2007－OS－104 (7)
2007／1／31

リファレンスモニタの多重化法の提案

川崎 仁嗣 † 鈴木 勝博 † 阿部 洋丈 ‡ 加藤 和彦 †,‡

† 筑波大学 システム情報工学研究科 ‡ 科学技術振興機構

要 旨

リファレンスモニタを利用したシステムにはサンドボックスやIDSがある．サンドボックスやIDSは組み合わせて用いられることがあるが，一般的に用いられているリファレンスモニタは多重化可能なようには設計されていない．これは，リファレンスモニタが利用しているプロセス監視用システムコールの設計による制限である．本研究ではプロセス監視用システムコールの動作をエミュレーションすることにより，リファレンスモニタを用いた防御用システムの多重化を実現する方法について述べる．

A proposal for the method of nested reference monitor

Satoshi Kawasaki† Katsuhiro Suzuki† Hirotake Abe‡ Kazuhiko Kato†,‡

† University of Tsukuba, Graduate School of Systems and Information Engineering
‡ Japan Science and Technology Agency

Abstract

The system that use reference monitor is sandboxes and IDSs. Although the combination of sandboxes and IDSs might be used, the reference monitor that is used generally is not designed for nesting. This is a limitation in the system call for the process monitoring that the reference monitor uses. The method proposed in this paper emulates system call for process monitoring and enables nested defense systems using reference monitor.

1 はじめに

近年，個人情報の保護に対する関心が高まっており，サーバ上の情報を狙った攻撃などへの対策が重要となっている．その一方で，ソフトウェアの脆弱性が発見され，それに対処するパッチなどが配布される前にその脆弱性を悪用して攻撃を行う，いわゆるゼロデイアタック [1] が広まりつつある．サーバソフトウェアの脆弱性を完全になくすことは難しく，適正な運用が行われていても常にこれらの脅威に晒されている．このような脅威に対する根本的な解決策は，その脆弱性を修正したり回避することである．しかし，そのような対策を行うにはある程度の時間を要する．そのため，ゼロデイアタックへの対策には別の手法が用いられる．サーバソフトウェアとは別に，セキュリティを向上するためのシステム（以降，防御用システムと呼ぶ）を用いることで改善できる．防御用システムとしては，IDSやサンドボックスがよく利用される．

IDSはネットワーク型とホスト型の2種類に大別され，本研究ではホスト型IDSを対象とする．ホスト型IDSとは監視対象となるホスト内で動作し，ファイルの状態や動作しているプログラムを監視するシステムである．ホスト型IDSの様子を図1に示す．この種類のIDSとしては，Tripwire[2] やDragon Squire[3] がある．これらのシステムは，ファ

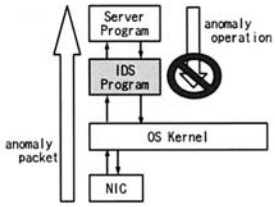


図 1 ホスト型 IDS

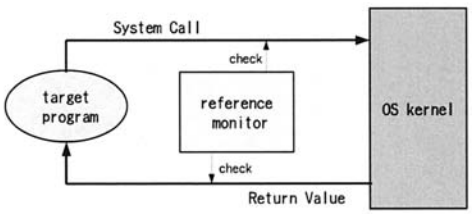


図 2 リファレンスモニタの概観

イルのサイズや最終アクセス時刻，システムログを監視する．しかしファイルの状態を調べる方法では，実際に不正行為が行われてしまう．そこで，プログラムの動作を監視し正常動作時には示さない動作を検知する方法を利用した IDS がある．これは異常検知型 IDS と呼ばれ，この手法を用いた IDS としては Wagner らのシステム [4] や Giffin らのシステム [5]，阿部らのシステム [6] などがある．これらの IDS では正常時とは異なる動作を検出することで攻撃を検知するため，サーバソフトウェアに脆弱性があるかどうかは問題とならない．

