平成　２１年度　卒業論文概要			
所　　属	機械情報工学科	指導教員	光来　健一
学生番号	06237005	学生氏名	飯田　貴大
論文題目	仮想マシンを用いた既存ソフトウェアのオフロード手法		

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いる。万一、攻撃者に侵入されてしまうと、重要な情報を盗まれたり、他のサーバへの攻撃に利用されたりしてしまう。そこで、攻撃者の侵入を検知するために侵入検知システム（IDS）がよく用いられている。この IDS が攻撃を受けて停止させられてしまうと侵入を検知できなくなってしまうため、近年、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、サーバと IDS を別々の仮想マシン上で動作させることにより IDS が攻撃を受けるのを防ぐ。しかし、IDS をオフロードして動作させられるようにするにはプログラムの大幅な変更などの多大な労力が必要とされることが多く、既存の IDS を使うことができなかった。

この問題を解決するために、本研究では IDS に修正を加えることなくオフロードできるようにするシステム Transcall を提案する。本システムは IDS が発行するシステムコールを横取りし、必要に応じて監視対象のサーバが動いている OS の内部情報を返す。これにより、IDS のプログラムへの修正が不要になる。

2.　仮想マシンを用いた IDS のオフロードの問題点

仮想マシンは計算機のハードウェア資源を仮想化したものである。仮想マシンを用いると、一台の計算機上に複数の計算機を仮想的に作成することができる。個々の仮想マシンはそれぞれ独立しており、お互いに干渉したり影響を及ぼしたりすることはない。本研究では Xen という仮想化ソフトウェアを用いて仮想環境を構築する。

仮想マシンを用いた IDS のオフロードとは、ある仮想マシンから別の仮想マシンへと IDS を移動させる手法である。移動させた IDS は移動元の仮想マシンの監視を行う。攻撃を受けやすいサービスを提供している仮想マシンから IDS を切り離すことによって IDS を安全に動かすことが可能になる。(図 1)

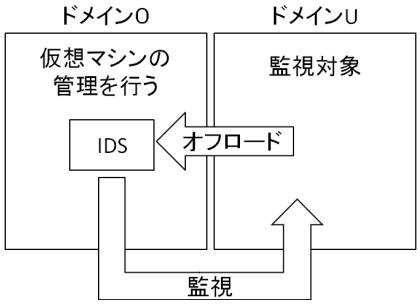

図1　IDS のオフロード

仮想マシンを用いたオフロードの問題点は、オフロードを行うためには多大な労力が必要となることである。IDS は動作している仮想マシン内を監視するように設計されているため、そのままでは別の仮想マシンを監視することはできない。別の仮想マシンを監視できるようにするには、仮想化ソフトウェアの機能を利用する必要がある。そのため、IDS のプログラムを大幅に書き換える必要がある。

例えば、本研究で用いた chkrootkit という IDS はプロセスの一覧を取得する ps コマンドを使用している。ps コマンドは実行された仮想マシン上のプロセス情報を取得するため、監視対象の OS のプロセス一覧を取得することができない。したがって、従来手法では ps コマンドに修正を加えなければならなかった。

3.　Transcall

本研究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システム Transcall を提案する。仮想化ソフトウェアとして Xen を用い、IDS をドメイン 0 と呼ばれる特権を持った仮想マシンで動作させる。監視対象のサーバはドメイン U と呼ばれる通常の仮想マシンで動作させる。Transcall は IDS と同じドメイン 0 で動作し、図 2 のようにシステムコールのエミュレートを行うことで、既存の IDS を用いてドメイン U の監視を行うことを可能にしている。

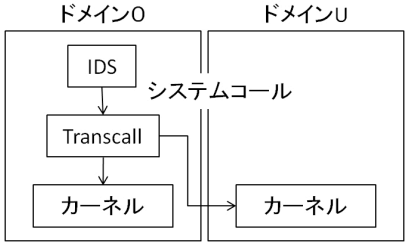

図2　システムの構成

3.1.　システムコールのエミュレート

IDS が監視対象のドメイン U の情報を取得できるようにするために、Transcall はシステムコールのエミュレートを行う。IDS は OS に対してシステムコールを発行することで、OS の情報を取得したりファイルの読み書きを行ったりしながら監視を行っている。そのため、IDS が動いているドメイン 0 の情報ではなく、監視しているドメイン U の情報を返すようにエミュレートを行う。

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。ptrace を用いると子プロセスの監視および制御を行うことができ、IDS を Transcall の子プロセスとして実行することで、IDS がシステムコールを発行した時に親プロセスである Transcall に制御が移る。Transcall はドメイン U の情報を参照して、IDS にその情報を返す。

3.2. 疑似 proc ファイルシステムの提供

本研究に用いた Linux ではプロセス情報を proc ファイルシステム経由で取得するため、Transcall は疑似的な proc ファイルシステムを提供する。proc ファイルシステムには現在動作しているプロセスに関する情報が格納されている。例えば、/proc/1/stat というファイルを読むことでプロセス ID が 1 のプロセスの状態を取得することができる。IDS がこのようなファイルにアクセスしようとすると、Transcall によってシステムコールのエミュレートが行われ、疑似 proc ファイルシステムへのアクセスに変更される。疑似 proc ファイルシステムはその時点のドメイン U のプロセス情報を取得し、システムコールの結果として IDS にプロセス情報を返す。この疑似 proc ファイルシステムは FUSE を用いて作成した。

