DEOS-FY2013-QK-01J

JST-CREST

研究領域

「実用化を目指した組込みシステム用ディペンダブル・オペレーティングシステム」

DEOS プロジェクト

QEMU-KVM + D-System Monitor
環境構築手順書

Version 1.0

2013/05/01

DEOS 研究開発センター

科学技術振興機構
Japan Science and Technology Agency

目次

1. D-Visor と D-System Monitor 3
2. 動作環境 3
3. 環境構築の流れ 3
4. Guest OS の Uni Processor Kernel 化 4
5. QEMU-KVM の修正 5
5-1. 必要なパッケージのインストール 5
5-2. パッチの適用 5
5-3. アドレスの変更 5
5-4. QEMU-KVM のビルド 5
5-5. シンボリックリンクの作成 5
6. Host OS の KVM カーネル・モジュールの修正 6
7. QEMU-KVM の実行 6
8. Guest OS へのパッチの適用 7
9. デモ用のカーネル・モジュール等の作成 7
9-1. rpldev.ko と rpld の作成 7
9-2. rpld_receiver の作成 8
9-3. file_rootkit.ko の作成 8
9-4. process_rootkit.ko の作成 8
9-5. 監視用ユーザープログラムの準備 8
10. GUI の準備 9
11. デモの実行 9
11-1. GUI の起動 9
11-2. FoxyKBD の実行 9
11-3. RootkitLibra の実行 10
11-4. Waseda LMS の実行 10
12. おわりに 10

1. D-Visor と D-System Monitor

D-System Monitor は OS を外側から観察し、OS に対する攻撃・改竄を監視します。OS を外側から監視するためには、OS を稼働させるハードウェアに特別な観察のための仕組みを用意する方法も考えられますが、DEOS プロジェクトでは、OS を仮想マシン(VM)上で動かし、VM を管理する仮想マシンモニタ(VMM)に観察を支援する機能を組み込み、それを使って OS を外側から監視します。この仮想マシンモニタを D-Visor と呼びます。

DEOS プロジェクトでは、D-Visor として x86 マルチコア CPU 上で動く D-Visor86 を開発し（「D-Visor86 + D-System Monitor 環境構築手順書」を参照）、D-Visor86 上で動作する D-System Monitor も開発しました。しかし、D-Visor86 は稼働するハードウェアに制限があります。そこで、D-System Monitor の適用領域を拡大することを目的として、様々な環境で使用されている QEMU-KVM を修正し D-Visor として利用しようとを試みたのが、本書で説明する D-System Monitor を組み込んだ QEMU-KVM（以下「QEMU-KVM+D-System Monitor」）です。

OS に対する攻撃・改竄としては様々なものが考えられ、それぞれに応じた監視機構を D-System Monitor 内に実現することが必要となりますが、QEMU-KVM+D-System Monitor で現在使用できる監視機構は以下の 3 つです。

1. FoxyKBD
   疑似的に大量のキーボード入力を発生させ、同時にネットワークの転送量を監視します。無関係なはずのキーボード入力とネットワーク転送に相関がみられる場合は、異常な振舞いの可能性があります。
2. RootkitLibra
   NFS マウントしているディレクトリについて Guest OS 上で見える結果と NFS パケット内のデータを比較して、Guest OS が不正なファイルメタデータを返すことを検出します。
3. Waseda LMS (Lightweight Monitoring Service)
   Guest OS の kernel 中の task list と run queue を比較して、プロセス情報に矛盾があることを検出します。

いずれも D-Visor86 上で稼働している D-System Monitor の監視機構と同じ機能であり、ソースコードの一部を修正して QEMU-KVM+D-System Monitor に組み込んでいます。

2. 動作環境

QEMU-KVM+D-System Monitor の稼働実績がある環境は以下のとおりです。

- ハードウェア
  Intel VT をサポートする x86_64 CPU が搭載された PC。AMD-V の CPU には未対応です。
- ソフトウェア
  Host OS: Ubuntu 12.04 x86_64 版
  Guest OS: Ubuntu 12.04 x86_64 (Uni Processor Kernel)

