CacheShadow ファイルシステム：仮想ディスクと VM 内キャッシュの統合

土田 賢太朗[1]　光来 健一[1,2]

概要：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（VM）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 VM 上で動作させ，IDS だけ別の VM 上で動作させる手法である．しかし，オフロードした IDS が監視対象 VM の仮想ディスクを監視する際に，VM 内のページキャッシュなどのファイルシステムのキャッシュを考慮することができなかった．本稿では，ファイルシステムキャッシュと仮想ディスクを統合して VM の外から安全な監視を行えるようにする CacheShadow ファイルシステムを提案する．CacheShadow ファイルシステムは VM 内のメモリ上にあるファイルシステムキャッシュを VM イントロスペクションを用いて解析することで取得する．IDS が CacheShadow ファイルシステムにアクセスすると，キャッシュ上に最新のデータがある場合はそのデータを返し，なければ仮想ディスク上のデータを返す．我々は CacheShadow ファイルシステムを Xen および FUSE を用いて実装し，その性能を調べる実験を行った。

1. はじめに

近年インターネットに接続されたサーバへの攻撃を検知するための侵入検知システム（IDS）の重要性が高まっている．IDS は，ディスクを監視し，侵入の痕跡が見つかると管理者に通知を行うが，IDS 自身が攻撃を受けた場合，監視を行うことができなくなってしまう．この問題を解決するために，仮想マシン（VM）を用いて IDS をオフロードする手法が提案されている [1]．この手法では監視対象のシステムを VM を用いて動作させ，IDS だけを別の VM で動作させる．これにより監視対象のシステムが動作する VM の外で IDS を安全に動かすことが可能になっている．

しかしながら，オフロードした IDS は監視対象 VM の仮想ディスクを直接監視することになるため，監視対象 OS の保持しているファイルシステムのキャッシュは監視対象に含まれていなかった．OS はファイルの書き換えなどのファイルシステムに関する更新をまず，メモリ上のキャッシュに対して行い，一定時間が経過した後などにディスクに書き戻す．そのため，ファイルシステムキャッシュ上の更新データはディスクに書き戻されない限り，監視することができなかった．例えば，オフロードされた IDS は VM 内で行われたファイルへの更新やファイルの追加・削除などを即座に検出できない．ファイルシステムキャッシュがディスクに書き戻されるまでの時間は一般にはそれほど長くないが，VM に侵入した攻撃者はキャッシュの書き戻し時間を容易に長くすることができる．このような攻撃を行われると，攻撃者は IDS に検知されずに不正ファイルを使い続けることができる．

本稿では，VM を用いてオフロードした IDS が仮想ディスクとファイルシステムキャッシュを統合して監視できるようにする *CacheShadow* **ファイルシステム**を提案する．CacheShadow ファイルシステムは VM イントロスペクション [1] を用いて監視対象システムが動作する VM の OS カーネルのメモリを解析することでファイルシステムキャッシュに関する情報を取得する．ファイルシステムキャッシュとして，ページキャッシュ，ディレクトリキャッシュ，メタデータキャッシュを対象としている．CacheShadow ファイルシステムはこれらのキャッシュが存在する時にはキャッシュ上の情報を返し，存在しない時には仮想ディスク上の情報を返す．これにより，IDS をオフロードせずに監視対象システム上で動作させる場合と同じように最新の情報に基づいた監視を行うことができる．

我々は CacheShadow ファイルシステムを Xen 4.1.2 [2] のドメイン 0 上に実装した．Xen のドメイン U 上で監視対象システムを動作させ，ドメイン 0 に IDS をオフロードして動作させる．CacheShadow ファイルシステムは FUSE [3] を用いて実装され，ファイルやディレクトリの読み込

[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
JST，CREST

み時にキャッシュ上のデータを優先しながら仮想ディスク上のデータと統合する．この際に，Linux の page 構造体，dentry 構造体，inode 構造体を解析することでファイルシステムキャッシュに関する情報を取得する．CacheShadow ファイルシステムを用いて書き戻し時間を長くする攻撃を防ぐことができることが確認できた．また，ページキャッシュの解析にかかる時間はファイル数に比例し，ファイルの読み込み時間は従来の 1.3 倍程度になることが分かった．

以下，2 章でディスク監視を行う IDS のオフロードにおけるファイルシステムキャッシュの影響について述べ，3 章で CacheShadow ファイルシステムを提案する．4 章で CacheShadow ファイルシステムの実装について述べ，5 章で実験を示す．6 章で関連研究に触れ，7 章で本稿をまとめる．