サンドボックスはサーバソフトウェアの環境をサーバ全体の環境から隔離し，内部で起こる事象による影響を外に及ぼさないようにするシステムである．サンドボックス内の環境では，ファイルシステムへのアクセスを特定のディレクトリ以下に制限したり，サンドボックス外にあるプロセスを隠し，アクセス出来ないようにすることで実現されている．サンドボックスとしては，UNIX の chroot や Free BSD の jail[7]，LIDS[8]，Solaris Zone[9]，SoftwarePot[10] などがある．サーバソフトウェアをサンドボックスの環境に閉じ込めることによって，サーバプログラムが乗っ取られてしまっても影響を及ぼす範囲をサンドボックス内に抑えることが出来る．

これらの防御用システムを実装する場合，リファレンスモニタの手法がしばしば用いられる．リファレンスモニタは監視対象となるプログラムの動きを見張り，動作の許可や棄却を行う．本研究ではリファレンスモニタの中でも，プロセスの発行するシステムコールをチェックする方式のものを対象とする（図 2）．プロセスはシステム上のリソース（ファイルやソケットなど）へのアクセスなど，OS の提供する機能を利用するためにはシステムコールを呼び出さなければならない．そのため，プロセスの動作を監視するには監視対象プロセスの発行するシステムコールをチェックすれば良い．

本稿では，防御用システムを組み合わせて利用する際の問題点を挙げ，それを解決するための手法を提案する．まず，2 節では防御用システムを組み合わせる必要性について述べる．その後，3 節にて提案手法を示し，4 節にて実装の詳細を述べる．5 節では多重化により費されるコストを測定し，考察した．

2　防御用システムの多重化

IDS を利用したサーバソフトウェアの保護では，サーバ自体への侵入を防ぐことが出来る．しかし，IDS の持つ異常検知の性質を逆手に利用して正常動作に見せかけながら攻撃を行うことが可能であり，Wagner らの方法 [11] が示されている．それゆえ，IDS を単体で用いていても完全に侵入を防ぐことは出来ない．また，IDS による保護が不完全である点だけでなく，IDS に脆弱性があった場合が考えられる．IDS によりサーバソフトウェアが保護されていたとしても，IDS 自体に脆弱性があれば IDS を乗っ取られてしまう．もし侵入を許してしまった場合には，サーバへの被害を最小限に抑えるための対策も必要である．

通常，IDS はサーバソフトウェアよりも広範なアクセス権を持っていることが多い．そこで例えば，IDS をサンドボックスの環境に閉じ込めることで，IDS 自体に侵入されてしまったとしても影響を抑えることが可能である．また，防御用システムのプログラムに悪意のあるコードが含まれていても，外部に影響を与えることはない．つまり，いくつかの防御用システムを利用することを考えた場合，一番外側の防御用システム（例えばサンドボックス）が正常な動作をすると分かっていれば，その中で動かす防御用システムに未知の問題があったとしても外部

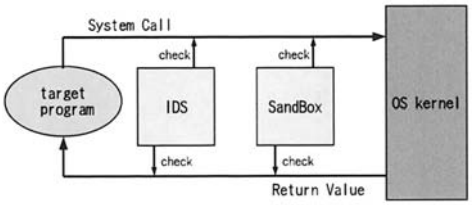


図 3 リファレンスモニタのネスト化

の環境には被害を与えずに済む．まとめると，IDS などの侵入される前の対策と，サンドボックスなどの侵入された後の対策があり，これらを組み合わせることにより安全性を高めることが可能である．

このように，防御用システムは組み合わせて利用されることがあるのだが，一般的に用いられているリファレンスモニタは多重化可能なようには設計されていない．リファレンスモニタを多重化するというのは，例えば，サンドボックスにより隔離している環境の中でサーバソフトウェアを動作させ，そのサーバソフトウェアを IDS で監視する場合，サンドボックスと IDS はネスト関係になる（図 3）．つまり，サンドボックスは IDS のプロセスとサーバのプロセスを監視し，IDS はサーバのプロセスを監視する．ここで問題となるのは，一つのプロセスの監視を行うことの出来るプロセスが一つに限られるという制限である．これを解決するためには，一つのプロセスの監視を複数のプロセスから行えるようにする必要がある．次の節ではリファレンスモニタを一部拡張することによって，複数の防御用システムを多段に動作可能にする手法を述べる．