3. 環境構築の流れ

QEMU-KVM+D-System Monitor が動作する環境を構築するには、以下のような作業が必要になります。

1. Guest OS を Uni Processor Kernel に変更する
2. Host OS の QEMU-KVM を修正する
3. Host OS の KVM カーネル・モジュールを修正する

さらに、前述の 3 つの監視機構を動かして、異常を検出するデモを行なうには、以下のような作業が必要です。

4. Guest OS にパッチをあてて、デモのために異常なデータを返す状態にする
5. デモ用のカーネル・モジュール等を準備する
6. 監視機構が使用するユーザープログラムを準備する
7. GUI を準備する

次章以降で、これらの作業の詳細を説明します。Host OS として Ubuntu 12.04 x86_64 版がインストールされ、さらに、KVM パッケージがインストール済みであることを前提としています。また、プログラムのビルド等はすべて Host OS 上で実行しているものとしています。Host OS 上ではデモ用 GUI も動作させるので、X Window 環境も必要です。

公開している qemu-kvm-dsysmon-0.1.0.tar.gz には以下のようなファイルが含まれています。プログラムはいずれもソースコードかパッチです。

名前	内容
COPYING	GPL v2 ライセンス
COPYRIGHT	コピーライト
GUI/	GUI のファイル
README	Readme ファイル
file-rootkit/	異常なファイルメタデータを返すデモ用カーネル・モジュール
guest-os/	Guest OS 変更のための patch 等
host-os/	Host OS の KVM カーネル・モジュールの修正のための patch
process-rootkit/	異常なプロセス情報を返すデモ用カーネル・モジュール
qemu-kvm/	QEMU-KVM に D-System Monitor を組み込むための patch
rpld-receiver/	キーボード入力データを受信するプログラム
rpld.conf	rpld の設定ファイル
rtkl-collec.kvm/	RootkitLibra 監視機構のユーザープログラム。GuestOS 上で動作
start-qemu.sh	QEMU-KVM+D-System Monitor を起動するためのシェル・スクリプト
ttyrpld-2.60/	キーボード入力データを記録するためのオープンソースプログラム用の patch

これらを使用して以下の環境構築作業を実施する前に、Ubuntu 12.04 x86_64 版をインストールした Guest OS の KVM 用イメージを、従来の KVM コマンドを使って作成しておいてください。以下では、その Guest OS の KVM イメージに対して kernel を入れ替えたりカーネル・モジュールを導入したりします。

4. Guest OS の Uni Processor Kernel 化

現在のところ、Guest OS としては、Linux x86_64 版の Uni Processor Kernel だけがサポートされています。Ubuntu 12.04 x86_64 版の Linux カーネルソースコードを入手し、SMP ではなく Uni Processor 用の kernel としてビルドしてください。ビルドできた kernel を Guest OS の kernel として使用します。従来の KVM コマンドを使用して Guest OS を起動し、ビルドした Uni Processor Kernel を転送しインストールしてください。DEOS 研究開発センターで kernel をビルドした時の config ファイルを、guest-os/.config として参考のために tar.gz ファイルに入れてあります。

異常を検出するデモを実行するためには、第 8 章で説明するように、さらに Guest OS の kernel を修正する必要があります。第 8 章の修正も合わせてここで実行してから kernel をビルドすることで、kernel のビルドとイントールの手間を減らすことも可能です。

このビルドした kernel 中の以下の 3 つのシンボルの仮想アドレスが後で必要になるので、System.map ファイルから調べておいてください。

- init_task
- runqueues

- init_level4_pgt

5. QEMU-KVM の修正

この章では、QEMU-KVM を修正して D-System Monitor を組み込みます。

5-1. 必要なパッケージのインストール

ビルド等で必要となるパッケージを Host OS にインストールしておいてください。

```
$ sudo apt-get install zlib1g-dev
$ sudo apt-get install libglib2.0-dev
$ sudo apt-get install libsdl1.2-dev
$ sudo apt-get install uml-utilities
```

5-2. パッチの適用

qemu-kvm-1.0+noroms のソースコードを入手し、パッチ(qemu-kvm/qemu-kvm-1.0+noroms.patch)を適用してください。

```
$ mkdir work
$ cd work
$ apt-get source qemu-kvm
$ patch –p0 < /path/to/patch/file
```