2. IDS オフロードにおけるディスクの監視

VM を用いた IDS のオフロードは，監視対象のシステムを VM 上で動作させ，IDS だけを別の VM で動作させることにより IDS を守る手法である．監視対象システムを動作させる VM をサーバ VM，IDS を動作させる VM を IDS VM と呼ぶ．この手法を用いることで，サーバ VM では IDS を動作させる必要がなくなるため，サーバ VM に侵入されたとしても IDS を攻撃されることはない．そのため，IDS VM 上で動作する IDS はサーバ VM の監視を継続することができ，安全に侵入を検知することができる．IDS VM にも侵入される可能性があるが，IDS VM では IDS 以外のサービスをできるだけ動作させないようにすることで，攻撃を受けにくくすることができる．

IDS VM にオフロードされた IDS はサーバ VM の仮想ディスクを監視することで侵入の痕跡を見つける．IDS VM はサーバ VM の仮想ディスクを直接マウントし，マウント先のディレクトリに対して IDS を実行する．例えば，ディスクの完全性のチェックを行う Tripwire の場合，あらかじめ仮想ディスクに対して正常時の状態を記録したデータベースを作成しておく．そして，定期的に仮想ディスクを検査してデータベースとの照合を行う．もし，ファイルの状態がデータベースと一致しなければ改竄されたとみなすことができる．

従来，オフロードされた IDS はサーバ VM の仮想ディスクを監視する際にサーバ VM のファイルシステム内部のキャッシュを考慮していなかった．例えば，ページキャッシュはディスクから読み込んだファイルの内容を一時的に保持しておくために使われ，OS カーネルのメモリ上に作成される．アプリケーションはページキャッシュ上のファイルを読み書きすることで低速なディスクへのアクセスを行うことなく，高速にファイルアクセスを行うことができる．アプリケーションがファイルを書き換えるとまず，ページキャッシュ上のファイルが更新され，一定時間が経過した後などにディスクへ書き戻される．IDS VM にオフロードされた IDS は，サーバ VM の仮想ディスクを直接参照するため，サーバ VM 内のページキャッシュ上にある最新のファイルの内容を監視することができなかった．

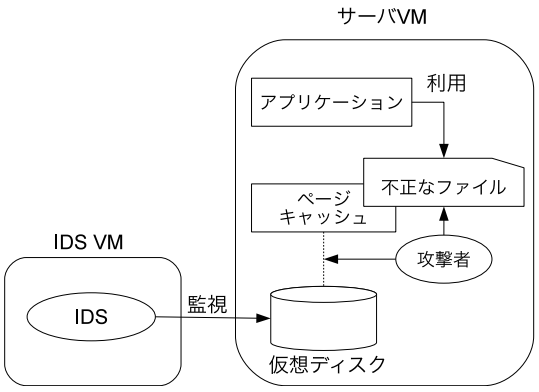

図 1 ページキャッシュを利用した攻撃

サーバ VM のファイルシステムキャッシュを参照できないことにより，様々な検知漏れが起こる可能性がある．例えば，ページキャッシュを参照しないとログファイルに追記された最新のアクセスログをチェックすることができない．ディレクトリキャッシュを参照しないと新たにインストールされた不正なファイルを見逃すことになる．また，ファイルの更新時刻などのメタデータキャッシュを参照できないと，ファイルが更新されていることに気がつかず，改竄が行われていないと誤って判断してしまう可能性がある．

ファイルシステムキャッシュは一定時間ごとにディスクに書き戻されるため，上記の問題は通常は深刻な問題にならない．しかし，ディスクへの書き戻しを阻害されると大きな問題となる．例えば Linux 2.6 では pdflush デーモンがファイルシステムキャッシュの書き戻し処理を行なっているが，書き戻しの間隔は proc ファイルシステムで設定することができる．この設定は管理者権限があれば変更することができる．そのため，書き戻し間隔を長くするだけで，ファイルシステムキャッシュへの変更がディスクに反映されない時間を長くすることができる．この状態で図 1 のようにサーバ VM 上の攻撃者がファイルを不正に書き換えても，オフロードした IDS は長時間それを検知することができない．その一方で，サーバ VM 内のアプリケーションにはファイルシステムキャッシュ上の不正なファイルを使わせ続けることができる．サーバ VM の管理者権限を取得した攻撃者は他にも様々な攻撃を行うことができるが，この攻撃手法はシステムへの変更を一切加えていないように見える点で非常に強力である．

