Vol.2012-OS-123 No.8
Vol.2012-EMB-27 No.8
2012/12/5

中立的VMMによる動画像を対象とした著作権保護

小清水 滋[1]　新城 靖[1]　板野 肯三[1]　榮樂 英樹[2]　松原 克弥[2]

概要：現在、PC への動画配信が普及して久しいが、依然として著作権保護の問題が残されている。本研究ではソフトウェア的に破れないことを保証し、かつユーザが安心して利用できる著作権保護手法の提案と実装を行う。本研究では、ユーザと権利者に中立な仮想計算機モニタを用いる。中立的仮想計算機モニタ上のユーザ用の OS とは隔離されたメモリ空間でメディアプレイヤを動かし、動画像の再生中はメディアプレイヤに画面デバイスを専有させる。メディアプレイヤと配信サーバ間では暗号通信を行う。中立的仮想計算機モニタの非改竄はリモートアテステーションを用いユーザと権利者双方から検証可能とする。本研究では中立的仮想計算機モニタを BitVisor を用いて実現する。メディアプレイヤは H.264 動画が再生できる FFmpeg を BitVisor の拡張機能として実行する。

1. はじめに

現在、PC への動画配信が普及して久しいが、依然として著作権保護の問題が残されている。代表的な著作権保護技術には Microsoft 社のデジタル著作権管理技術 (Digital Right Management)[6] や TCP/IP で繋がった機器間でのデータ交換の際に用いる通信プロトコル DTCP(Digital TransmissionContent Protection)[11]-IP を用いた DiXiM がある。これらはソフトウェアのみで実現されているためリバースエンジニアリングによって破ることが可能である。Sony 社の PlayStation3[13] では独自の仮想計算機モニタ (Virtual Machine Monitor, VMM) を用いることで著作権保護を実現している。PlayStation3 では、ユーザがインストールした OS 上のプログラムで DVD のデータをコピーしようとすると VMM によりアクセスが拒絶される。しかし、独自の VMM であるためユーザに秘密で個人情報を送信している可能性もあり、ユーザが安心して利用できない問題がある。

本研究の目的は、次の 2 つを両立した著作権保護を実現することである。

- 動画像のストリーミング再生中にユーザ、または、悪意の第三者に動画データをコピーされないことを保証する。
- ユーザが安心して利用できる著作権保護を行う。

本研究ではこれを中立的な仮想計算機モニタを用いることで実現する。中立とはハードウェアの CPU と同様に期待された動作のみを行い、特に利害関係者（ユーザと権利者）のうち、どちらか一方に加担した行動をとらないことである。本研究では中立的仮想計算機モニタ上のユーザ用の OS とは隔離された空間でメディアプレイヤを動かし、動画像の再生中はメディアプレイヤに画面デバイスを専有させる。メディアプレイヤと配信サーバ間では暗号通信を行う。中立的仮想計算機モニタの非改竄はリモートアテステーション [7] を用いユーザとコンテンツプロバイダ双方で検証可能とする。本研究では中立的仮想計算機モニタを BitVisor を用いて実現する。メディアプレイヤは H.264 動画が再生できる FFmpeg を BitVisor の拡張機能として実行する。

本稿は以下のような構成になっている。2 章では本研究で実現する中立的仮想計算機モニタの要件について述べる。3 章では本研究で提案する手法を述べ、4 章ではその実装について述べる。5 章では提案した手法の評価を行う。そして、6 章では関連研究について述べ、最後に 7 章で本稿をまとめる。

2. 中立的仮想計算機モニタ

中立的仮想計算機モニタとは CPU 同様に規定の動作以外を行わない VMM である。本研究で用いる中立的仮想計算機モニタは以下の要件を満たす必要がある。

(1) ユーザ環境とメディアプレイヤを動作させる環境を提供する。これらの環境のメモリを隔離する。
(2) メディアプレイヤを動作させる環境では画面デバイス

[1] 筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻
Department of Computer Science of Systems and information Engineering, University of Tsukuba
[2] 株式会社イーゲル
Igel Co., Ltd

