RDMAを用いたプロセスの物理隔離実行の提案

安斉 周[1]　味曽野 雅史[1]　中村 遼[1]　空閑 洋平[1]　品川 高廣[1]

概要：近年，仮想化技術やハードウェア支援機能を用いたアプリケーションの隔離実行の手法が数多く提案されている．しかし，隔離実行環境を実現するソフトウェアに脆弱性があった場合には，システム全体が乗っ取られてしまう可能性があるほか，Spectre に代表されるハードウェアの脆弱性を利用したサイドチャネル攻撃を完全に防止することは極めて難しい．本研究では，隔離対象のプロセスのみを 1 台の物理マシン上で独立して実行させる物理隔離実行の手法を提案する．隔離対象のプロセスが発行するシステムコールを遠隔のホストマシンから RDMA で処理することにより，隔離マシン上で動作する特権ソフトウェアを最小化し，ローカル権限昇格のアタック・サーフェスを極限まで削減する．また，実行環境を物理的に隔離することによって，サイドチャネル攻撃の可能性を大幅に低減させる．本稿では，NetTLP の RDMA 機能を用いて初歩的なシステムコールの遠隔実行を実装した内容を報告する．

1. はじめに

セキュリティ意識の高まりにより，アプリケーションを隔離実行する環境の必要性は日々高まっている．隔離実行を実現する手法としては，システムコールの仲介によるサンドボックスをはじめとして，仮想マシンを利用した手法やコンテナを利用した手法など，様々な手法が提案されてきた．また，近年は Intel SGX や Arm TrustZone のような隔離実行のための様々なハードウェア支援機能が利用可能になっている．これらの手法は，信頼できないソフトウェアを隔離して安全に実行できることから，クラウドコンピューティングをはじめとして広く注目を集めている．

しかし，攻撃手法の進化により，これらの手法も必ずしも安全とは言い切れなくなっている．例えばコンテナによる隔離は，OS カーネルに脆弱性があった場合には，ローカル権限昇格によりホスト全体が乗っ取られてしまうことが知られている．仮想マシンによる隔離はコンテナによる隔離よりも堅牢であるとされているが，仮想マシンモニタも複雑化しているため，脆弱性をなくすことは容易ではない．ハードウェア支援機能による隔離実行環境であっても，様々なセキュリティ上の脆弱性が指摘されており [1]，安全な隔離実行環境の構築には依然として課題がある．

脆弱性を突いた攻撃を防止するためには，アタック・サーフェスと呼ばれる攻撃される可能性のある個所を減らすことが有効である．例えば，コンテナにおいては発行可能なシステムコールの種類や時間を制限する手法 [2] が提案されているほか，仮想マシン環境においては仮想マシンモニタのサイズや機能を削減する手法 [3] や Library OS を用いる手法 [4] などが提案されている．しかし，いずれの手法でも隔離実行環境を実現するために特権レベルで動作するソフトウェアが多く残っており，それらの脆弱性をなくすことは難しい．さらに，Spectre のような過渡実行攻撃 [5] は今後も続くことが予想されるが，そのようなハードウェアの脆弱性を利用したサイドチャネル攻撃を防止することは極めて難しい．

本研究では，アプリケーションを実行するマシンを OS 機能を提供するマシンから物理的に分離することで強力な隔離を実現する物理隔離実行の手法を提案する．隔離マシン上で動作するアプリケーションが発行したシステムコールの大部分を，遠隔のホストマシンから RDMA で処理することにより，隔離マシン上で動作する特権ソフトウェアを最小化し，ローカル権限昇格のアタック・サーフェスを極限まで削減したサンドボックスを実現する．また，実行環境を物理的に隔離することによって，Spectre のようなサイドチャネル攻撃の可能性を大幅に低減させる．

