卒業論文　2019 年度（令和元年度）

RDMA を用いた遠隔ベアメタルマシン監視におけるカーネルコンフィグによる差異の吸収

慶應義塾大学 環境情報学部

石川 達敬

徳田・村井・楠本・中村・高汐・バンミーター・植原・三次・中澤・武田
合同研究プロジェクト

2020 年 1 月

卒業論文　2019 年度（令和元年度）

RDMA を用いた遠隔ベアメタルマシン監視におけるカーネルコンフィグによる差異の吸収

論文要旨

アブスト

キーワード

OS

慶應義塾大学 環境情報学部

石川 達敬

Abstract Of Bachelor's Thesis　Academic Year 2019

Title

Summary

abst

Keywords

OS

Bachelor of Arts in Environment and Information Studies
Keio University

Tatsunori Ishikawa

目 次

図 目 次

表 目 次

第1章　序論

1.1　背景

コンピュータの管理者は，動作中，あるいはカーネルパニックなどによって停止したコンピュータの情報を監視・解析することが必要となる場面がある．動作中のコンピュータ自身に対しては，同一ホスト内の top コマンドや ps コマンドを用いて，プロセスの一覧を得たり，gdb コマンドを用いてプロセスをトレースし，プロセスの状態を把握する．ユーザー空間ではなくカーネルのデバッグしたい場合は，kdb と呼ばれるデバッガを，カーネルビルド時に有効にすることで，使用することができる．

論理的に停止したコンピュータに対しては，kdump と呼ばれる機構を通してメモリダンプを静的に解析し，原因の究明をする．また，状態を監視したいホストを物理的なマシンではなく，仮想マシンとして起動し，qemu や Xen などの基盤上で libvmi などを通して状態を解析する手法がすでに存在している．

上述した状況は，コンピュータの管理者，すなわち root 権限を保持している人にとって可能な手法である．

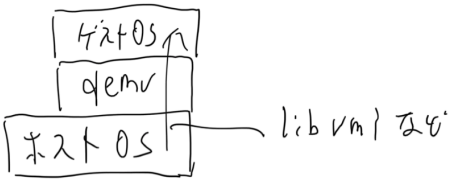
一方で，データセンター管理者など，大量の物理サーバーを保持し，顧客に貸し出している人の場合，上記の手法を使用することはできない．通常は，顧客の情報にアクセスすることはするべきではないが，例えば貸し出しているサーバーがマルウェアなどに感染するなどした場合に事業者としての責任として，原因究明や現状調査のために，解析する必要が出てくる可能性がある．

サーバーを貸し出している会社は，本来はセキュリティ対策として，サーバーを稼働しているオペレーティングシステム上に，セキュリティソフトを入れたいが，大量にあるコンピュータの全てにセキュリティソフトを入れることは容易ではない．当然，顧客から root パスワードを知らされることもないため，ログインをすることもできない．

1.2　課題

1.1 で述べたように，データセンターの管理者は，顧客に貸し出している物理的なコンピュータの内部の状態を知ることはできない．すなわち，死活監視として，ネットワーク的な監視を行うことは可能であるが，コンピュータのオペレーティングシステムにおけるコンテキストを知ることはできない．オペレーティングシステムの内部で起こっていることは，当然，通常は知るべきではないが，マルウェアに感染した場合，意図しない挙動を起こした場合，あるいはカーネルパニックに陥った場合，これらの状態の時に，どの状態でも監視・解析を行うことができるツール

図 1.1: libvmi を用いる際のアーキテクチャ



が存在しない．

VM を用いた解析では，様々な解析手段がすでに豊富にあることは，1.1 で述べ，（hoge で）後述するとおり，VM を管理している物理的なコンピュータに以上が起きた場合に対処ができない．

