Vol.2012-OS-122 No.10
2012/8/1

ファイルシステムキャッシュを考慮したIDSオフロード

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 が監視対象システムの仮想ディスクを監視する際に，ページキャッシュなどのファイルシステム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する．CacheShadow ファイルシステムは VM イントロスペクションを用いて監視対象 OS のメモリ内部にあるファイルシステムキャッシュを取得する．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 上に実装し，IDS がページキャッシュのデータを参照できるようにした．

キーワード：仮想マシン，ディスク監視，ファイルシステム，ページキャッシュ

Offloading IDSes Considering the Filesystem Cache

Tsuchida Kentaro[1]　Kourai Kenichi[1,2]

Abstract: To execute intrusion detection systems (IDSes) securely, offloading IDSes with virtual machines (VMs) has been proposed. This technique runs a monitored system on one VM and executes only IDSes on another VM. However, offloaded IDSes cannot consider the filesystem cache such as the page cache because they directly monitor the virtual disks of the monitored system. This paper proposes the CacheShadow filesystem, which integrates the filesystem cache with the virtual disks. The CacheShadow filesystem obtains the filesystem cache in the memory of a monitored system using VM introspection. It returns the latest data on the filesystem cache if the cached data is found; otherwise, it returns the data in the virtual disks. We have implemented the CacheShadow filesystem in Xen and confirmed that IDSes could access the data on the page cache.

Keywords: Virtual Machine, Disk Monitoring, filesystem, page cache

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仮想マシンの外で 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 は監視対象システムの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

[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

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，ファイルへの更新やファイルの追加・削除などをオフロードされた IDS は即座に検出できない．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，攻撃者はページ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ある時にはキャッシュの情報を返し，ない時には仮想ディスクの情報を返す．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 4.1.1[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，Linux のページ構造体を解析することでページ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ページキャッシュ情報の取得に関する実験を行ったところ，監視対象システムのページキャッシュをすべて見つけることができることが分かった．また，ページキャッシュ情報の取得にかかる時間はメモリサイズとキャッシュサイズに比例することが分かった．

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りえるが，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．

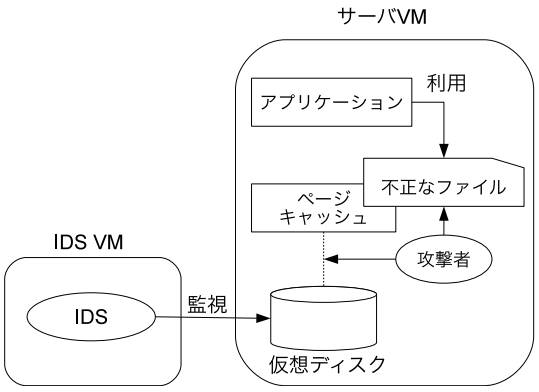

図 1 ページキャッシュを利用した攻撃

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検査してデータベースとの照合を行う．もし，ファイルの内容がデータベースと一致しなければ改ざん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．

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，ページキャッシュが参照できないとログに記録されている不正アクセスのチェックを行うことができない．

ディレクトリに関するキャッシュを参照できないと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に関するキャッシュを参照できないと，ファイルの改ざん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が，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．

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 上で動作させた場合と同様に，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について取り扱う．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の内容を返し，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，ファイルの改ざんを即座に検知できるようになり，最新のログを調べられるようになる．

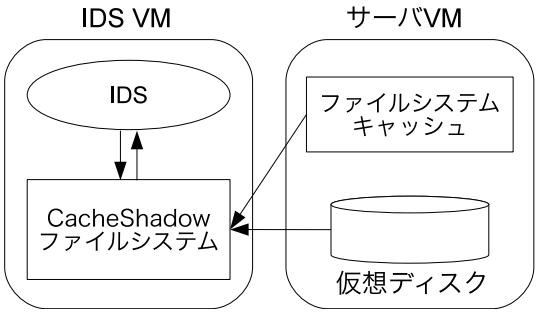

図 2 CacheShadow ファイルシステム

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，キャッシュされていればその情報を使って再帰的にパス名解決を続ける．キャッシュになければディスクからディレク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検索し，見つからなければ仮想ディスクのディレクトリ情報を用いる．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ファイルやディレクトリの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検索する．メタデータが見つかればその情報を返し，見つからなければディスクから読み込む．これにより，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．