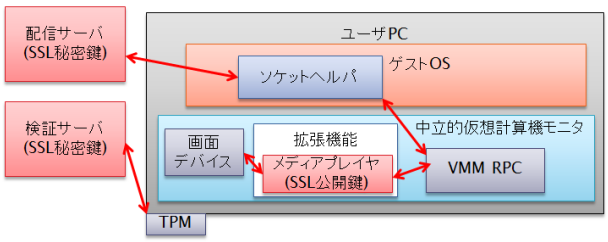

図 1 中立的 VMM を用いた著作権保護の仕組み

のみ扱える。ユーザ環境ではデバイス全般を扱える。

(3) 両環境のプロセスによる画面デバイスの排他的アクセスができる。

(4) 環境間の通信機能を提供する。

(5) ユーザと権利者双方から VMM の完全性の検証が可能である。

(6) VMM の中立性はソースコードを公開することで担保する。

VMM の完全性の検証は TPM を用いたリモートアテステーションにより行う。TPM (Trusted Platform Module) とは耐タンパなセキュリティチップである [7]。TPM は Trusted Boot の際に用いられる。Trusted Boot とは CRTM(Core Root of Trust Measurement) を起点として次に読み込むモジュールのハッシュ値を計算し TPM の PCR(Platform Configuration Register) へ格納することで信頼の連鎖を作りながら起動する手法である。VMM の完全性（改ざんされていないこと）はリモートアテステーションにより起動後に検証できる。

3. BitVisor による著作権保護手法の設計

本研究では BitVisor により中立的仮想計算機モニタを実現し、動画像への著作権保護を実現する（図 1）。

BitVisor とは筑波大学が中心となって開発したハイパーバイザ型の VMM である [1]。BitVisor ではゲスト OS に依存しないデバイス単位でのアクセス制御やディスク、ネットワークパケットの暗号化を行うことが出来る。BitVisor 上では一つのゲスト OS しか動作させることができないが、全体で 21,582 行 [12]、動作時は 13,789 行と VMM の中では Trusted Computing Base(TCB) が小さい。BitVisor とゲスト OS は互いに異なるメモリ空間で動作する。また、BitVisor は BitVisor 本体とは異なるメモリ空間で権限が制約された拡張機能を実行できる。BitVisor では BitVisor の拡張機能として BitVisor の管理者が許可したプログラムを動的にロードすることができる [5]。本研究では BitVisor の拡張機能を用いてユーザ用、コンテンツプロバイダ（Content Provider、以後 CP と呼ぶ）用の 2 つの実行環境を作成する。さらに、BitVisor は TPM に対応しているため、リモートアテステーションによる BitVisor の完全性の検証が比較的容易に行える [5]。以下の節で詳しく述べる。

3.1 脅威モデル

本研究では、次のグループの人々を考える（表 1）。本研究では、ユーザと CP に対して検証者と VMM 管理者は中立とする。本研究では、VMM 管理者は必ずしも中立であることを要求しない。ユーザ自身、または、CP が VMM 管理者となることも有りえる。

本研究では、以下の 2 点を保証する著作権保護システムを実現する。

- ユーザが安心できる。メディアプレイヤやソケットヘルパから個人の情報を守ることができる。
- CP の権利が守られる。ユーザ、または、ネット上の第三者に復号された動画像データをアクセスさせない。

ここでは、まず本研究で想定する TCB について述べる。そして双方にとって最悪のシナリオである VMM の管理者が PC ユーザの味方をする場合と CP の味方をする場合の脅威モデルについて述べる。

3.1.1 Trusted Computing Base

本研究ではユーザ環境においてメモリを凍結して情報を盗む攻撃などのハードウェアに対する攻撃は想定しない。TPM の秘密鍵はハードウェア所有者ですら取り出せず、適切に管理されているものとする。TPM の公開鍵は検証者が確認できるプライバシ認証局が適切に管理しているものとする。PC にインストールされている VMM の完全性は TPM を用いたリモートアテステーションにより検証者が検証でき、ユーザと配信サーバ双方で検証者の結果を信用する。リモートアテステーションにより、VMM が再起動された際は配信サーバ側から検知可能とする。BitVisor、メディアプレイヤ、ソケットヘルパには脆弱性はないものとする。

3.1.2 ソースコードとバイナリの同一性検証

本研究では、次の手順でソースコードとバイナリの同一性を確認する（これを同一性検証と定義する）。

(1) バイナリの供給者は、ソースコードを公開し、ソースコードのレベルでそのソフトウェアが規定の動作以外を行わないことを確認する。

(2) バイナリの供給者は、コンパイル環境（コンパイラのバージョン、OS など）を公開し、誰でも再現可能とする。

(3) バイナリの供給者は、(1) を (2) でコンパイルしたバイナリを公開する。

(4) 同一性検証者は、(1)、および、(2) の手順を自ら繰り返し、(3) と同じバイナリが再現できることを確認する。

バイナリを利用する人は、自分自身が同一性検証者となり、手順 (4) を行ってもよい。あるいは、バイナリを利用する人は、他の信頼できる人が行った検証を信じることも出来る。