5-3. アドレスの変更

ソースコード中の以下の定数値を変更してください。それぞれ、上記 4 章でビルドした Guest OS の Uni Processor Kernel の System.map から調べた値を使用します。

qemu-kvm-1.0+noroms/dsysmon/lms.c の以下の 2 行を変更してください。

```
#define INIT_TASK_DEFAULT       0xffffffff81a16300UL    (init_task のアドレス)
#define PER_CPU_RQ_DEFAULT      0xffffffff81a2d420UL    (runqueues のアドレス)
```

qemu-kvm-1.0+noroms/dsysmon/dsm_lib.c の以下の 1 行を変更してください。

```
#define INIT_LEVEL4_PGT     0xffffffff81a0c000UL    (init_level4_pgt のアドレス)
```

ただし、これらの値は、QEMU-KVM 実行後の D-System Monitor 起動時に指定することも可能です。

5-4. QEMU-KVM のビルド

ビルドする前に、qemu-kvm-1.0+noroms/Makefile.objs 中の lms_kern.o の QEMU_CFLAGS の定義(2 行)の -I が、第 4 章でビルドした Guest OS の Uni Processor Kernel のソースコード中の 2 個所の include ディレクトリを参照するように変更しておいてください。

obj ディレクトリを作成し、以下のようにして QEMU-KVM をビルドします。

```
$ mkdir obj
$ cd obj
$ ../qemu-kvm-1.0+noroms/configure --target-list=x86_64-softmmu
$ make
```

これによって、obj/x86_64-softmmu ディレクトリの下に qemu-system-x86_64 として実行モジュールが作成されます。

5-5. シンボリックリンクの作成

ビルドした qemu-system-x86_64 を実行するためには、以下のように、obj/pc-bios ディレクトリの下にシンボリックリンクを作成しておく必要があります。

```
$ cd pc-bios
$ ln -s /usr/share/qemu/*.bin .
```

6. Host OS の KVM カーネル・モジュールの修正

Host OS の KVM カーネル・モジュールを修正する必要があります。パッチ(host-os/kvm.patch)を使用して、Host OS の KVM カーネル・モジュールを修正してください。

Ubuntu 12.04 x86_64 版の kernel ソースコードを入手してビルド環境を整えたのち、KVM カーネル・モジュールだけを変更します。

```
$ cd /path/to/linux-source-3.2.0
$ cd arch/x86/kvm
$ patch -p1 < /path/to/kvm.patch
$ cd ../../..
$ make M=arch/x86/kvm modules
```

これで arch/x86/kvm の下に kvm.ko と kvm-intel.ko が作成されるので、Host OS の /lib/modules/<os-version>/kernel/arch/x86/kvm の下のファイルを置き換え、Host OS を reboot してください。

7. QEMU-KVM の実行

第 4 章～第 6 章までの作業が済むと、QEMU-KVM が実行可能になります。tar.gz ファイルに含まれる start-qemu.sh シェル・スクリプト中の qemu-system-x86_64 実行モジュールの path と Guest OS イメージファイルの場所を適当に変更して

```
$ sudo /path/to/start-qemu.sh
```

を実行すると QEMU-KVM が起動します。start-qemu.sh を実行したターミナルに Guest OS の login プロンプトが表示されると共に、VNC viewer で localhost:5901 に接続すると、Guest OS の画面が表示されるはずです。これで従来の QEMU-KVM と同様に使用可能です。

この時、別のターミナルから

```
$ sudo nc -U /tmp/qemu-monitor
```

を実行すると QEMU-KVM の monitor に接続することができます。QEMU-KVM の monitor が表示する（qemu）のプロンプトから dsysmon コマンドを入力すると、以下のように D-System Monitor を起動することができます。

```
(qemu) dsysmon      <<< 入力行
dsysmon
start dsysmon init_level4_pgt = 0x1a0c000 init_task = 0xffffffff81a16300 runqueues =
0xffffffff81a2d420
```

ここで、表示されている init_level4_pgt 等は、前記の 5.3 で設定した仮想アドレスの値です（ただし、init_level4_pgt は上位の 0xffffffff8 が省略された値として表示されています）。

また、dsysmon コマンドの入力時に、以下のように init_levet4_pgt, init_task, runqueues を指定することも可能です。

```
(qemu) dsysmon 0xffffffff81a0c000 0xffffffff81a16300 0xffffffff81a2d420      <<< 入力行
dsysmon
start dsysmon init_level4_pgt = 0x1a0c000 init_task = 0xffffffff81a16300 runqueues =
0xffffffff81a2d420
```