4. 実装

CacheShadow ファイルシステムを Xen 4.1.1 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．サーバ VM のゲスト OS として Linux 2.6.39.3 を対象とした．

4.1 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 3 のようにして，まずサーバ VM の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，P2M 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．このマシンアドレスを含むメモリページを IDS VM

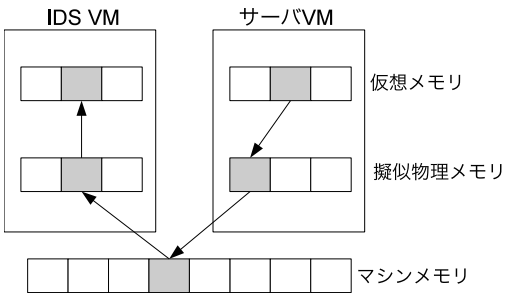

図 3 サーバ VM のメモリ解析

にマップすることで，IDS VM からサーバ VM の指定したメモリアドレスを参照することができる．

4.2 ページキャッシュの解析

Linux ではページキャッシュはファイル単位で管理されている．そのため，まずファイルのパス名をたどってファイル構造体を見つける必要がある．さらに，ページキャッシュは Radix Tree と呼ばれるデータ構造で管理されているため，Radix Tree を探索して目的のページキャッシュを見つける必要がある．これらのデータ構造はかなり複雑なため，IDS VM から解析するのは容易ではない．

そこで，今回の実装ではサーバ 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を見つける方法を採用した．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，OS 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，ページ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，ページ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．

表 1 ページキャッシュのフラグ

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

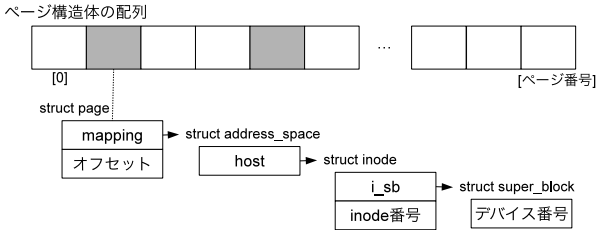

図 4 ページ構造体の解析

ページキャッシュと判定されたページについて，図 4 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，ファイルのデータのページ単位でのオフセットを取得する．まず，ページ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，ページ構造体に 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．

4.3 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の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，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．

表 2 inode 構造体のメタデータ

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

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．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の探索はまだ未実装である．

4.4 メタデータキャッシュの解析

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．ファイルやディレクトリの 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が見つかればメタデータキャッシュも同時に見つけることができる．

4.5 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 OS のファイルシステムの機能を用いる．マウントする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 ファイルシステムを用いてマウントする．

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 read 関数の中でファイルのデータの読み込み元を切り替える．読み込みを行うファイルのデバイス番号，inode 番号，オフセットを取得し，これらをキーとしてページキャッシュの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，read 関数が読み込んだデータを返す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読み込む．読み込むオフセットとサイズがページキャッシュの境界をまたぐ場合は，それぞれについて同様の処理を行う．

ファイルのオープン時に呼ばれる open 関数などでパス名を指定してアクセスする場合には，ディレクトリキャッシュを探索して inode 構造体を特定する．指定されたパス名がディレクトリキャッシュに存在しない時には，仮想ディスクから情報を取得する．この部分については現在のところ，未実装である．

4.6 議論

現在の実装では，ページ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動作を確認するための実験を行った．この実験は，Intel Core i7 2.93GHz の CPU，4GB のメモリ，SATA 500GB の HDD を搭載したマシンで行った．Xen 4.1.1 を用い，ドメイン 0 とドメイン U の OS は Linux 2.6.39.3 であった．