また，オフロードされた IDS はメモリをディスクとして扱う tmpfs 上のファイルを監視することもできない．tmpfs はページキャッシュ上にデータを保持し，そのデータはディスクに書き戻されることがないためである．攻撃者は

キャッシュが書き戻されるまでの時間に関係なく，tmpfs 上に攻撃用ファイルを置くことで検知されないようにすることができる．Ubuntu では tmpfs は/run などにマウントされている．

3. CacheShadow ファイルシステム

本稿では，サーバ VM の仮想ディスクとファイルシステムキャッシュを統合して監視を行えるようにする CacheShadow ファイルシステムを提案する．CacheShadow ファイルシステムを用いる場合の IDS オフロードのシステム構成を図 2 に示す．CacheShadow ファイルシステムは IDS VM 上で動作し，IDS に対してサーバ VM のファイルシステムに関する情報を提供する．CacheShadow ファイルシステムはファイルシステムキャッシュに関する情報を VM イントロスペクション [1] を用いてサーバ VM のメモリから取得する．そして，ファイルシステムキャッシュ上に最新の情報があればその情報を返し，なければ仮想ディスク上の情報を返す．このようにして，IDS VM 上の IDS はサーバ VM 上で動作している場合と同様に，ファイルシステムに関する最新の情報をより安全に得ることができる．

CacheShadow ファイルシステムは，ファイルシステムキャッシュとしてページキャッシュ，ディレクトリキャッシュ，メタデータキャッシュを扱う．ページキャッシュはファイルのデータそのもののキャッシュであり，メモリページ単位で管理される．ファイルシステムはファイルのデータにアクセスする時にまず，ページキャッシュを探し，なければディスクから読み込む．CacheShadow ファイルシステムでも同様に，まずサーバ VM 内にページキャッシュがあるかどうかを調べ，あれば見つかったページキャッシュ上のデータを返し，なければ仮想ディスクからファイルを読み込む．これにより，ファイルの改竄を即座に検知できるようになり，最新のログファイルをチェックできるようになる．

ディレクトリキャッシュはディレクトリエントリのキャッシュである．ファイルシステムはパス名を解決する際にまず，ディレクトリキャッシュを調べ，ディレクトリエントリがキャッシュされていれば再帰的にパス名解決を続ける．キャッシュにされていなければディスクからディレクトリエントリをキャッシュに読み込み，パス名解決を行う．CacheShadow ファイルシステムでもまず，サーバ VM のディレクトリキャッシュを探索し，見つからなければ仮想ディスク上のディレクトリエントリを用いる．このようにすることで，サーバ VM で追加・削除されたファイルやディレクトリを検知できるようになる．

メタデータキャッシュはファイルやディレクトリの更新時刻やアクセス権などのメタデータのキャッシュである．CacheShadow ファイルシステムがメタデータにアクセスする時にはまず，サーバ VM 内のメタデータキャッシュを探す．メタデータが見つかればその情報を返し，見つからなければ仮想ディスクから読み込む．これにより，取得されるファイルやディレクトリの更新時刻を最新に保つことができ，アクセス権の変更も即座に検知することができる．

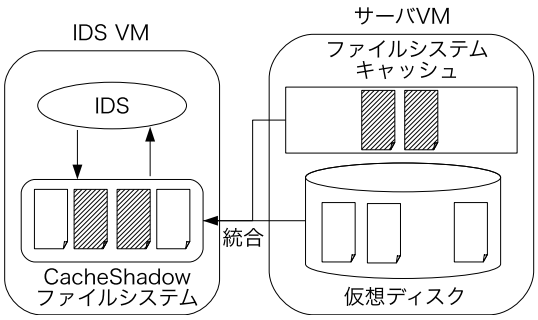

図 2 CacheShadow ファイルシステム

4. 実装

CacheShadow ファイルシステムを Xen 4.1.2 のドメイン 0 上に実装した．通常の VM 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 VM であるドメイン 0 を IDS VM とした．サーバ VM のゲスト OS として準仮想化 Linux 2.6.39.3 を対象とした．

4.1 ファイルシステム キャッシュの解析

4.1.1 ページキャッシュの解析

CacheShadow ファイルシステムではサーバ VM の物理メモリを管理しているページ構造体を解析することでページキャッシュ情報を取得する．ページ構造体には対応するメモリページがどのような用途で使われているかという情報が格納されている．まず，サーバ VM のページ構造体の配列が置かれているメモリ領域を IDS VM にマップする．メモリモデルとしてフラットメモリまたは仮想メモリマップをサポートしたスパースメモリを用いている Linux カーネルでは，ページ構造体の配列は仮想アドレス空間上で連続している．その先頭の仮想アドレスは固定であり，カーネルのシンボル情報から取得することができる．配列のサイズはサーバ VM に割り当てた物理メモリのサイズから計算することができる．一方，メモリモデルとして不連続メモリまたは仮想メモリマップをサポートしないスパースメモリを用いている場合は，page 構造体の配列が連続していないため，メモリマップは少し複雑になる．現在の実装では，仮想メモリマップをサポートしたスパースメモリについてのみ対応している．

