit, `http://www.chkrootkit.org/`.
10) Roesch, M.: Snort – Lightweight Intrusion Detection for Networks, *Proc. USENIX System Administration Conf.* (1999).
11) 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).
12) 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).