3 提案手法

監視対象プロセスの発行するシステムコールをチェックする手段としてはいくつか挙げることが出来るが，OS の提供する監視機能を利用する場合，UNIX 系 OS で広く採用されているプロセス監視のためのシステムコール（ptrace）や Solaris 10 で実装された DTrace[12] を利用する方法がある．また，システムコールテーブルを書き換えることにより OS の提供する監視機能を利用せずに実装することも出来る．本研究では，ptrace システムコールを利用してリファレンスモニタを実装した．その理由として，ユーザモードで実行可能なこと，UNIX 系 OS に広く採用されていることが挙げられる．

ptrace システムコールには大きく分けて 4 つの役割がある．

1. 自分自身をトレース状態にする．
   プロセス監視システムコールを呼び出したプロセス，つまり自分自身をトレース状態にするようカーネルに通知する．トレース状態のプロセスは，システムコールの呼出し直前（及び，直後）に親プロセスに対して通知（シグナルの送信）を行い，停止する．
2. メモリ領域やレジスタの読み書きを行う．
   トレース状態にあり停止している子プロセスのメモリ領域やレジスタの値を読み込んで調べたり，任意の値を書き込むことが出来る．この機能を利用すると，システムコールの呼出し直前で停止した際に，システムコール番号や引数を取得したり，書き換えることも出来る．また同様に，システムコール呼出し直後で停止した際に，システムコールの結果を取得したり，書き換えを行うことが出来る．
3. 停止している子プロセスを再開させる．
   トレース状態にある子プロセスは，システムコール呼出し直前などの特定の動作により停止する．子プロセスの状態を調べた後は子プロセスを再開させる．オプションにより子プロセスがどのような動作をしたときに再び停止させるかを指定することが出来る．1 命令実行後に停止させたり，システムコールを呼び出す直前，直後に停止させることも可能である．
4. 指定したプロセスをアタッチ／デタッチする．
   アタッチされたプロセスは，アタッチしたプロセスの子プロセスとなり，トレースされた状態になる．トレース状態の子プロセスをデタッチすると，トレース状態は解除される．

プロセス監視システムコールを用いたリファレンスモニタの場合，すでに監視されているプロセスに対して多重にプロセス監視システムコールを使うことが出来ない．プロセスを監視中にシステムコールの呼出しが行われた場合，カーネルはそのプロセスの親プロセスに対してシグナルを送り通知するが，あるプロセスに対して親プロセスは 1 つしか存在しない．したがって，リファレンスモニタを複数動作させた場合実際に通知を受けとることのできるプロ

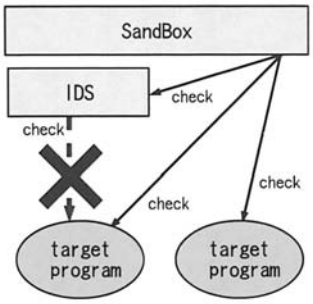


図 4 複数のリファレンスモニタによる監視

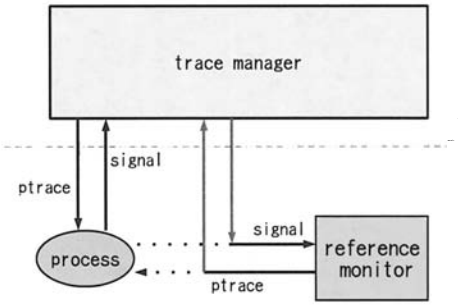


図 5 トレースマネージャ

セスは 1 つだけであり，通知を受けることのできなかったプロセスは監視動作を行うことが出来ない．ただし，1 つのプロセスが複数の子プロセスに対してプロセス監視を行うことは出来る．この場合，複数の子プロセスからの通知を 1 つの親プロセスが処理する形となる．これをまとめると，監視する側と監視される側の関係が 1 対 n であると言える．例えば，図 4 のようにサンドボックスが監視を行っているプロセスは，IDS から監視を行うことは出来ない．