次に，page 構造体の配列の要素を順番に調べて，対応するメモリページがページキャッシュとして使われているかどうかの判別を行う．ページ構造体はメモリページの用途を表 1 のようなフラグで管理している．例えば，特殊な用途で使うために予約済みのページの場合には PG_reserved がセットされる．しかし，ページキャッシュかどうかを直

表 1　page 構造体のフラグ（抜粋）

フラグ名	内容
PG_slab	ページをスラブで使用
PG_reserved	予約済みページ
PG_swapcache	スワップキャッシュとして使用
PG_tail	ラージページの先頭以外のページ
PG_pinned	Xen によってピン留めされたページ

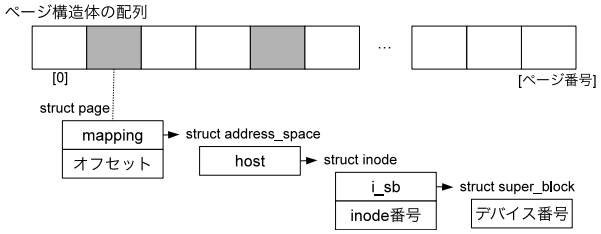

図 3　page 構造体の解析

接示すフラグは存在しないため，複数のフラグを組み合わせて消去法でページキャッシュとして使われているページを割り出す．ページキャッシュには表 1 のフラグはいずれもセットされないため，これらのフラグがセットされていないページをページキャッシュと判定する．

ページキャッシュと判定されたページについて，図 3 のようにページ構造体を解析することでキャッシュされているファイルが存在するデバイスの番号，ファイルの inode 番号，ファイルデータのページ単位でのオフセットを取得する．まず，page 構造体に格納された address_space 構造体をたどり，そこから inode 構造体を見つける．inode 構造体の中に inode 番号が格納されており，inode 構造体からたどられる super_block 構造体にデバイス番号が格納されている．これらの構造体が NULL ポインタであった場合には，ページキャッシュではないと判定する．ファイルのオフセットはページ構造体の中に格納されている．このようにして取得したページキャッシュに関する情報は，デバイス番号， inode 番号，オフセットから対応するファイルキャッシュのページ番号を見つけられるようにハッシュ表に格納する．

ただし，ページキャッシュのデータがディスクのデータと同一の場合にはこのような解析を省略することができる．ディスクに最新の情報があれば，ディスク上のファイルのデータを返せばよいためである．この判定には，page 構造体の PG_dirty フラグを用いることができる．このフラグがセットされていなければページキャッシュのデータが更新されていないことを意味する．

4.1.2　ディレクトリキャッシュの解析

Linux ではディレクトリのデータもページキャッシュとしてキャッシュされている．そのため，ページキャッシュの探索と同様にしてディレクトリに関するキャッシュを見つけることが可能である．しかし，ディレクトリのデータ構造はファイルシステムごとに大きく異なっているため，解析方法がサーバ VM で用いられているファイルシステムに依存してしまう．そこで，Linux カーネル内で行われている処理と同様に，ファイルやディレクトリのパス名からディレクトリキャッシュを探索する．

表 2　inode 構造体にキャッシュされるメタデータ（抜粋）

メンバ	内容
i_atime	更新時刻
i_mode	アクセス保護
i_uid	所有者のユーザ ID
i_size	ファイルの大きさ

Linux では dentry 構造体を用いてディレクトリに関する情報を木構造でキャッシュしている．ディレクトリキャッシュを探索するために，まず init プロセスの task_struct 構造体からルートディレクトリの dentry 構造体を取得する．Linux では dentry 構造体を用いてディレクトリに関する情報が木構造でキャッシュされている．dentry 構造体は子の dentry 構造体と同じ親を持つ兄弟の dentry 構造体の 1 つにリンクされている．そのため，対象のパス名に沿って，ルートディレクトリの dentry 構造体から子の dentry 構造体を調べていくことで，目的のディレクトリエントリがキャッシュされていれば到達することができる．