5.1 ページキャッシュ上のデータの読み込み

CacheShadow ファイルシステムを用いることで、サーバ VM のページキャッシュ上にあるファイルのデータを読み込めることを確認する実験を行った．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た。このファイルをサーバ 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、参照できる内容を確認した。

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、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．

5.2 取得できたページキャッシュのページ数

CacheShadow ファイルシステムがすべてのページキャッシュを見つけられていることを確認するために、CacheShadow ファイルシステムが取得できたページキャッシュのページ数とサーバ VM 内のページキャッシュのページ数を比較したサーバ VM 内のページキャッシュのページ数は/proc/vmstat 内の nr_file_pages の値となる。この実験ではドメイン U に 1GB のメモリを割り当て，ページキャッシュのページ数を 820、25491、219699 の 3 種類に変更して比較を行った．その結果、CacheShadow ファイルシステムが取得できたページキャッシュのページ数はサーバ VM 内での情報と一致した。このことから、CacheShadow ファイルシステムはページキャッシュを正しく取得できたと考えられる．

5.3 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、数秒程度でメモリ全体の解析を行うことができる。

次に、ページキャッシュのサイズを増やしながらページキャッシュの解析にかかる時間を測定した。この実験ではすべてのページキャッシュの解析を行った。実験結果は図

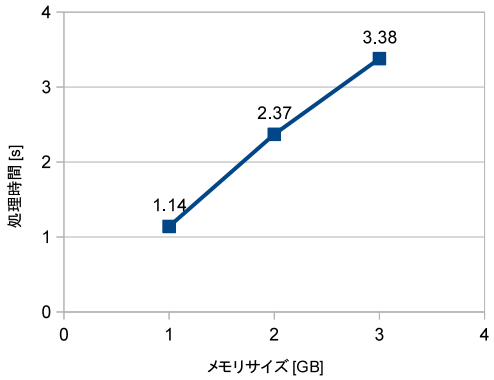

図 5 ダーティなページキャッシュがほぼ 0 の時の解析時間

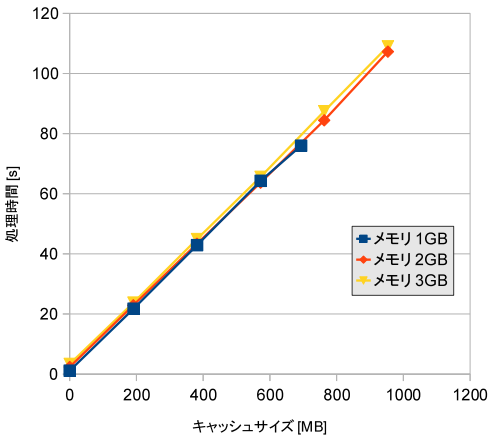

図 6 ページキャッシュ情報を解析するのにかかる時間

6 のようになり、10MB のページキャッシュを解析するのに約 1 秒かかることが分かった。これはページキャッシュごとにいくつもの構造体をマップしなければならないためである。ディスク全体の監視に非常に時間がかかる場合にはこの解析時間はそれほど問題にならないが、そうでない場合には解析のオーバヘッドが大きい。ただし、一般的にはダーティなページキャッシュのサイズはそれほど大きくないことが多いと考えられる。

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．

7. まとめ

本稿では，VM を用いてオフロードした IDS が監視対象システム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，ページキャッシュからファイルのデータを読み込むことができることを確認した．

今後の課題は，まず，ページキャッシュを解析するよりよい方法について検討することである．4.6 節で述べたように，いずれの手法でも利点と欠点がある．その検討結果を踏まえて，CacheShadow ファイルシステムの実装を完成させることである．現在のところ，ページキャッシュの情報を取得することができているが，ディレクトリキャッシュ，メタデータキャッシュの探索は行えていない．また，FUSE を用いた実装もページキャッシュに関して限定的に行えているだけであるため，すべての種類のキャッシュを考慮してより一般的な実装を行う必要がある．

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://code.google.com/p/bindfs/⟩.
[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).