そこで，プロセス監視システムコールを直接利用するのではなく，プロセスの監視を専門に行うトレースマネージャ（図 5）を用意し，実際のプロセス監視はトレースマネージャが一括して行うことにした．また，トレースマネージャはプロセス監視システムコールが呼び出された際に，どのプロセスがどのプロセスを監視しようとしているのかを認識する．監視されているプロセスがシステムコールを発行した場合，まずトレースマネージャに通知が行われる．通知を受けたトレースマネージャが監視する側のプロセスに対して通知を行う．通知を行う際にはその通知が実際の監視対象プロセスから送られてきたように見せかけることで，トレースマネージャ配下のリファレンスモニタは既存の実装のままで動作させることが可能である．これはつまり，トレースマネージャによりプロセス監視システムコール動作をエミュレーションしていることになる．

これにより，監視する側と監視される側の関係が n 対 m であっても正常に動作させることが可能である．トレースマネージャを追加したリファレンスモニタを用い，サンドボックス内でホスト型 IDS を動作させたときを考えると，図 6 のような状態になる．リファレンスモニタ上で動くプロセスは，図 6 の左に示すようなプロセスの親子関係であることを

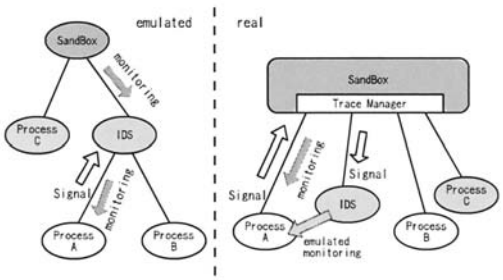


図 6 提案手法型リファレンスモニタ

期待している．しかし，実際には図 6 の右に示すように，全てのプロセスはサンドボックスの直接の子プロセスとなっている．トレースマネージャがプロセス監視のエミュレーションを行うため，IDS はプロセス A が子プロセスであるかのように動作することが出来る．

4 実装

リファレンスモニタはその上で動作するプログラムから透明であるよう，いくつかのシステムコールにフックをかけてエミュレーションに必要な処理を行う．エミュレーションが必要なのは，fork システムコールに代表されるプロセス生成の操作や wait システムコールに代表されるプロセス間同期の操作，sigaction システムコールに代表されるシグナルハンドラの操作，そして ptrace システムコールなどのプロセス監視システムコールである．Linux の場合，エミュレーションの必要なシステムコールは表 1 に示す通りである．

表 1 エミュレーションの必要なシステムコール

プロセス生成	fork
	vfork
	clone
プロセス間同期	wait
	waitpid
	waitid
	wait3
	wait4
シグナルハンドラ操作	sigaction
	signal
プロセス監視	ptrace

4.1 プロセス生成

プロセス生成の操作（fork など）が行われる場合，新たに生成されたプロセスはトレース状態ではないので，生成されたプロセスをトレース状態にする必要がある．あるプロセスをトレース状態にするためには，ptrace システムコールでアタッチする方法がある．しかしそれではプロセスが生成されてからアタッチされるまでの間は監視出来ない．そこで，プロセスの生成を行うシステムコールを呼び出した時は clone システムコールに書き換え，呼出元プロセスのトレース状態をコピーするように指定するフラグを追加する．親プロセスはトレース状態であるので，生成されたプロセスもトレース状態のままになる．

4.2 プロセス間同期

プロセス間同期の操作（wait など）が行われる場合，通常は子プロセスの状態変化を検知することができる．しかし，子プロセスがリファレンスモニタ上で動作しているとリファレンスモニタ側が状態変化を検知してしまうため，プロセス間同期の操作を行ったプロセスはブロック状態のまま待ちつづけてしまう．そこで，リファレンスモニタ側でプロセス間同期の操作を検知した時はどの子プロセスを待っているのか記録し，条件にあった子プロセスで状態変化があった場合，そのことを親プロセスに通知する．親プロセスは通知を受けブロック状態を解除される．