ディレクトリキャッシュは削除されたファイルやディレクトリの dentry 構造体も保持している．実体が削除されているかどうかは，対応する dentry 構造体が dentry のハッシュから削除されているかどうか，また，どの inode 構造体にもリンクされなくなっているかどうかで判断することができる．実体が削除されている場合には，その dentry 構造体はディレクトリキャッシュの探索の際に存在しないものとして扱う．

4.1.3　メタデータキャッシュの解析

ファイルやディレクトリのメタデータは inode 構造体にキャッシュされている．inode 構造体の中にはメタデータキャッシュとして表 2 のような情報が格納されている．inode 構造体は dentry 構造体からたどることができるため，ディレクトリキャッシュを探索して dentry が見つかればメタデータキャッシュも取得できる．

4.2　ファイルシステムキャッシュの統合

CacheShadow ファイルシステムは FUSE を用いて実装された bindfs [4] をベースに開発した．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．bindfs は指定したディレクトリをマウント先からアクセスすることを可能にするファイルシステムである．bindfs は常に指定したディレクトリを参照するが，CacheShadow ファイルシステムではサーバ VM のファイルシステムキャッシュがあればそちらを優先的に参照する．

CacheShadow ファイルシステムからサーバ VM の仮想

ディスクにアクセスできるようにするために，まず，サーバ VM の仮想ディスクを IDS VM にマウントする．このマウントには IDS VM 内の OS のファイルシステムの機能を用いる．この仮想ディスクはサーバ VM でもマウントされており，同時にマウントしても整合性を保てるようにするために読み込み専用でマウントする．次に，仮想ディスクをマウントしたディレクトリを指定して CacheShadow ファイルシステムでマウントする．

4.2.1 ページキャッシュの統合

CacheShadow ファイルシステムでは，ファイルの読み込み時に呼ばれる FUSE の read 関数の中でファイルのデータの統合を行う．まず，読み込み対象のファイルのパス名を基にディレクトリキャッシュを探索し，メタデータキャッシュからデバイス番号と inode 番号，ファイルサイズを取得する．次に，inode 番号とファイルのオフセットをキーとして 4KB ごとに 4.1.1 節の解析で作成したハッシュ表を検索する．ハッシュ表に登録されていれば，ハッシュ表から得られたページ番号に対応するページをマップし，そのページキャッシュに格納されているデータをプロセスのバッファにコピーする．ハッシュ表に登録されていない場合は仮想ディスク上のファイルを 4KB 読み込み，バッファにコピーする．この処理をバッファが一杯になるか，ファイルの終わりまで繰り返す．ファイルの終わりの判定には，メタデータキャッシュ上のファイルサイズを用いる．

4.2.2 ディレクトリキャッシュの統合

CacheShadow ファイルシステムでは，ディレクトリの読み込み時に呼ばれる FUSE の readdir 関数内ではディレクトリエントリの統合を行う．まず，パス名を基にディレクトリキャッシュを探索して対象の `dentry` 構造体を取得する．次に，そのすべての子の `dentry` 構造体の情報を FUSE の提供するバッファに格納する．このとき，実体が削除されている `dentry` 構造体の情報はバッファに格納せず，削除フラグを立てて管理する．

次に，仮想ディスクからディレクトリエントリを読み込む．ディレクトリキャッシュになかったエントリの情報だけを FUSE のバッファに格納し，既にキャッシュから取得しているエントリは重複して格納しないようにする．また，削除済みのフラグが立っているエントリは，仮想ディスクにエントリがあったとしてもバッファに格納しないようにする．

4.2.3 メタデータキャッシュの統合

CacheShadow ファイルシステムでは，getattr 関数内でメタデータキャッシュの統合を行う．まず，ディレクトリキャッシュを探索してパス名から対象の `dentry` 構造体を取得する．`dentry` 構造体からリンクされている `inode` 構造体が取得できればメタデータキャッシュから表 2 のようなメタデータを返す．キャッシュ内に `dentry` 構造体や `inode` 構造体が存在しない場合は，lstat システムコールを用いて仮想ディスクからメタデータを読み込む．