3.1.3 VMM 管理者がユーザである場合の脅威モデル

ユーザやネット上の第三者からメディアプレイヤと配信

表 1 脅威モデルにおけるグループ一覧

グループ	説明	中立性
ユーザ	PC の利用者。ユーザ環境でプログラムを実行できる。	ユーザ
CP	配信サーバを管理する。メディアプレイヤとソケットヘルパのソースコード、およびバイナリを提供する。	CP
VMM 開発者	VMM のソースコードを公開する。	中立
VMM 管理者	VMM 管理者は署名のみ行える。ユーザは署名されたプログラムのみ VMM へロードすることができる。	任意
検証者	TPM の AIK、および、リモートアテステーションのためのサーバを管理する。	中立
ネット上の第三者	ネットワーク上に存在し、常に通信を傍受している。	任意

サーバ間の通信を傍受し著作物を入手する試みは、メディアプレイヤへ配信サーバの SSL（Secure Socket Layer）公開鍵を埋め込み SSL を用いて通信することで防ぐ。ゲスト OS や他の拡張機能からの動画再生中に画面デバイスのメモリデータをコピーする試みは、画面デバイスをメディアプレイヤで専有させることで防ぐ。他の BitVisor の拡張機能からメディアプレイヤのメモリにアクセスして復号された著作物を入手する試みは、BitVisor の拡張機能ごとにメモリ空間を隔離して防ぐ。

ユーザ、すなわち VMM 管理者はメディアプレイヤへ VMM 管理者として署名するためには、CP によるメディアプレイヤのソースコードの公開が必須である。それを逆手に取って、ユーザが著作物を入手できるよう改造したメディアプレイヤをロードする攻撃が考えられる。それは拡張機能が読み込まれると TPM にその拡張機能のハッシュを追加する機能を付加することで、動画データを配信する前にメディアプレイヤの完全性を配信サーバ側から検証することで防ぐ。

メディアプレイヤからのゲスト OS で直前まで利用していた画面デバイス内のメモリから画面情報を盗む攻撃が考えられる。この攻撃は、BitVisor により画面デバイスを専有しているプロセス（ゲスト OS、メディアプレイヤ、他の拡張機能）が切り替わる際にゼロクリアすることで防ぐ。もしくは、VMM の管理者がユーザ自身であるため、メディアプレイヤのソースコードを見て悪意ある動作をすると判断した場合は、拡張機能として署名しないことで防ぐ。

3.1.4 VMM 管理者が CP である場合の脅威モデル

CP が VMM の管理者である場合、自身が作成したメディアプレイヤが如何に悪意ある動作をするとしても署名することができる。そのため、ソースコードが公開されていない拡張機能は単なるバイナリの実行ファイルと同様の意味を持つ。ユーザがバイナリの供給者を信用できると判断すればそのバイナリを VMM にロードすればよい。この時、拡張機能は、画面デバイスに動画像を表示する以上のことはできないことが VMM により保証されている。したがって、メディアプレイヤがユーザの個人の情報を盗むことはできない。ユーザがバイナリの供給者を信用できない場合、ユーザはメディアプレイヤのソースコードを要求する。そして、3.1.2 項で述べた手順でソースコードとバイナリの同一性を検証した上で、そのバイナリを VMM にロードする。また、ゲスト OS 上で動作するソケット・ヘルパも jail 等の既存の仕組みで個人の情報を保護することができる。

3.2 画面デバイスへのアクセス制御

動画像の著作権を保護するためには、動画再生中はゲスト OS からの画面デバイスへのアクセスをブロックする必要がある。BitVisor には、メモリマッピング変更機能や準パススルー機能、デバイスを隠す機能がある。そこで、本研究では BitVisor を用いて画面デバイスのゲスト OS からの隠ぺいを実現する。この論文では画面デバイスの最も簡単なものとしてメインメモリ上のフレームバッファを対象として、画面デバイスの保護について述べる。

BitVisor では複数の拡張機能をロードすることが可能である。すなわち、メディアプレイヤ以外からも画面デバイスを利用することができる。そこで、先に画面デバイス利用の要求のあった拡張機能が利用を続ける限り、他の拡張機能からのアクセスを拒絶する。これにより、メディアプレイヤ以外の拡張機能からの画面デバイスへのアクセスによる画像データのコピーを防ぐことができる。

3.3 ソケットヘルパ

BitVisor の拡張機能では OS 的なプログラムがないことによりソケットによる通信ができない。そこで、ゲスト OS 上にソケットによる通信を行うプログラムを作成する。以後このプログラムをソケットヘルパと呼ぶ。ソケットヘルパでは BitVisor の拡張機能として動作しているメディアプレイヤがソケット機能を要求すると代わりにゲスト OS の機能を使ってソケットによる通信を行う。ソケットヘルパは、メディアプレイヤのユーザインタフェースとしての機能も持つ。