4.3 シグナルハンドラ操作

シグナルハンドラの操作（sigaction など）が行われると，そのプロセスがシグナルを受け取ったときの動作を変更することが出来る．しかし，リファレンスモニタ上で動作している場合，シグナルを受け取るとそのことがリファレンスモニタに通知されてプロセスは停止する．受け取ったシグナルの処理はリファレンスモニタ側で行われるため，シグナルハンドラの操作を検知した場合，その情報を記録しておく．プロセスがシグナルを受け取り停止した場合，リファレンスモニタはそのシグナルに対応する動作が設定されているか確認し，設定されていればプロセスの動作を再開させシグナルを渡す．設定されていなければ，既定の動作（プロセスの停止など）を行う．

4.4 プロセス監視

ptrace システムコールによるプロセス監視では，親プロセスが wait システムコールにより子プロセスからのシグナルを待ち，子プロセスはシステムコールの直前，および直後に停止し，シグナルが親プロセスに送られる．このシグナルにより親プロセスは wait から復帰し，子プロセスの状態を調べる．子プロセスのレジスタやメモリの状態を取得し，システムコールの種類や引数などを調べることが出来る．ここで得られたシステムコール番号より，それぞれのシステムコールに対応する監視処理を行う．例えば，サンドボックスの場合は open システムコールに代表されるファイル操作関連のシステムコールの引数を監視する．そして，アクセスポリシーに基づいてファイルへのアクセスを許可，または拒否する．これらの処理が終わったら ptrace システムコールを呼出し，子プロセスの動作を再開させる．以上の動作の流れを図 7 に示す．

ptrace によるプロセス監視を開始するためには 2 通りの方法があり，子プロセス側で ptrace システムコールを呼出し，自分自身をトレース状態にする方法と，指定したプロセスへアタッチして子プロセス化しトレース状態にする方法がある．

監視されるプロセスはシグナルをトレースマネージャに対して送ってくる．そこで，トレースマネージャは監視を行っているプロセスに対してシグナルを伝える．そのためには，トレースマネージャがプ

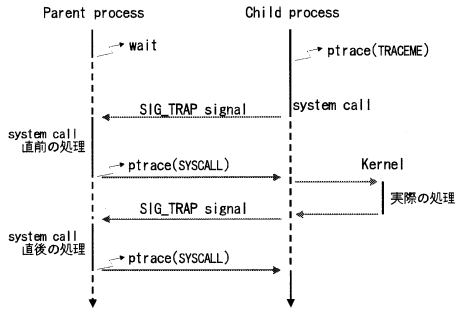


図 7 通常のプロセス監視の流れ

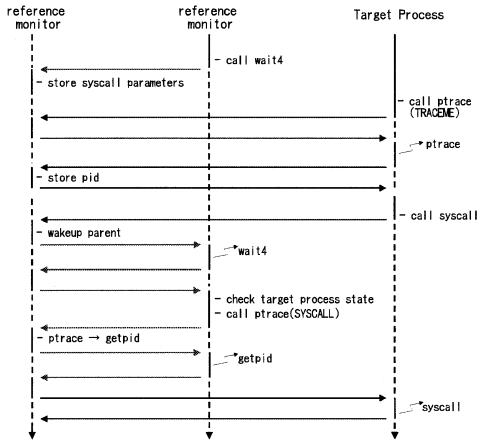


図 8 プロセス監視のエミュレーション

ロセス監視の対応表を構成する必要がある．それには，図7の TRACEME オプション付きの ptrace システムコールを検知すれば良い．そして，システムコールを呼び出したプロセスと，対応する親プロセスの組を記憶しておく．これらの情報を利用して，トレースされているプロセスからのシグナルをどのプロセスに配送すれば良いのか判断出来る．対応関係を記憶した後は，ptrace システムコールを副作用のないシステムコール呼出し（例えば，getpid）に書き換えてしまう．何故ならばこの ptrace システムコールはエミュレーションされるものであり，実際に ptrace システムコールが実行されてしまうとトレースマネージャがトレースを行えなくなってしまうか，ptrace システムコール自体が失敗してエラーを返してしまう．