3.3. ドメイン U のカーネル情報の取得

Transcall は Xen の機能を用いてドメイン U のカーネルメモリを直接参照することにより必要な情報を取得する。例えば、ドメイン U のプロセス情報を取得するには、図 3 のようにプロセス情報が置かれているカーネルメモリをドメイン 0 にマップしてから参照する。プロセス情報のデータ構造や置かれているアドレスはドメイン U のカーネルのデバッグ情報を基に解析する。

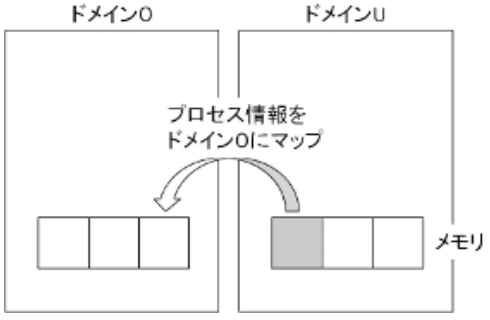

図 3 メモリ参照方法

Transcall がドメイン U と通信して情報を送ってもらう方法も考えられるが、ドメイン U が既に攻撃を受けていた場合には信用できない情報になる。ドメイン U のカーネルメモリを直接参照して情報を取得することで、ドメイン U 上では発見できない攻撃を見つけられる可能性が高まる。例えば、ドメイン U では隠されているプロセスであってもドメイン 0 からなら発見することができることが多い。

3.4. ps コマンドの動作例

例として、ドメイン 0 で実行した ps コマンドの動作について説明する。ps コマンドはプロセス情報を取得するために図 4 のようなシステムコールを発行する。この中の/proc を参照しているシステムコールがプロセス情報と関係しているため、その引数を置き換えることでシステムコールのエミュレートを行う。この時、/myproc にマウントされている疑似 proc ファイルシステムにアクセスさせるために、/proc を/myproc で置き換える。/myproc にアクセスすると、疑似 proc ファイルシステムはドメイン U のカーネルメモリを参照してプロセス情報を取得する。その結果、ps コマンドにドメイン U のプロセス情報が返され、この ps コマンドはドメイン U のプロセス一覧を表示する。

```
stat("/proc/503", {st_mode=S_IFDIR|0555, st_size=0, ...}) = 0
open("/proc/503/stat", O_RDONLY)      = 5
read(5, "503 (scsi_eh_1) S 19 1 1 0 -1 328"..., 1023) = 158
close(5)                              = 0
```

図 4 システムコール例

4. 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 Transcall を用いることで修正なしにオフロードが可能になっている。Transcall を用いた隠しプロセスの発見および実行性能の評価のための実験を行った。実験に使用したマシンは、CPU Intel Quad 2.83GHz、メモリ 4GB であった。Xen 3.4.0 を用い、ドメイン 0 で Linux 2.6.18.8、ドメイン U で Linux 2.6.27.35 を動作させた。

4.1. 隠しプロセスの発見

Transcall を用いて隠しプロセスを発見できるかどうか調べる実験を行った。この実験ではドメイン U で init プロセスを隠した。図 5 が実行結果である。(a)はドメイン U で実行した ps コマンドの結果であるが、init プロセスが表示されていない。(b)はドメイン 0 での実行結果であり、init プロセスが表示されている。このように、Transcall を用いることで隠された init プロセスを発見できた。

```
[root@yone-vm ~]# 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
   15 ?        00:00:00 xenwatch
   16 ?        00:00:00 xenbus
   79 ?        00:00:00 kintegrityd/0
   81 ?        00:00:00 kblockd/0
   87 ?        00:00:00 cqueue
```

(a) ドメイン U

```
[root@yone fuse_renew]# ./myps -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
   15 ?        00:00:00 xenwatch
   16 ?        00:00:00 xenbus
   79 ?        00:00:00 kintegrityd/0
   81 ?        00:00:00 kblockd/0
```

(b) ドメイン 0

図 5 プロセス情報比較

4.2. Transcall のオーバヘッド

Transcall を用いることによるオーバヘッドを調べる実験を行った。ドメイン U で ps コマンドを動かした場合（orig_ps）とドメイン 0 で Transcall を使って ps コマンドを動かした場合（transcall_ps）について実行時間を測定した。表 1 に測定結果を示す。表から orig_ps の実行速度のほうがかなり速いことが分かる。

表 1 実行速度比較

	実行時間（秒）
orig_ps	19.9
transcall_ps	65.6

5. まとめ

本研究では既存の IDS を変更することなく、仮想マシンを用いてオフロードできるようにするシステム Transcall を提案した。Transcall はシステムコールのエミュレーション、疑似 proc ファイルシステム、ドメイン U のカーネルメモリの参照を行うことで、ドメイン 0 上の IDS がドメイン U を監視することを可能にする。今後の課題は、chkrootkit で使用されている様々な外部コマンドに対応し、chkrootkit 全体をオフロードできるようにすることである