大量の物理的なコンピュータに対する監視の場合，ネットワーク越しの死活監視，あるいはコンピュータの中でプロセスとしてセキュリティソフト，あるいは状態監視ツールを起動するほかない．この手法は，大量のコンピュータに適用するのは，現実的ではない．第一に顧客に貸し出すサーバーのリソースを微量でも使用してしまうという点．第二に，全てのサーバーにセットアップするのが大変だという点（加筆が必要）

また，このプロセスとして起動する方法は，物理的なコンピュータのオペレーティングシステムがカーネルパニックに陥った際，コアダンプの解析を行う他に解析手段が存在しない．コアダンプの設定を正しく行っていれば，静的ファイルを解析することは可能であるが，ストレージの不足など，不足の事態によって，ダンプを取ることができない可能性も存在する．

さらに，マルウェアなどに感染し，ログインをして解析することが危険にさらされる可能性もある場合，コンピュータにログインすることが推奨されない場合も存在する．

以上のことをまとめて，本研究における解決したい課題として，ネットワーク越しにある物理的なコンピュータに対して，電源さえ入っていればリアルタイムで安全に解析可能なオペレーティングシステムの監視・解析ツールが存在しないこと，と定義する．

1.3 目的

本研究では，大規模データセンターのような，大量かつ様々な環境の物理的なコンピュータを管理する現場において，root 権限がない中でネットワーク越しにあるコンピュータの状態を一元的に把握できることを示すことで，1.2 で述べた問題を解決することを目指す．

この章で述べた様々な環境とは，同じバージョンのオペレーティングシステムでもビルドする際の設定によって，挙動が変わるという意味で述べている．

本研究の実装によって，事前に与える情報が少ない中で，オペレーティングシステムのコンテキストを復元できることを示すために，タスクキューに乗っているプロセスの一覧を取得することを目指す．

1.4 本論文の構成

2 章では、本研究で使用する libtlp と，その基盤として使用している RDMA について述べる．

3 章では、RDMA を通してネットワーク越しにあるコンピュータを監視・解析を行うための実行環境の構成に関して述べる．

4 章では、本研究で実装したものについて述べる（加筆）

5 章では、本研究における評価として，Linux カーネルのバージョンのみが与えられた状態でプロセスリストの一覧を取得できることを示す．

6 章では、本研究に関する結論と，今後の課題について述べる．

第2章　関連技術

本章では，本研究における手法を選ぶに当たって，既存の基盤手法の比較と，基盤技術として使用する RDMA（Remote Direct Memory Address）に関して述べる．

2.1　オペレーティングシステム解析手段

2.1.1　kgdb

2.1.2　コアダンプを用いた解析

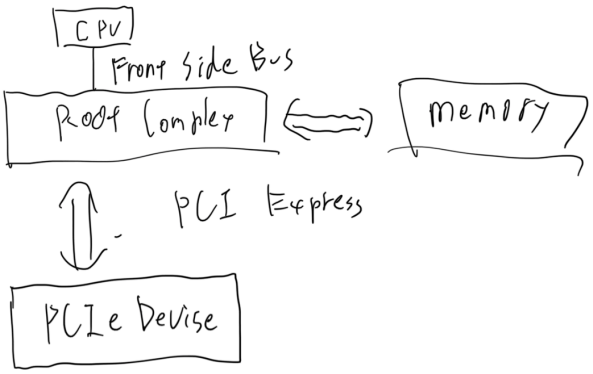
2.1.3　VM を用いた解析

(1)　libvmi

2.2　RDMA

ここに RDMA の説明をかく

図 2.1: PCI Express



2.2.1　Infiniband における RDMA 実装

第3章　アプローチ

1.3で，本研究の目的を，動作中のコンピュータのメモリのダンプをリアルタイムで解析することで，コンピュータの状態をリモートホストから知ることができるようにする，と定義した．

そこで本章では，メモリのダンプをリアルタイムで取得．解析する上で前提となる情報と，この手法における課題について述べる．

3.1　オペレーティングシステムのコンテキスト

コンピュータの状態，すなわちオペレーティングシステムの動作中におけるコンピュータのコンテキストは，コンピュータ内部におけるレジスタの値および，内部から参照できる仮想アドレス空間上に保持されている．その例を下に示す．