監視されているプロセスがシステムコールを発行すると，システムコール入口で停止し，トレースマネージャにシグナルを送る．トレースマネージャはそのプロセスを監視していることになっているプロセスに対して，まず，シグナルを送り wait システムコールによるブロック状態から復帰させる．すると親プロセスは wait システムコールから戻り，その時点で停止する．ここで wait の返り値を，監視しているプロセスのプロセス ID に書き換える．これにより親プロセスは監視していたプロセスが停止してシグナルを送ってきたように見える．親プロセスは子プロセスの状態を取得し，例えば IDS 動作に必要な処理を行った後，ptrace システムコールを呼出し子プロセスの実行を再開しようとする．しかし，前に述べた通りこの ptrace システムコールはエミュレーションされているるものである．そこで，ptrace システムコールを副作用のないシステムコール呼出しに書き換え，子プロセスの再開はトレースマネージャ側で行う．以上の処理の流れを図8に示す．

5 評価

提案手法による実装を行い，多重化を行うためにかかるコストを調査した．実装の環境は Linux 2.6 上であり，調査を行った環境は Pentium 4 1.8GHz である．調査方法としては，ptrace を利用せずに実行した場合（(1)native），ptrace システムコールによりトレースされている場合（(2)ptrace），リファレンスモニタを適用して実行した場合（(3)reference monitor），リファレンスモニタを2段適用して実行した場合（(4)nested reference monitor）のそれぞれでシステムコールの処理にかかる時間を測定し，比較した．なお，ここで言うシステムコールの処理にかかる時間とは，プロセスがシステムコールを呼び出してから結果が帰ってくるまでの時間のことである．実際に測定した結果を表2に示す．

リファレンスモニタを適用した場合，すべてのシステムコール呼出しにおいてシグナルが発生するため，それに対する処理が行われる．したがって，getpid システムコールのようにエミュレーション処理を行わないシステムコールでも，オーバヘッドが出てしまう．また，fork + wait，sigaction システムコールの処理ではリファレンスモニタ側でエミュレーション処理が行われるためその分余計に時間が

表 2 システムコール処理時間

system call	(1)native	(2)ptrace	(3)reference monitor	(4)nested reference monitor
getpid	0.024	0.030	0.036	0.145
read	3.3	44	125	1090
fork + wait	519	673	1586	2743
sigaction	1	20	133	657
ptrace (TRACEME)	2.7	–	1448	3980
ptrace (SYSCALL)	7.6	–	172	1124

(unit: micro sec.)

かかる．特に wait システムコールはリファレンスモニタ側で条件チェックなど複雑な処理を行うため，ptrace による監視を行わない場合と比較して特に遅くなっている．なお，fork + wait の評価は次のように行った．fork により生成された子プロセスは直ちに終了し，親プロセスは wait で子プロセスの終了を待つという操作を行った．

ptrace システムコールでは，自身のプロセスをトレース状態にする呼出し（TRACEME）と，停止した子プロセスを再開する呼出し（SYSCALL）があり，それぞれ処理にかかる時間が違う．多重化したリファレンスモニタを適用した場合，TRACEME オプションの処理にかなりの時間を要している．しかし，実際の使用環境ではプロセスの開始時に一回だけ呼び出されるものであるので，それほど問題にはならないと考える．一方，SYSCALL オプションによる実行の再開では，リファレンスモニタを単体で適用した場合と比べるとかなり遅くなっている．これは多重化の段数を n としたとき，2^n のコンテキストスイッチが余分に発生してしまうからである．