この時、アドレスは init_levet4_pgt, init_task, runqueues の順に 16 進で指定する必要があります（ソースコード中で定義された値のまま変更不要なアドレスも含めて必ず 3 つのアドレスを指定する必要があります）。

dsysmon コマンドによって D-Syetem Monitor が起動され、QEMU-KVM 中で D-System Monitor 用の thread が複数走り出しますが、監視はまだ開始されていません。監視の開始には、以下の手順に従って GUI 等を準備することが必要になります。

8. Guest OS へのパッチの適用

※この章で説明している Guest OS の修正は、デモを目的として異常な動作をさせるためのものです。デモ用の Guest OS 以外にはこの修正は適用しないでください。

デモを実行するためには、Guest OS に異常な動作をさせる必要があります。そのために、第 4 章で作成した Guest OS 用 Uni Processor Kernel にパッチ(guest-os/linux-source.patch)を適用して修正し、第 9 章で説明するデモ用のカーネル・モジュールが使用可能となるようにする必要があります。DEOS 研究開発センターで kernel をビルドした時の config ファイルを、guest-os/.config として参考のために tar.gz ファイルに入れてあります。

以下のように Guest OS 用 Uni Processor Kernel のソースコードに linux-source.patch を適用してください。

```
$ cd linux-source-3.2.0
$ patch -p1 < /path/to/linux-source.patch
```

その後、第 4 章と同様に kernel をビルドし、Guest OS の Uni Processor Kernel を置き換えます。この修正によって Guest OS 用 Uni Proceeor Kernel 中のシンボル init_levet4_pgt, init_task, runqueues のアドレスがそれまでと異なってしまった場合には、前記 5-3 と 5-4 を再度実行して QEMU-KVM を作成し直す必要があります(第 7 章で説明したように QEMU-KVM の monitor から dsysmon コマンドを実行する時に指定することも可能です)。

9. デモ用のカーネル・モジュール等の作成

※この章で説明しているカーネル・モジュール等は、デモを目的として異常な動作をさせるためのものです。デモ目的以外ではここで説明するカーネル・モジュール等は使用しないでください。

デモの実行のために Guest OS を異常な状態にする必要があり、そのためのカーネル・モジュールを用意しています。この章で作成したカーネル・モジュールを実行するためには、第 8 章に記述した Guest OS の修正が済んでいることが必要です。

9-1. rpldev.ko と rpld の作成

rpldev.ko はキーボード入力データを取得するためのカーネル・モジュールで、rpld は取得したキーボード入力データを他のマシンに転送するプログラムです。この 2 つのプログラムによって、Guest OS のキーボード入力データが他のマシンに転送される状態を作り出します。

ビルドと実行のために libhx-dev が必要になるので、Guest OS と Host OS の双方にインストールしておいてください。

```
$ sudo apt-get install libhx-dev
```

まず、ttyrpld-2.60 のソースコードを Web で見つけてダウンロードしておきます。たとえば、以下の Web ページからダウンロードできます。

http://sourceforge.net/projects/ttyrpld/

ダウンロードした ttyrpld-2.60.tar.bz2 を適当なディレクトリに展開してください。それに対してパッチ(ttyrpld-2.60/ttyrpld-2.60.patch)を適用した後に、rpldev.ko と rpld を作成します。

```
$ cd ttyrpld-2.60
$ patch –p1 < /path/to/ttyrpld-2.60.patch
```

ビルドの前に、ttyrpld-2.60/k_linux-2.6/Makefile の MODULES_DIR を Guest OS の kernel のソースコードを参照できるように適切に変更しておいてください。

```
$ cd ttyrpld-2.60
$ ./configure
$ make
$ cd k_linux-2.6
$ make
```

これで、ttyrpld-2.60/k_linux-2.6 ディレクトリの下に rpldev.ko が作成され、ttyrpld-2.60/user ディレクトリの下に rpld が作成されます。また rpld.conf も必要なので、rpldev.ko, rpld, rpld.conf の 3 ファイルを Guest OS に転送しておきます。rpld.conf は Guest OS 上で rpld と同じディレクトリに置いてください。この rpld.conf ファイル中の NET_ADDR には、Host OS マシンの IP アドレスを指定する必要があります。以下では、Guest OS に/rootkits というディレクトリがあり、そこに rpldev.ko, rpld, rpld.conf が置かれているとします。