あるプロセスを実行する際に，プロセッサはインストラクションポインタレジスタの命令を読み込み，逐次実行をしていく．call命令などで別の関数を呼ぶ際には，その時点におけるインストラクションポインタレジスタの値をメモリ上に退避し，関数が終わった際に，呼び出し元に返るように設定されている．実行コードが整合性を保っているかは，実行可能ファイルを生成したコンパイラの責務なので，本論文では述べないが，プロセッサはプログラムの実行を行う際，レジスタの値を参照，退避，復帰，上書きさせることで，状態を保持，進行させていると言える．

プロセッサがレジスタの値を参照しそれを仕組みは，カーネルのコードを実行する際にも言える．ここでは，オペレーティングシステムから見たコンピュータの状態として，プロセスの切り替え処理，コンテキストスイッチにおける処理の流れを述べる．コンテキストスイッチとは，割り込み処理などによって定期的に呼ばれるプロセススケジューラーから呼ばれる機構である．この機構は，実行中のプロセスの状態，すなわち，各レジスタの値および仮想アドレス空間に関する情報などをカーネルが管理しているメモリ上にあるデータ構造の中に退避する．

本セクションのまとめとして，コンピュータの状態は，ある瞬間においてはレジスタの値であり，この状態を保存する際は，メモリ上にレジスタの値を退避させていることを述べた．

3.1.1　task_struct 構造体

3.1でコンテキストスイッチにおいて，退避されるプロセスの情報は，対応したデータ構造に退避されると述べたが，この時に使用されるデータ構造が `task_struct` 構造体である．`task_struct` 構造体には，一つのプロセスの情報の全て（？）が格納されている．その中には，仮想アドレス空間に関する情報を保持する `mm_struct` 構造体を参照するフィールドも存在する．

コンテキストスイッチ時には，task_struct 構造体に保持されている情報をレジスタに復帰させることで，中断される直前の情報を復元している．

3.1.2　オペレーティングシステムのビルドにおけるコンフィグ

task_struct 構造体をはじめとして，Linux カーネルの変数や型，関数は，様々なアーキテクチャやカーネルコンフィグに対応するため，マクロによって分岐されている．この分岐が確定するのは，Linux カーネルをビルドするときであり，構造体のメンバへのアクセス，関数のアドレスなどはコンパイラが保証している．

実際のカーネルのバイナリは，vmlinux としてコンパイルされた後，strip され bzImage となる．ユーザーが作成したカーネルモジュールなどで関数を呼び出す際は，シンボルとアドレスの変換表である‘/boot/System.map‘を参照し，仮想アドレスを得たのち，実際にメモリにアクセスする際に物理メモリアドレスを算出しアクセスしている．

3.1.3　ホスト自身によるレジスタやシンボルの参照

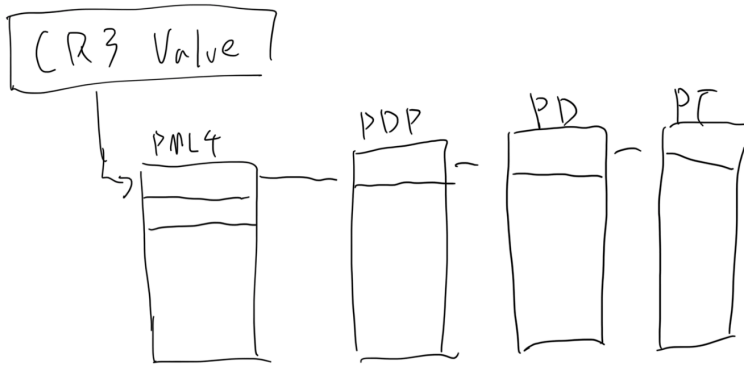
書き直し！！

3.1 で述べたように，オペレーティングシステムでは，レジスタの値などを退避する際，そのプロセッサ自身が’push’ 命令などを用いてメモリにアクセスできる

レジスタを参照して，かつ mmu すら自分で参照ができる．

CPU レジスタの現在の値は直接知ることができないため，例えばプロセスの一覧を取得したい場合は，コンテキストスイッチ時に退避された値を辿っていく必要がある．しかし，上述の通り task_struct はビルドされた際のカーネルコンフィグによって，どのメンバが先頭アドレスからどのオフセットに保持されているかは変動する．