ソケットヘルパは RPC(Remote Procedure Call)[2] を発行し、ソケットの要求を取得する。結果も RPC を用いて BitVisor のメディアプレイヤへ送ることができる。メディアプレイヤにおいて SSL の暗号通信を行うことでソケットヘルパはルータと同様に配信サーバの通信内容を傍受することはできない。配信サーバ側の SSL の公開鍵はあらかじめメディアプレイヤへ埋め込むことで中間者

```
static int bmp_encode_frame(AVCodecContext *avctx,
     unsigned char *buf, int buf_size, void *data)
    {
  /* 省略 */
  fbp_p = fbp + (vinfo.xres *
  vinfo.bits_per_pixel / 8) *
  (avctx->height - 1);
  for(i = 0; i < avctx->height; i++) {
    if (bit_count == 16) {
      const uint16_t *src = (const uint16_t *) ptr
          ;
      uint16_t *dst = (uint16_t *) buf;
      for(n = 0; n < avctx->width; n++)
      AV_WL16(dst + n, src[n]);
    } else {
      memcpy(buf, ptr, n_bytes_per_row);
      // for frame buffer
      fb_memcpy3(fbp_p, ptr, n_bytes_per_row);
      fbp_p -= vinfo.xres * vinfo.bits_per_pixel /
           8;
    }
    buf += n_bytes_per_row;
    memset(buf, 0, pad_bytes_per_row);
    buf += pad_bytes_per_row;
    ptr -= p->linesize[0]; // ... and go back
  }
  return n_bytes;
}
```

図 2 BMP 出力のフレームバッファ出力への変更

(man-in-the-middle) 攻撃を防ぐ。

4. BitVisor による著作権保護手法の実装

4.1 ユーザ PC の構成

ユーザ PC には ThinkPad の X61 を用いる。X61 は拡張ユニットを用いるとシリアル通信が可能なため BitVisor のデバッグが容易に行えるためである。また BitVisor は ver. 1.2 を用いる。ゲスト OS には Fedora 13 を用いる。

4.2 動画プレイヤの移植

本研究では、BitVisor の拡張機能上で H.264(MPEG-4) の動画を対象とする。本研究では Linux 用の H.264 などが再生できる最小構成の動画プレイヤである FFmpeg[3] を BitVisor の拡張機能として実行する。

4.2.1 H.246 動画のフレームバッファへの出力

本稿では画面デバイスとして最も単純な構造であるフレームバッファを用いる。フレームバッファとはディスプレイに表示される画像データのバッファである。ThinkPad X61 ではメインメモリ上のアドレスにマップされており、そのメモリの内容が一定周期でディスプレイに走査されることで画面上に画像が表示される。

FFmpeg は標準でフレームバッファへの出力に対応していない。しかし、Microsoft BMP (ピクセルマップ画像) への変換には対応している。BMP と ThinkPad X61 のフレームバッファのデータ形式は異なるが容易に変換が可能である。そこで、BMP への出力を行うプログラムを ThinkPad X61 のフレームバッファへの出力に変更する。コードの主要部分を図 2 に示す。もともとの bmp_encode_frame 関数では BMP ヘッダを生成した後に BGR(Blue Green Red の順) 形式で画像の下から上の順に画像データを生成して

```
% time seconds usecs/call calls errors syscall
------------------------------------------------
46.43 0.002123 3  646    close
35.02 0.001601 0  5214   write
15.99 0.000731 1  646    open
0.74  0.000034 5  7      munmap
0.72  0.000033 0  648    lseek
0.68  0.000031 0  1582   select
0.28  0.000013 0  165    read
0.13  0.000006 0  1575   gettimeofday
0.00  0.000000 0  6      fstat
0.00  0.000000 0  7      mmap
0.00  0.000000 0  14     brk
0.00  0.000000 0  6      rt_sigaction
0.00  0.000000 0  1      rt_sigprocmask
0.00  0.000000 0  10 2   ioctl
0.00  0.000000 0  1      execve
0.00  0.000000 0  1      uname
0.00  0.000000 0  1      getrlimit
0.00  0.000000 0  2      getrusage
0.00  0.000000 0  1      arch_prctl
0.00  0.000000 0  1      set_tid_address
0.00  0.000000 0  1      set_robust_list
------------------------------------------------
100.00 0.004572   10535 2   total
```

図 3 サンプル動画をフレームバッファ形式へ変換する際の FFmpeg の strace の結果

```
static void swap(void *a, void *b, size_t size){
  unsigned char buf[8];
  memcpy(buf, a, size);
  memcpy(a, b, size);
  memcpy(b, buf, size);
}
static void qsort2(void *base, size_t nmemb,
    size_t size, int(*compar)(const void *, const
    void *)){
  int i, j;
  for (i = 0; i < nmemb; i++) {
    for (j = i + 1; j < nmemb; j++) {
      if ((*compar)((unsigned char *)base + size *
           i, (unsigned char *)base + size * j) >
          0) {
        swap((unsigned char *)base +
        size * i, (unsigned char *)base + size * j
            , size);
      }
    }
  }
}
```