9-2. rpld_receiver の作成

tar.gz ファイル中の rpld-receiver ディレクトリの下のファイルを使用して、rpld-receiver を作成します。このプログラムは、9-1 で作成した rpld からのデータを受信するプログラムで、Host OS 上で動かします。

```
$ cd rpld-receiver
$ make
```

9-3. file_rootkit.ko の作成

file_rootkit.ko は異常なファイルメタデータを返すためのカーネル・モジュールで、Guest OS 上で使用します。file_rootkit ディレクトリの下のファイルを使用して、file_rootkit.ko を作成します。file_rootkit/Makefile の MODULES_DIR を Guest OS の kernel のソースコードを参照できるように適切に変更しておいてください。

```
$ cd file_rootkit
$ make
```

これで、file_rootkit.ko が作成されるので、Guest OS の/rootkits ディレクトリに転送しておいてください。

9-4. process_rootkit.ko の作成

process_rootkit.ko は異常なプロセス情報を返すためのカーネル・モジュールで、Guest OS 上で使用します。process_rootkit ディレクトリの下のファイルを使用して、process_rootkit.ko を作成します。process_rootkit/Makefile の MODULES_DIR を Guest OS の kernel のソースコードを参照できるように適切に変更しておいてください。

```
$ cd process_rootkit
$ make
```

これで、process_rootkit.ko が作成されるので、Guest OS の/rootkits ディレクトリに転送しておいてください。

9-5. 監視用ユーザープログラムの準備

3 つの監視機構の内、RootkitLibra の使用の際には Guest OS 上で監視用ユーザープログラムを実行する必要があります。その監視用ユーザープログラム rtkl-collect のソースは rtkl-collect.kvm ディレクトリの下にありますので、以下のようにして rtkl-collect を作成してください。

```
$ cd rtkl-collect
$ make
```

これで、rtkl-collect が作成されるので、Guest OS の適当なディレクトリに転送しておいてください。

10. GUI の準備

GUI は python プログラムなので Host OS に python をインストールしておく必要があります。また、python-wxgtk2.8 と python-matplotlib が必要なので、インストールしておいてください。

```
$ sudo apt-get install python-wxgtk2.8 python-matplotlib
```

GUI の起動は tar.gz ファイル中の GUI ディレクトリの下のシェルスクリプト(DEMO-B-gui.sh)で行ないます。DEMO-B-gui.sh の中の${TOOL_DIR}/dsm-gui.py と${RTKL_DIR}/rpld_receiver がそれぞれの実行ファイルのパスを適切に参照できるように、TOOL_DIR と RTKL_DIR を変更しておく必要があります。

11. デモの実行

第 10 章までの作業が済むと、GUI を使用するデモが実行可能になります。次のように監視機構を動かすことができます。

11-1. GUI の起動

以下のように、QEMU-KVM の monitor プロンプトから dsysmon を起動した後に GUI を起動します。QEMU-KVM が異常終了すると/tmp/dsysmonNNNNN (NNNNN は数字)というディレクトリが残ってしまうことがあります。その場合は、GUI が起動できないので、事前に/tmp/dsysmonNNNNN ディレクトリとその下のすべてのファイルを削除しておいてください。

まず、1 つのターミナルから QEMU-KVM を起動します。

```
$ sudo /path/to/start-qemu.sh
```

次に、別のターミナルから qemu の monitor に接続して dsysmon を起動します（第 7 章で説明したように init_levet4_pgt,等のアドレスを dsysmon 起動時に指定することも可能です）。

```
$ sudo nc -U /tmp/qemu-monitor
(qemu) dsysmon      <<< 入力行
dsysmon
start dsysmon init_level4_pgt = 0x1a0c000 init_task = 0xffffffff81a16300 runqueues =
0xffffffff81a2d420
```