提案手法の実装には，NetTLP [6] を用いることで，隔離マシン上でドライバなどの特権ソフトウェアのサポートなしで RDMA 機能や割り込み機能を実現する．

本稿では，プロトタイプ実装として隔離マシン上で Linux を動作させ，RDMA 機能を用いて初歩的なシステムコールの遠隔実行を実装した内容を報告する．

[1] 東京大学
The University of Tokyo

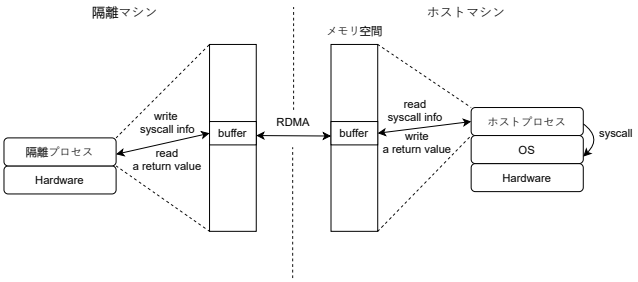

図 1 提案手法の構成

2. 提案手法

図 1 に提案手法の概要を示す．提案手法では，隔離実行するアプリケーションが動作するマシン（隔離マシン）を，システムコールを実行するマシン（ホストマシン）から物理的に隔離する．隔離マシン上のプロセス（隔離プロセス）がシステムコールを発行する際は，隔離マシンのユーザ空間にマッピングされたメモリ領域に確保したバッファにシステムコール番号や引数を格納してホストマシンに通知する．ホストマシン上のプロセス（ホストプロセス）は，RDMA でバッファの内容を読み込み，ホスト OS でシステムコールを実行し，その結果を RDMA で隔離プロセスに反映する．

隔離マシン上の特権モードで動作するコードを最小化するために，スケジューリングやメモリ管理など隔離マシン上のカーネル空間で処理する内容も可能な限り RDMA で遠隔から行い，コンテキストスイッチやページテーブルの切り替えなど CPU に対する処理が必要なコードのみ隔離マシン上で実行する．ディスクやネットワークなどの I/O は，RDMA を用いてユーザ空間のバッファに直接読み書きすることで，カーネル空間を経由せずに実行する．

システムコールの実行は，FlexSC [7] のように非同期的に実行することで，RDMA によるレイテンシを隠蔽する．また，システムコールは必要に応じてフィルタリングをおこなうことで，サンドボックスの機能を実現するとともに，ホスト OS への攻撃を防止する．

3. 実装

実装には NetTLP を利用した．NetTLP は，Transaction Layer Packet (TLP) という PCIe バスでやり取りされるパケットを Ethernet 経由で転送することで，PCIe デバイスを遠隔でソフトウェア的に実装できるシステムである．NetTLP を用いることで，隔離マシン上での OS やデバイスドライバのサポートなしでホストマシンからの RDMA や隔離マシンからホストマシンへの通知をおこなうことができる．現在は，プロトタイプ実装として隔離マシンとホストマシンの両方で Linux 環境を使用して開発を進めており，open, close, read, write といった初歩的なシステムコールを実装して動作することを確認している．

例として，隔離マシンで read システムコールを発行したときの流れを以下に示す．

(1) 隔離プロセスが read(fd, buf, count) を実行する
(2) buf のアドレスを物理アドレスに変換し，システムコールバッファに引数などを書き込む
(3) システムコールバッファのアドレスを，NetTLP が監視する特定メモリ領域（BAR 空間）に書き込む
(4) BAR 空間への書き込みに対して，ホストプロセスのコールバック関数が実行される
(5) コールバック関数は，システムコールバッファの内容を RDMA read で読み込む
(6) ホストプロセスは，システムコールバッファの内容を調べて，対応する read システムコールを発行する
(7) read したデータを RDMA write で隔離プロセスの buf に書き込む
(8) システムコールの返り値をシステムコールバッファに書き込む

4. まとめと今後の予定

本稿では，RDMA を用いて隔離プロセスのシステムコール処理を遠隔のホストマシンへ委託することで，プロセスを物理隔離実行するシステムを提案した．今後は，実装するシステムコールの策定や mmap など複雑なシステムコールの実装を行うほか，隔離マシン上でブートして最低限の隔離実行環境だけを構築するベアメタル OS の実装を行う予定である．

参考文献

[1] Fei, S. et al.: Security Vulnerabilities of SGX and Countermeasures: A Survey, *ACM Computing Surveys*, Vol. 54, No. 6 (2021).
[2] Ghavamnia, S. et al.: Temporal System Call Specialization for Attack Surface Reduction, *Proc. 29th USENIX Security Symposium* (2020).
[3] Agache, A. et al.: Firecracker: Lightweight Virtualization for Serverless Applications, *Proc. 17th USENIX Symposium on Networked Systems Design and Implementation* (2020).
[4] Olivier, P. et al.: A Syscall-Level Binary-Compatible Unikernel, *IEEE Transactions on Computers*, pp. 1–1 (2021).
[5] Xiong, W. and Szefer, J.: Survey of Transient Execution Attacks and Their Mitigations, *ACM Computing Surveys*, Vol. 54, No. 3 (2021).
[6] Kuga, Y. et al.: NetTLP: A Development Platform for PCIe devices in Software Interacting with Hardware, *Proc. 17th USENIX Symposium on Networked Systems Design and Implementation*, pp. 141–155 (2020).
[7] Soares, L. and Stumm, M.: FlexSC: Flexible System Call Scheduling with Exception-Less System Calls, *Proc. 9th USENIX Symposium on Operating Systems Design and Implementation* (2010).