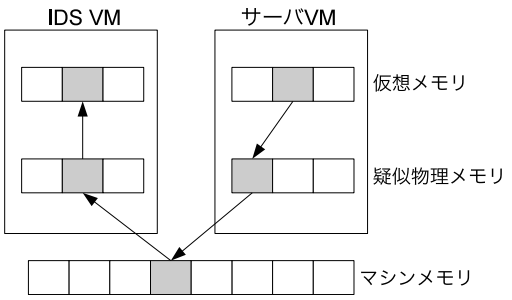

図 4 サーバ VM のメモリ解析

4.3 サーバ VM のメモリ解析

CacheShadow ファイルシステムはサーバ VM の OS カーネルのメモリを解析することでファイルシステムキャッシュに関する情報を取得する．図 4 のようにして，まずサーバ VM のメモリ上にあるページテーブルを参照してサーバ VM 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，サーバ VM のメモリ上に作られる P2M テーブルを参照して，疑似物理アドレスからマシンアドレスを求める．このマシンアドレスを含むメモリページを IDS VM にマップし，そのアドレスに対してアクセスすることで，IDS VM からサーバ VM の指定したメモリを参照することができる．

CacheShadow ファイルシステムではこのアドレス変換のオーバヘッドを減らすために，IDS VM 上にアドレス変換のキャッシュを作成する．このキャッシュを用いることで，同じデータに対して何度もアドレス変換を行う必要がなくなる．例えば，ページキャッシュを解析する際に `inode` 構造体などを調べるが，キャッシュされているファイルが同じであれば `inode` 構造体も同じである．また，ディレクトリキャッシュのルートディレクトリに近い `dentry` 構造体は何度も参照される．これらのアクセスがキャッシュにより高速化される．

4.4 議論

現在の実装では，`page` 構造体の配列から解析を行っているため，特定のファイルのページキャッシュだけを探すのには向いていない．すべてのページキャッシュの解析を行ってから目的のページキャッシュを見つける必要がある．Tripwire のようにディスク全体の整合性をチェックする場合にはほとんどすべてのページキャッシュを参照するため効率がよいが，少数のファイルだけを監視する場合には非常に効率が悪い．

また，一括して解析を行うと，解析している間，および，その情報を用いて監視を行っている間，監視対象のサーバ VM を停止させ続ける必要がある．これはサーバ VM が動

くとページキャッシュに関する情報も更新されてしまうためである．この問題を解決するには，ある時点でのサーバ VM のスナップショットを取って，スナップショットに対して監視を行う方法が考えられる．この方法では，サーバ VM を停止させる時間をスナップショットを取る時間だけに短縮することができるが，サーバ VM の最新の状態を監視できない場合がある．

別の方法としては，Linux カーネル内と同様にして，ファイルのパス名をたどって Radix Tree を解析することで目的のページキャッシュを見つける方法が考えられる．この方法を用いれば，解析を行っている間だけサーバ VM を停止させればよく，少数のファイルだけを監視する場合にも効率よく解析を行うことができる．しかし，ディスク全体を監視する場合には逆に効率が悪くなる．また，CacheShadow ファイルシステム上でファイルをオープンしてから読み込むまでの間にサーバ VM 上でファイルが削除される可能性などを考えると，一貫性を保ちながら監視を行うのが難しい場合がある．

5. 実験

CacheShadow ファイルシステムの有効性を確認し，性能を調べるための実験を行った．この実験は，Intel Xeon 3.60GHz の CPU，16GB のメモリ，SATA 500GB の HDD を搭載したマシンで行った．Xen 4.1.2 を用い，IDS VM とサーバ VM の OS は Linux 2.6.39.3 であった．

5.1 有効性の確認

CacheShadow ファイルシステムを用いることで，サーバ VM のキャッシュの書き戻し時間を変更する攻撃が防げることを確かめる実験を行った．まず，サーバ VM 上にテキストファイルを作成し，ページキャッシュからディスクへの書き戻し時間を十分に長く設定して書き戻しが起きないようにした．その状態でファイルの中身を書き換え，ファイルをサーバ VM から参照した場合，IDS VM から従来手法で仮想ディスクのみを参照した場合，IDS VM から CacheShadow ファイルシステムを使って参照した場合について，参照できる内容を調べた．

サーバ VM 上で参照した場合は書き換えた内容を参照することができた．しかし，IDS VM から仮想ディスクのみを参照した場合は，書き換える前の古い内容しか参照することができなかった．一方，CacheShadow ファイルシステムを用いた場合，書き換えた内容を参照することができた．このことから，CacheShadow ファイルシステムによりキャッシュ上の最新のファイルのデータを読み込むことができることを確認できた．

また, tmpfs 上にファイルを作成した場合, C acheShadow ファイルシステムを用いることでページキャッシュから tmpfs 上のファイル情報が取得できることも確認した．