図 3.1: Linux における mmu



3.2　本研究で保持する情報

これまでで述べたように，本研究では，メモリダンプを局所的に取得し，リアルタイムで解析することで，物理的に異なるコンピュータからオペレーティングシステムのコンテキストを復元することを目的と設定した．

3.3　なければならない情報

オペレーティングシステムの状態を保持しているものとして，3.1 で述べたようにレジスタがある．しかし，3.2 で述べたように，現在のレジスタの値は，メモリから知ることはできない．

また，退避された値は，メモリから読み出しを行うが，その際には，

第4章　実装

4.1　実装の概要

本研究では，RDMA を用いて，動作中のマシンのメモリの値を取得していくことで，リモートホストから監視対象ホストのオペレーティングシステムのコンテキストを復元していくことを目指す．この目的を実現するために，本研究では，[1] を用いて実験を行う．

4.2　nettlp

nettlp 本来の目的は，PCIe デバイスの開発プラットフォームである．（リファレンス）

その機能の一つとして，DMA message と ethernet パケットを相互変換する機能がある．2 で述べたが，RDMA の Infiniband 実装は，制限が多い．（もう少し詳しく）nettlp における RDMA では，物理アドレスを指定することで，任意のバイト数（なんバイトだっけ？）の値を取得することが可能である．また，アクセスできないメモリアドレスは存在しない，すなわち全メモリアドレス空間から値を取得することが可能となっている．

nettlp は FPGA ボード上で動作するものであり，これを利用するためのインターフェースとして，libtlp が用意されている．libtlp では，RDMA を用いてメモリダンプを取得するためのインターフェースが関数として用意されている．この関数を含んだヘッダファイルを include し，プログラムから呼び出すことで，メモリアドレスの値が返ってくる．

用意されている関数は，`dma_read` 関数と `dma_write` 関数の二つである．`dma_read` 関数は，値を読みだすための関数であり，呼び出す際に読みたいメモリアドレスを渡す．`dma_write` 関数は，値を指定した物理アドレスに書き込むための関数であり，呼び出す際に，書き込みたいメモリアドレスと値を渡す．