図 4 qsort(3) を置き換えた qsort2 関数

いた。そこで、新しく作成した fb_memcpy3 関数を用い、同様の箇所でフレームバッファのデータ形式である BGR α (Blue Green Red αチャンネルの順) 形式へ変換している。また、フレームバッファは一般的な BMP と違い、画像データの格納順は上から下であるため、順序を逆にしてフレームバッファに出力するように変更する。

4.2.2 システムコールの削除とライブラリ関数の置き換え

BitVisor の拡張機能で利用可能なシステムコールは Linux 上で利用可能なシステムコールとは異なっている。そのため、FFmpeg から一切の Linux のシステムコールを排除する必要がある。実際に H.264 動画をフレームバッファ形式へ変換する際に FFmpeg で使用されたシステムコールの一覧を図 3 に示す。main 関数から main 関数を抜けるまでに呼ばれているシステムコールについては不要なら削除、必要ならば処理をシステムコールを使わない別の関数で置き換える (同様の動作をする関数を新たに定義しそちらを呼び出すよう変更する)。

```
u64 gmm_pass_gp2hp (u64 gp, bool *fakerom) {
  bool f;
  u64 r;
  if (phys_in_vmm (gp)) {
    r = phys_blank;
    f = true;
  } else if (phys_in_fb (gp)) {
    r = 0xE07F0000;
    f = false;
  } else {
    r = gp;
    f = false;
  }
  if (fakerom)
    *fakerom = f;
  return r;
}
```

図 5 ゲスト OS からのフレームバッファの隠ぺい

置き換えの例として、図 4 に qsort(3) の置き換えである qsort2 関数を示す。qsort(3) は内部で/proc/meminfo へのファイルアクセスを行っておりシステムコールである open(2)、close(2)、read(2)、mmap(2) などを呼び出していたため BitVisor の拡張機能では正常に動作しない。置き換えた qsort2 関数は選択ソートを行っているが FFmpeg では 50 程度の要素数のソートにしか利用していないため計算量の問題は無視できる。swap 関数の buf サイズについても FFmpeg での必要最低限のサイズとし、動的確保は行っていない。

システムコールを排除し静的リンクされた FFmpeg は、BitVisor の拡張機能として動作すると期待される。しかし、実際には一部の libc.a ライブラリコールでセグメンテーションフォールトが発生する。これは libc.a が Linux を前提として書かれているためである。例えば、main 関数が呼ばれる以前に確保されたメモリなどへのアクセスがこれに当たる。そのため、BitVisor の拡張機能で動かないものについては置き換える。算術ライブラリである libm.a については FFmpeg の H.264 動画のデコードに必要なものに限り問題なく動作することを確認済みである。

4.3 BitVisor の修正

本研究では著作権保護の目的から H.264 などの動画のデコードを BitVisor の拡張機能として実行する必要がある。また SSL を利用したネットワーク通信も必要となってくる。しかし既存の BitVisor では浮動小数点演算やソケットを用いたネットワーク通信ができない等の制限がある。そこで、ここでは BitVisor の修正について述べる。

4.3.1 フレームバッファへのアクセス制御

既存の BitVisor には VMM の特性上ゲスト OS から自身のメモリ空間へのアクセスを保護する機能が備わっている。そこで、フレームバッファへのアクセス制御はその機能を利用することで実現する。コードの主要部分を図 5 に示す。gmm_pass_gp2hp 関数はもともと BitVisor に備わっている関数であり、ゲスト OS が保持する物理アドレスを VMM が保持する物理アドレスへ変更する関数である。そ

```
static int process_new (int frompid, void *bin) {
  /* 省略 */
  for (i = 0; i < fb_size / fb_p_size; i++){
    mm_process_map_shared_physpage(
    fb_virt+(i*fb_p_size),fb_phys+
    (i*fb_p_size), true);
  }
  /* 省略 */
}
```

図 6 拡張機能のメモリ空間へのフレームバッファのマップ

```
static void vmmcall_rpc_stub (void) {
  /* 省略 */
  /* SET CR0_TS_BIT = 0 */;
  asm_rdcr0(&cr0);
  cr0 &= ~CR0_TS_BIT;
  asm_wrcr0(cr0);
  save_fpu_registers(env);
  ret = vmmcall_rpc(arg.desc, arg.request, arg.
      result);
  /* SET CR0_TS_BIT = 1 */;
  asm_rdcr0(&cr0);
  cr0 |= CR0_TS_BIT;
  asm_wrcr0(cr0);
  restore_fpu_registers(env);
  /* 省略 */
}
```

図 7 拡張機能 (VMM RPC) での浮動小数点演算の実現

の関数の内部でゲスト OS が持つフレームバッファの物理アドレスを判定することで、ゲスト OS からはフレームバッファへアクセスしているつもりでも実際には VMM 上のダミーのメモリへのアクセスとなる。そして、シャドウページテーブルから該当エントリを削除することでゲスト OS からフレームバッファへアクセスすることを禁止する。