さらに、別のターミナルから GUI を起動します。

```
$ sudo /path/to/DEMO-B-gui.sh
```

これで GUI ウィンドウが表示されるはずです。

11-2. FoxyKBD の実行

まず、Host OS の上で rpld_receiver を起動してください。コマンドのオプションは不要です。

```
$ cd /path/to/directory/of/rpld_receiver
$ ./rpld_receiver &
```

次に、Guest OS 上で rpldev.ko カーネル・モジュールをロードしてから、/rootkits/rpld を実行します。どちらも root 権限で実行してください。その際に、/rootkits/rpld.conf ファイル中の NET_ADDR に Host OS の IP アドレスが設定されていることを確認してください。

```
$ sudo insmod /rootkits/rpldev.ko
$ sudo /rootkits/rpld &
```

これによって、Guest OS 上で/rootkits/rpld がキーボード入力データを取得し、そのデータをネットワーク経由で Host OS 上の rpld_receiver に転送するようになります。

この状態で、GUI 上の「control fkbd injection」の「start」ボタンを押すと大量のキーボード入力が疑似的に生成されます。「stop」ボタンを押すと、キーボード入力の疑似的な生成が止まります。GUI 上の左側のグラフは Guest OS と Host OS の間のネットワーク転送量を表示しているので、キーボード入力の増加とネットワーク転送量が関連していることが確認でき、何らかの異常な振舞いが起こっていることを検出したことになります。

11-3. RootkitLibra の実行

Guest OS 上で file_rootkit.ko カーネル・モジュールをロードしてください。

```
$ sudo insmod /rootkits/file_rootkit.ko
```

これによって、Guest OS 上で/proc/fhide と/proc/fmodify という仮想的なファイルが作成され、以下のような操作で、i-node 番号が InodeNumber であるファイルのメタデータとして異常な値を返すようになります。

```
$ echo InodeNumber > /proc/fhide
$ echo InodeNumber size > /proc/fmodify
```

返される異常値は、Guest OS 上で ls -li 等のコマンドで確認できます。

Host OS から NFS で export したディレクトリを Guest OS から以下のように NFS version 3 でマウントしてください。NFS version 4 は現在のところサポートしていません。NFS マウントされたディレクトリ下のファイルのメタデータを上記の/proc/fhide か/proc/fmodify で異常な値に変更し、さらに、そのディレクトリで rtkl-collect を実行すると、Host OS 側で異常が検出できます。

```
$ sudo mount.nfs -o nfsvers=3 192.168.1.26:/aaa /mnt
$ echo 123456 > /proc/fhide
$ /path/to/rtkl-collect -r /mnt
```

この例では、Host OS の IP アドレスが 192.168.1.26 であり、Host OS の/aaa ディレクトリを Guest OS 側では/mnt に NFS マウント(version 3)しています。echo コマンドで/proc/fhide に 123456 を書き込むことで、その/mnt の下の 123456 という i-node 番号のファイルが見えなくなるようなファイルメタデータの変更を行なっています。rtkl-collect を実行すると Guest OS 側で見えているファイル情報が Host OS 側に送られ、その情報と NFS パケット中のデータを比較することで、異常を検出します。

この後、Host OS の GUI 上の「RootkitLibra」の「Start」ボタンを押すと、5 秒に 1 回ずつ検出された異常が報告されます。

11-4. Waseda LMS の実行

Guest OS 上で process_rootkit.ko カーネル・モジュールをロードしてください。

```
$ sudo insmod /rootkits/process_rootkit.ko
```

これによって、Guest OS 上で/proc/phide という仮想的なファイルが作成され、以下のような操作でプロセス ID が PID のプロセスについて正常なプロセス情報を返さないようになります。

```
$ echo PID > /proc/phide
```

このプロセスは ps コマンド等でも表示されません。

この後、Host OS の GUI 上の「Lightweight Monitoring Service」の「Start」ボタンを押すと、1 秒に 1 回ずつ検出された異常が報告されます。異常なプロセス情報は、そのプロセスが走行状態にある場合のみ検出可能なので、デモの際には、無限ループ等の常に走行状態にあるプロセスを使用する必要があります。

12. おわりに

QEMU-KVM+D-System Monitor を使用すると、Guest OS を外側から監視して、異常な振る舞いを検出することができます。様々な環境で使用されている QEMU-KVM に組み込まれているので、適用領域の拡大が期待されます。