本研究では，`dma_read` 関数のみを用いる．

4.2.1　process-list.c

このファイルでやっていることをかく．本研究の実装が，この process-list.c を拡張したものであるということを書く．

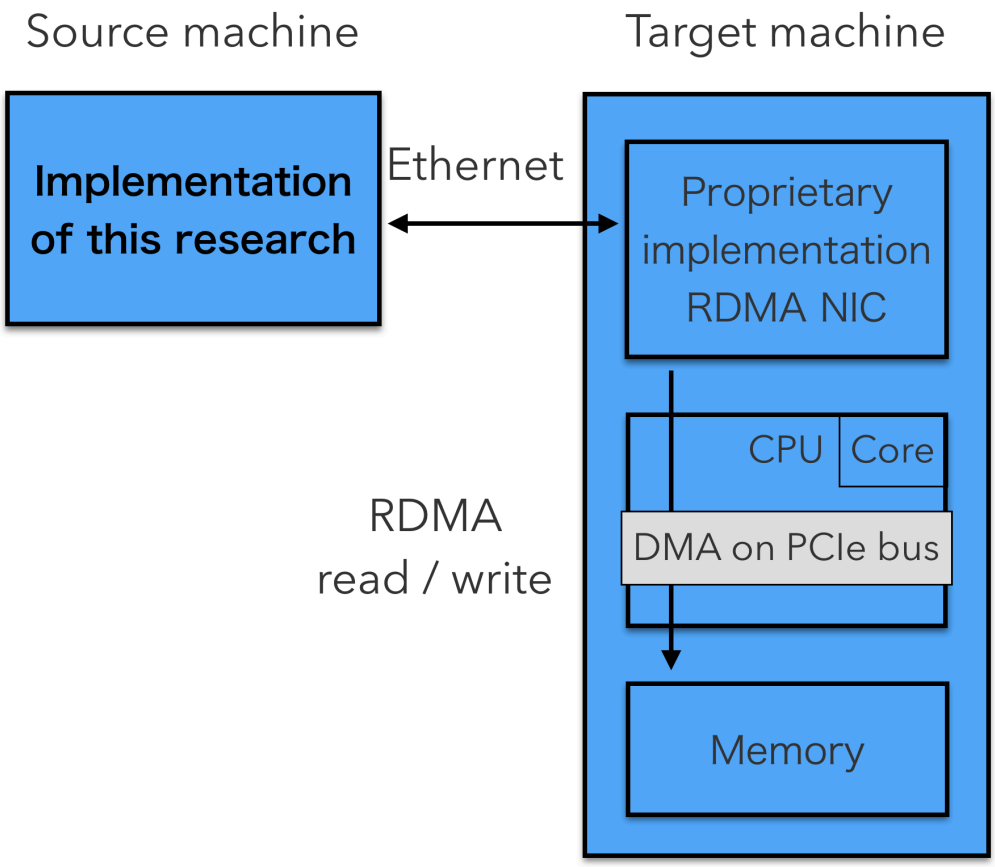
4.3 実験環境

本研究で実装を行う環境は，図 4.1 にあるように，nettlp が書き込まれた FPGA が刺さった監視対象ホストと，本研究における実装を実行するホストの 2 台で構成する．

監視対象ホストは，Linux 4.15.0-72-generic の ubuntu であり，PCIe デバイスとして，nettlp が書き込まれた FPGA ボードが刺さっている．本研究では，FPGA ボードとして，ザイリンクスのやつを使用している．また，この FPGA ボードは，ネットワークインターフェースでもあり，IP アドレスとして，192.168.10.1 を静的に振ってある．

実装を実行するホストは，Linux 4.19.0-6-amd64 の Debian buster であり，光ファイバーケーブル (名称はあとで修正) が刺さる NIC を刺している．以後，実装ホストと呼称する．この NIC には IP アドレスとして，192.168.10.3 を静的に振ってある．監視対象ホストに対して RDMA を実行する際は，`dma_read` 関数，あるいは `dma_wirte` 関数を通して 192.168.10.1 に対して IP パケットを送信している．

図 4.1: 全体



4.4 実装の前提情報

本研究では，Linux カーネルのバージョンは，実装ホストは知っている情報とする．で述べたもののうち，Linux カーネルのバージョンだけは一旦 Given でやる．

また，KASLR（kernel address space layout randomization）を無効にしてある．

4.4.1 KASLR

KASLR の簡単な説明と，無効にする理由と無効にしても良い理由を書く．

4.5 実装の全体

ダンプしてくるということを書く．物理アドレスのマッピングに関しても書く

実験における第一段階として，4.4 で述べたように，監視対象ホストのカーネルコンフィグおよび `init_task` の仮想アドレスを知ることを目指す．そこで，本研究では，この情報をメモリ上から探す．4.7.1 で述べる実装では，取得できるメモリダンプを全て取得し，解析する手法に関して述べる．

次の工程として，それをリストアップ．

次の工程として，収集したカーネルコンフィグを元に手元のコンピュータで Linux カーネルのソースコードに対してプリプロセスの処理を行い，`task_struct` 型を確定する．さらに，ソースコード上にある `__phys_addr` 関数の実体を収集する．