4.3.2 拡張機能からのフレームバッファへのアクセス

BitVisor の拡張機能は一般的なユーザプロセスの扱いであり、VMM とはメモリ空間を共有していない。そのため、BitVisor の拡張機能からはフレームバッファへアクセスすることができない。そこで、BitVisor の拡張機能のメモリ空間にフレームバッファの物理アドレスをマップ（対応付ける）することでフレームバッファへのアクセスを可能にする。コードの主要部分を図 6 に示す。process_new 関数は BitVisor の拡張機能を生成する際に呼ばれる関数である。その内部で、任意の物理アドレスを BitVisor の拡張機能のメモリ空間へマップする mm_process_map_shared_physpages 関数を利用し、フレームバッファを BitVisor の拡張機能のメモリ空間にマップするように変更する。その他、変更していない BitVisor では複数の拡張機能が実行可能である。本研究の手法で動画像に対する著作権保護を実現するためにはフレームバッファをマップする拡張機能を常に 1 つにするか、BitVisor へロードできる拡張機能の上限を 1 つにするように変更する。

4.3.3 浮動小数点演算の実現

既存の BitVisor は浮動小数点演算の処理を使わずに実装されており、浮動小数点演算をした際は例外が発生するようになっている。そのため BitVisor の拡張機能であっても浮動小数点演算を行うことができない。しかし、FFmpeg

```
struct env {
  unsigned long long int rax, rbx, rcx, rdx;
  unsigned long long int rsi, rdi, rsp, rbp;
  unsigned long long int r8, r9, r10, r11;
  unsigned long long int r12, r13, r14, r15;
};
typedef struct env env_t[1];
extern int env_save(env_t *e);
extern void env_restore(env_t *e);
```

図 8 コールスタックの保存に用いる関数

では H.264 動画のデコードの際に浮動小数点演算を使用している。そこで、BitVisor の拡張機能に限り浮動小数点演算が実行できるようにする必要がある。浮動小数点演算が行われた際の例外は CR0 レジスタの TS(Task Switch) ビットが立っていると発生する。そこで、BitVisor の拡張機能が VMM RPC(Remote Procedure Call) 関数によって呼ばれたときにのみ浮動小数点演算が可能になるように変更する。それを実現するコードの主要部分を図 7 に示す。vmmcall_rpc_stub 関数は VMM RPC 関数によって呼ばれる関数である。vmmcall_rpc 関数が実際に BitVisor の拡張機能に処理を渡す関数である。ここでは vmmcall_rpc 関数が呼ばれる前に CR0 レジスタの TS ビットを落とし、処理が戻ってくるときに TS ビットを立てることで BitVisor の拡張機能で浮動小数点演算命令が実行できるようにしている。また、その前後で FPU(Floating Point number processing Unit) 関連のレジスタの保存と復元も行う。

4.4 拡張機能とゲスト OS の通信

ここでは BitVisor の拡張機能とゲスト OS の通信に必要な機能について述べる。

4.4.1 拡張機能でのスリープの実現

既存の BitVisor の拡張機能は横取り禁止（non-preemptive）で動作するため、処理が終わらない限りゲスト OS に処理が戻らない。そのため、BitVisor の拡張機能は、ネットワーク通信においてデータの読み込みなどでスリープをするプログラムを書くことができない。そこで、ゲスト OS に処理を戻したいタイミングでコールスタックを保存し、ゲスト OS に処理を戻せるようにする。具体的にはスタックポインタ並びにレジスタ [4] の保存と復元を行う。そのために用いる env_save 関数と env_restore 関数のインターフェースを図 8 に示す。これらの関数本体はアセンブリ言語で書かれており、env_save 関数が呼ばれるとレジスタを保存した後 jmp 命令で_start 関数まで飛び、その後ゲスト OS に処理が戻る。この関数を用いる際は、_start 関数に現在の状態を確認し、必要ならば env_restore 関数を呼ぶ処理を追加する。