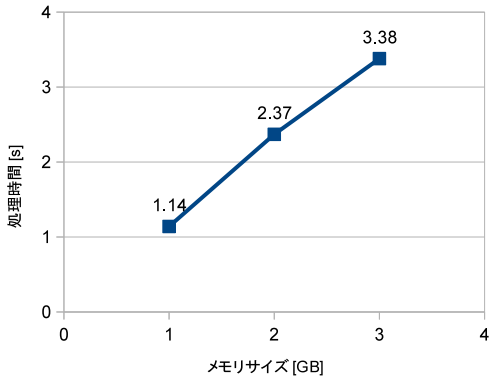

図 5 ページキャッシュの最小の解析時間

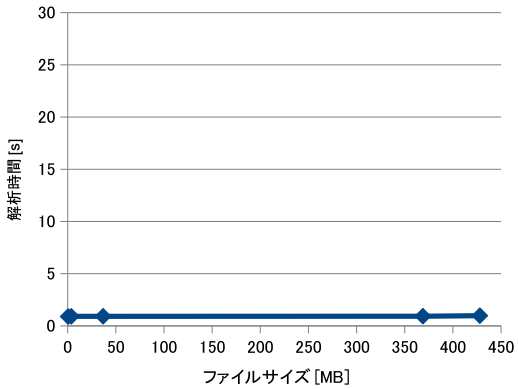

図 6 1 つのファイルをキャッシュさせた場合の解析時間

5.2 ページキャッシュ情報の解析にかかる時間

CacheShadow ファイルシステムがサーバ VM のページキャッシュ情報を解析するのにかかる時間を測定した．まず，サーバ VM に割り当てるメモリサイズを変更しながら，ディスクに書き戻されていないダーティなページキャッシュがほとんどない状態で測定を行った．この測定結果を図 5 に示す．この実験結果より，1 秒程度で 1GB のメモリを解析できていることが分かる．ディスクへの書き込み頻度が低く，書き戻し時間の設定が標準的なシステムでは，数秒程度で VM メモリ全体の解析を行うことができる．

次に，サイズを変更しながら一つのファイルを作成し，ページキャッシュの解析時間を測定した．ページキャッシュの書き戻し時間を長く設定して，作成されたページキャッシュがダーティであり続け，解析の対象となるようにした．実験結果を図 6 に示す．この場合，ファイルサイズの影響を受けずにほぼ一定の解析時間となった．これは，4.3 節で述べたアドレス変換のキャッシュにより，ページキャッシュと判別された `page` 構造体から `inode` 構造体などをたどる際のアドレス変換が 1 回で済むためである．

同様にして，4KB のファイルを 1〜10 万個作成し解析時間を測定した．実験結果は図 7 のようになり，ファイルの数に比例して解析時間が増加することが分かった．これは

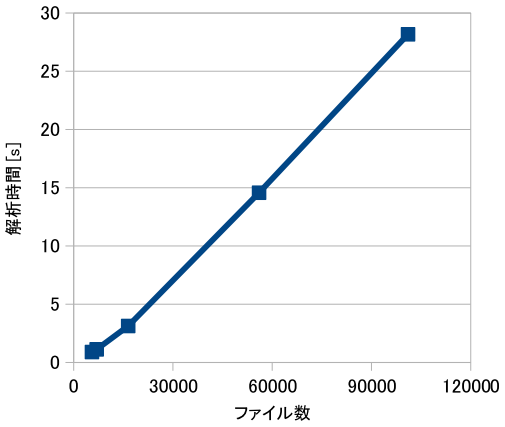

図 7 複数の 4KB のファイルをキャッシュさせた場合の解析時間

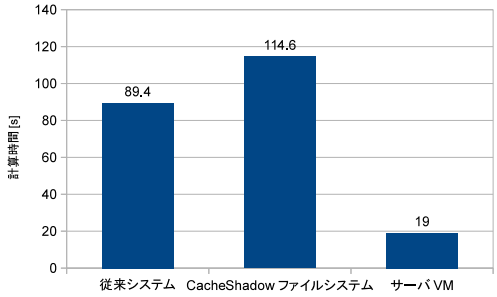

図 8 ファイルの読み込み性能

ページキャッシュと判別された全ての page 構造体からそれぞれ異なるファイルの情報を解析しなければならないことによるオーバーヘッドである．実際には，10 万個の異なるファイルがキャッシュされることは少ないため，解析時間はそれほど長くならないと考えられる．

5.3 ファイルの読み込み性能

CacheShadow ファイルシステムを用いて 4MB のファイルのハッシュ値を md5 コマンドで計算するのにかかる時間を測定した．実験結果を図 8 に示す．CacheShadow ファイルシステムを用いた場合，bindfs を用いてサーバ VM の仮想ディスクを参照する従来システムの約 1.3 倍の時間がかかっている．これはページキャッシュからデータを読み込むためにページごとにメモリマップを行うことによるオーバーヘッドである．