最後に，この工程で得られた情報をもとに，libtlp で提唱されている手法を用いて，プロセスの一覧を正しく取得できることを確認する．

4.6 mem_dump

第一の工程として，メモリの全ての情報を取得する．ソースコードは以下である．この実装を実装ホストで実行し，出力結果をファイルに格納する．この実装では，libtlp を通して，監視対象ホストのメモリを全探索する．この実装の実行には長い時間（何分？）かかるため，アトミックな情報ではない．そのため，ここで取得したメモリダンプは，解析には使えない．ここで取得したメモリダンプは，System.map のうち，`init_task` が配置されている仮想アドレス空間に関する情報および，Linux カーネル 4.15.0 におけるカーネルコンフィグに関する情報を収集するためのものである．

実行方法

```
./dump_mem > ~/Desktop/work9/dump
```

4.7 init_task およびカーネルコンフィグの取得

この章では，**??**で取得した，メモリダンプから，必要な情報を取得する．前処理として，以下に示すように，メモリダンプに対して strings コマンドをかけ，検索しやすくする．以後のファイル読み込みには，`str_list` を用いる．

strings

```
strings dump > str_list
```

4.7.1 init_task に関する情報の復元

で作成した `str_list` に対して，`init_task` に関する行を grep コマンドを用いて探す．

4.7.2 カーネルコンフィグの復元

Linux カーネル 4.15.0 におけるカーネルコンフィグの一覧は，下に示す通りである（あとではるかも）

これらのコンフィグに関する情報を以下のスクリプトで読み出す．

カーネルコンフィグには，各設定項目に対する値として，y,m や文字列，数値などがあり，設定しない項目については，その行がコメント行になるのに加えて，`is not set` という文言が付け足される．このスクリプトによってカーネルコンフィグ，ビルド時における `.config` というファイルを生成する．以下の章では，このファイルを設定ファイルとして，ビルドを行う．

search config script

未実装

4.8 Linux カーネルをプリプロセッサに通す

この工程では，収集したカーネルコンフィグを元に手元のコンピュータで Linux カーネルのソースコードに対してプリプロセスの処理を行い，`task_struct` 型，および`__phys_addr` 関数など，`process-list.c` の影響のあるソースコードを確定する．

さらに，ソースコード上にある`__phys_addr` の実体を収集する部分に関する実装をより詳しく書く．

makeoldconfig

```
cd path/to/kernelsource
cp path/to/さっき拾った/config path/to/kernelsource/.config
make oldconfig
# save-temps オプションを設定
make -j9
```

4.9 ていうか実装をカーネルに埋め込めばいいのでは

最後に，集められたデータをもとに，process-list を改造したものに関する説明をここに書く

4.10 実装のまとめ

この章は，相当長くなるので，まとめを書く．

第5章　評価

5.1　評価手法

カーネルのバージョンのみわかる状態から，正しく ps aux と同じような出力を得られるかどうかを評価とする．あと，かかった時間に関して書く（時間は本質ではないので，書く必要ないかも）

5.2　評価

5.2.1　時間

初期段階で時間はかかるが，それは問題ではない．大事なのは，どんなカーネルコンフィグを持つ Linux でも，解析が可能になるという点．

5.3　評価

未評価

第6章　まとめと結論

6.1　まとめ

各章の振り返り

6.2　結論

結論は俺の知見，考察．結局こうだったを述べる．こういうところでは使えた，こういうところではダメだった．

6.3　今後の課題

特にカーネルバージョンの特定と KASLR のバイパスについて書く．

6.3.1　KASLR

これは本来，ASLR に実装されたものであり，それのカーネルバージョンである．内部からの攻撃に向けた防御策であるため，この研究の手法であればバイパスは可能であると考えられる．なぜなら，メモリ保護機能，すなわちアクセスできるメモリアドレスの制限を受けないレイヤーで動作する環境だから．

6.3.2　セキュリティ的な課題

あるだけで危ない

謝辞

アドバイスをくれた全員に感謝

参考文献

[1] Nettlp: A development platform for pcie devices in software interacting with hardware. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, Santa Clara, CA, February 2020. USENIX Association.