4.4.2 VMM RPC 関数によるゲスト OS と拡張機能の通信

既存の BitVisor はソケット API によるネットワーク通信機能がないため BitVisor の拡張機能からも同様にネットワーク通信が簡単には行えない。そのため、ゲスト OS のソケットヘルパに通信を委託する。BitVisor の拡張機能として実行する FFmpeg とソケットヘルパの間の通信プロトコルを図 9 に示す。struct rpc_ffmpeg_result 型は RPC の結果を保存する構造体であるが、このプロトコルでは FFmpeg からゲスト OS のプロセス (ソケットヘルパ) への要求である。ソケットヘルパは req_type と syscall_num の値から定まるシステムコールを与えられた引数に従いゲスト OS 上で実行する。struct rpc_ffmpeg_arg 型は RPC の要求を保存する構造体であるが、このプロトコルではソケットヘルパがサーバからの応答を FFmpeg に返す際に用いる。

```
/* response to ffmpeg */
struct rpc_ffmpeg_arg {
  u8 res_type;// 0: nothing, 1: socket
  u8 syscall_num;// 1: read, 2: write, 3: connect
  u32 xid;
  u8 buf[1024*768*4];
  u32 buf_len;
  u32 ret;
};
/* request from ffmpeg */
struct rpc_ffmpeg_result {
  u8 req_type;// 0: nothing, 1: socket
  u8 syscall_num;// 1: read, 2: write, 3: connect
  u32 xid;
  u32 sys_args[10];
  u8 buf[4096*10];
  u32 buf_len;
  char inet_addr[32];
  u16 inet_port;
  u32 ret;
};
```

図 9 拡張機能として実行する FFmpeg との通信プロトコル

FFmpeg は SSL を用いることで安全に配信サーバとの通信が行える。現在までに、ソケットヘルパを介し FFmpeg からの HTTP によるサーバとの通信に成功している。

5. 評価

ここでは、本研究で提案した著作権保護システムの評価を行う。また、本稿の実装についての評価を行う。

Sony 社の PlayStation3 では、VMM というゲスト OS よりも高い権限で、ユーザが信頼できないプログラムが動作していた。これに対して本研究の方法では、拡張機能という制限された空間、および、ソケット・ヘルパという、ゲスト OS の上で動作する通常のプログラムを利用している。前者は、信頼できる中立の VMM により行動が監視されているので利用者は安心できる。後者は、通常のプロセスであり、ユーザは、jail 等の既知の方法で行動を監視できる。また、著作権者側も、拡張機能や画面デバイスに復号した著作物を置いたとしても、ユーザからアクセスされないことは、信頼できる中立の VMM により保証されている。

提案方式では、利用者は検証のためにメディアプレイヤやソケットヘルパのソースコードの開示を求める。それらのソースコードが公開できない場合、利用者は自分自身で検証することができない。もし、利用者と著作権者の両方

が信頼できるバイナリの検証機関があれば、それを利用する方法も考えられる。

3.1 節で述べたソースコードとバイナリの同一性確認手続きで、同じバイナリが作られることを再現するには一定の手間がかかる。その手間は、通常のパッケージ・ソフトウェアのインストールと同様に、信用がある会社のものならば、信頼するという方法も採用することができる。ただしこの場合でも、VMM のように高い権限を持つプログラムを信頼する必要はなく、拡張機能という制限された空間、および、ソケット・ヘルパという、ゲスト OS の上で動作する通常のプログラムを信頼すれば十分である。

本稿では画面デバイスとして最も単純なメインメモリ上のフレームバッファを用いた。現在普及している PC では通常、画像処理にグラフィックアクセラレータを用いている。グラフィックアクセラレータは種類が多く、BitVisor で専有するプロセスを切り替え可能とすることはフレームバッファより困難であると予想される。グラフィックアクセラレータへの対応は今後の課題である。

本稿では BitVisor の拡張機能として実行するメディアプレイヤを FFmpeg で実現した。現在は FFmpeg の内部的には H.264 を BMP へ変換する機能のみ拡張機能として実行できる。しかしながら、BitVisor への Linux 上の画面デバイスを扱うアプリケーションの移植は初の試みであり、本研究は BitVisor の発展の可能性を示した。FFmpeg の他の機能を BitVisor の拡張機能として実行するためには、本稿で示した方法を適用することで可能と予想される。今回、Linux 上で動作する pthread の部分に関しては、BitVisor の拡張機能として実現することが困難であるため FFmpeg のコンパイルオプションで無効にした。BitVisor のマルチスレッドへの対応は今後の課題である。

6. 関連研究

ADvisor[10] とは BitVisor を改造して実装された広告表示システムである。CR3 レジスタの更新情報を用いてグラフィックアクセラレータ内にあるフレームバッファへ画像データを書き込むことで広告を表示する。そのため、ゲスト OS ではグラフィックアクセラレータを用いた描画処理を行う限りにおいて、グラフィック処理が ADvisor と競合し広告が上書きされる。そのため、ADvisor がインストールされた PC へは、ゲスト OS に関係なく常に画面上への広告表示が可能である。しかし、ゲスト OS から広告画像へのアクセスが可能なため著作権保護できない。本研究ではメインメモリ上のフレームバッファを BitVisor の拡張機能で専有し、ゲスト OS からのアクセスを不可能にする。