一方、サーバ VM 内部で同様の処理を行う時間と比べると，CacheShadow ファイルシステムでは 6 倍の時間がかかっている．これはサーバ VM がキャッシュにしかアクセスしないため，非常に高速にファイルアクセスが行えるためである．

6. 関連研究

VMwatcher [5] や Storage IDS [6]，VM Shadow [7] は VM の外で IDS を動作させて，VM 内の OS の監視を行うことができる．VM watcher，VM Shadow は別の VM からディスクを監視し，StorageIDS は NFS サーバなどのディスク側で IDS を動作させ監視を行う．これらのシステムでは，監視をする対象はディスクだけであるため．ファイルシステムキャッシュを考慮した監視を行うことはできない．

HyperSpector [8] もサーバと IDS を別々の仮想マシンで動作させるシステムである．しかし，HyperSpector は OS レベルでの仮想化を用いているためサーバ VM と IDS VM は 1 つの OS を共有しており，OS 内部のファイルシステムキャッシュも共有できる．それゆえ，IDS VM はファイルシステムキャッシュを考慮してサーバ VM のディスクの監視を行うことができる．CacheShadow ファイルシステムでは IDS VM とサーバ VM が別々の OS を使うシステムレベルの仮想化を前提としているため，IDS VM がサーバ VM のファイルシステムキャッシュを参照できるようにするために VM イントロスペクションを用いている．システムレベルの仮想化を用いることで，サーバ VM と IDS VM をより強固に隔離することができる．

Volatility [9] はダンプしたメモリ上の tmpfs を解析しローカルディスク上にコピーすることができる．Volatility では Linux でページキャッシュを管理している Radix Tree を探索して目的のページキャッシュを取得している．Volatility は tmpfs 全体をコピーして読み込むことはできるが，ディスクを用いるファイルシステムの場合に仮想ディスクと統合することはできない．

7. まとめ

本稿では，VM を用いてオフロードした IDS が監視対象 VM 内のファイルシステムキャッシュと仮想ディスクを統合して監視を行えるようにする CacheShadow ファイルシステムを提案した．CacheShadow ファイルシステムは，VM イントロスペクションを用いることで，監視対象 OS からページキャッシュ，ディレクトリキャッシュ，メタデータキャッシュに関する情報を取得する．IDS が CacheShadow ファイルシステムにアクセスした時には，キャッシュがあればキャッシュの最新情報を返し，なければ仮想ディスクの情報を返す．我々は Xen 上に CacheShadow ファイルシステムを実装し，ファイルの読み込みについて統合が行えていることを確認した．

今後の課題は，CacheShadow ファイルシステムの読み込みのオーバーヘッドを減らすことや，実際の IDS で性能測定を行うことである．

参考文献

[1] Garfinkel, T. and Rosenblum, M.: A Virtual Machine Introspection Based Architecture for Intrusion Detection,

Network and Distributed Systems Security Symp, (2003).
[2] P.Barham, B.Dragovic, K.Fraser, S.Hand, T.Harris, A.Ho, R.Neugebauer, I.Pratt, and A.Warfield.: Xen and the Art of Virtualization, *In Proc. of the 19th Symposium on Operating Systems Principles*, (2003).
[3] M.Szeredi: Filesystem in Userspace, 入手先 ⟨http://fuse.sourceforge.net/⟩.
[4] bindfs - Mount a directory to another location and alter permission bits. 入手先 ⟨http://bindfs.org/⟩.
[5] Jiang, X., Wang, X. and Xu, D.: Stealthy Malware Detection through VMM-based ”Out-of-the-box ” Semantic View Reconstruct, *In Proc. Conf. Computer and Communications Security*, (2003).
[6] Adam G. Pennington, John D. Strunk, John Linwood Grifn, Craig A.N. Soules, Garth R. Goodson, Gregory R. Ganger.: Storage-based Intrusion Detection: Watching storage activity for suspicious behavior *In Proc. 12th USENIX Security Symposium*, (2003).
[7] 飯田貴大, 光来健一. VM Shadow: 既存 IDS をオフロードするための実行環境. 第 119 回 OS 研究会, (2011).
[8] Kourai, K. and Chiba, S.: HyperSpector: Virtual Distributed Monitoring Environments for Secure Intrusion Detection, *In Proc. Conf. Virtual Execution Environments*, (2005).
[9] volatility - An advanced memory forensics framework 入手先 ⟨https://code.google.com/p/volatility/⟩.