その他、VMM で VM 間のメモリやディスク内容を保護する研究 [14][15]、リモートと操作者の通信チャネルを保護する研究 [2][8][15]、VM で実行されるアプリケーションのデータを保護する研究 [9] がある。しかし、画面デバイスが専有できない。本研究では BitVisor の拡張機能で画面デバイスを専有する。

7. おわりに

この論文では、中立的仮想計算機モニタによる動画像を対象とした著作権保護について述べた。現在までに BitVisor の拡張機能として FFmpeg による H.264 動画のデコード並びに保護されたフレームバッファへの画像の出力が可能となった。また、ソケットヘルパを用いた配信サーバと FFmpeg の HTTP による通信により、動画のストリーミング再生も可能となった。BitVisor の拡張機能として Linux 上の画面でバイスを扱うアプリケーションの実行は初の試みであり、本研究は BitVisor の発展の可能性を示した。今後は SSL による配信サーバと FFmpeg 間の通信内容の暗号化、完全性が検証できた BitVisor 環境への配信サーバからの動画配信を行う予定である。グラフックアクセラレータへの対応と、音声データの保護は今後の課題である。

謝辞

本研究で利用した BitVisor の拡張機能は、筑波大学システム情報工学研究科コンピュータサイエンス専攻（現在、東京大学情報基盤センター）の品川高廣氏の設計による。この拡張機能がなければ、BitVisor へメディアプレイヤを移植することは困難であった。ここに深く感謝の意を表する。

本研究の一部は、日本学術振興会科学研究費補助金（挑戦的萌芽研究）、および、総務省戦略的情報通信研究開発制度（SCOPE）の支援を受けて行われた。

参考文献

[1] BitVisor Project: *BitVisor1.2*, http://www.bitvisor.org/ (2012).
[2] 江川友寿，光来健一：管理 VM へのキーボード入力情報漏洩の防止，情報処理学会研究報告. [システムソフトウェアとオペレーティング・システム]， Vol. 2011, No. 8, pp. 1–8 (2011).
[3] FFmpeg Project: *FFmpeg SVN-r26402*, http://ffmpeg.org/ (2012).
[4] Intel Corporation: *Intel 64 and IA-32 Architectures Software Developer's Manual Combined Volumes:1,2A,2B,2C,3A,3B and 3C* (2012).
[5] 松下正吾，新城　靖，榮樂英樹，松原克弥，東　悠：中立的仮想計算機モニタによる耐タンパーデバイスのアクセラレータの実装，情報処理学会研究報告. [システムソフトウェアとオペレーティング・システム]， Vol. 2011, No. 9, pp. 1–8 (2011).
[6] Microsoft Corporation: *Windows Media Technical Articles Archive* (2012).
[7] 宗藤誠治，須崎有康：5 高信頼を実現する Linux の新しい機能 (< 特集 >Linux のセキュリティ機能)，情報処理，Vol. 51, No. 10, pp. 1284–1293 (2010).
[8] 西村直樹，江川友寿，光来健一：IaaS における管理 VM への画面情報漏洩の防止，情報処理学会研究報告. [システムソフトウェアとオペレーティング・システム]， Vol. 2012, No. 5, pp. 1–8 (2012).

[9] 尾上浩一，大山恵弘，米澤明憲：アプリケーションデータを保護するための VMM に基づくアーキテクチャ，情報処理学会論文誌. コンピューティングシステム， Vol. 2, No. 3, pp. 173–188 (2009).
[10] 小川夏樹，大山恵弘：ADvisor:ゲスト OS の操作に連動した広告を表示するハイパバイザ，情報処理学会研究報告. [システムソフトウェアとオペレーティング・システム]， Vol. 2011, No. 5, pp. 1–7 (2011).
[11] Ripley, M., Brendan, C., Traw, S., Group, C. T. and Corporation, I.: Content Protection in the Digital Home, *Intel Technology Journal*, Vol. 6, No. 4, pp. 49–56 (2002).
[12] Shinagawa, T., Eiraku, H., Tanimoto, K., Omote, K., Hasegawa, S., Horie, T., Hirano, M., Kourai, K., Oyama, Y., Kawai, E., Kono, K., Chiba, S., Shinjo, Y. and Kato, K.: BitVisor: a thin hypervisor for enforcing i/o device security, *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '09, ACM, pp. 121–130 (2009).
[13] Sony Computer Entertainment Inc: *Documents of PS3 Linux Distributor's Starter Kit* (2008).
[14] 田所秀和，光来健一，千葉　滋：IaaS 環境における VM のメモリ暗号化による情報漏洩の防止，情報処理学会研究報告. [システムソフトウェアとオペレーティング・システム]， Vol. 2011, No. 15, pp. 1–8 (2011).
[15] Zhang, F., Chen, J., Chen, H. and Zang, B.: CloudVisor: retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization, *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, ACM, pp. 203–216 (2011).