6 関連研究

RECURSIVE SANDBOXES[13] は，監視されているプロセスと監視しているデーモンとが通信を行い，監視されているプロセスの子プロセスに対するセキュリティポリシーを動的に切替えることが出来る．つまり，子プロセスに対して別のセキュリティポリシーを持ったサンドボックスを適用することが出来る．しかし，これはサンドボックスにしか適用できないという点で本研究とは異なる．そのため，サンドボックスと IDS を組み合わせて使うという目的には使えない．本研究ではサンドボックスではないシステムの重ね合わせも可能である．なお，セキュリティポリシーの動的切替え機能を実装したものとして阿部らのシステム [14] がある．仮想マシンや仮想 OS などとはオーバヘッドの大きさで差がつくと予想される．

7 おわりに

本稿では，プロセス監視用のシステムコールを利用したリファレンスモニタを多重化する手法を提案した．我々の手法では，プロセス監視の動作をエミュレーションすることで実現した．これにより，監視側と監視される側の関係を n 対 m にすることが可能になった．エミュレーションはトレースマネージャにより行われる．また，トレースマネージャを組み込まなければならないリファレンスモニタは 1 つだけであり，このリファレンスモニタの監視空間内で動くリファレンスモニタは既存の実装のままで動作する．リファレンスモニタを適用した際のオーバヘッドは，リファレンスモニタを利用する上では避けることが出来ない．複数のリファレンスモニタを多重化するにはそれなりのコストが必要である．しかし，複数のリファレンスモニタを同時に使用可能になることにはそれ相応のメリットがある．

今後の課題として，今回用いているプロセス監視用システムコールのエミュレーションによる実装では，監視空間内のリファレンスモニタには変更を加

える必要が無い一方で，エミュレーションを行うためのコストが掛かってしまう．今後はコストを削減するための手法を検討していく予定である．

参考文献

[1] Levy, E.: Approaching zero [attack trends]. Security & Privacy Magazine, IEEE, Vol.2, Issue.2, pp. 65–66, (2004).

[2] Tripwire
http://www.tripwire.org/
http://www.tripwire.co.jp/

[3] Dragon Squire
http://www.enterasys.com/jp/products/ids/

[4] Wagner, D., Dean, D.: Intrusion Detection via Static Analysis. In: Proceedings of the 2001 IEEE Symposium on Security and Privacy, pp. 156–168, (2001).

[5] Giffin, J.T., Jha, S. and Miller, B.P.: Detecting manipulated remote call streams. In Proceedings of the 11th USENIX Security Symposium, (2002).

[6] 阿部洋丈，大山恵弘，岡瑞起，加藤和彦：静的解析に基づく侵入検知システムの最適化，情報処理学会論文誌 第 45 巻 SIG 3 (ACS 5), pp. 11–20, (2004).

[7] Jails: Confining the omnipotent root.
http://docs.freebsd.org/44doc/papers/jail/jail.html

[8] LIDS
http://www.lids.org/

[9] Solaris Zone
http://jp.sun.com/products/software/solaris/10/containersLowRes.html

[10] 大山恵弘，神田勝規，加藤和彦：安全なソフトウェア実行システム SoftwarePot の設計と実装，コンピュータソフトウェア，日本ソフトウェア科学会，Vol. 16, No. 6, pp. 2–12, (2002).

[11] D. Wagner and P. Soto.: Mimicry Attacks on Host- Based Intrusion Detection Systems. In Proceedings of the 9th ACM Conference on Computer and Communications Security, pp. 255–264, (2002).

[12] DTrace BigAdmin Forum.
http://www.sun.com/bigadmin/content/dtrace/

[13] Aleksey Kurchuk, Angelos D. Keromytis：RECURSIVE SANDBOXES: EXTENDING SYSTRACE TO EMPOWER APPLICATIONS. 19th IFIP International Information Security Conference (2004)

[14] 阿部洋丈，加藤和彦，王維：セキュリティポリシーの動的切替機構を持つリファレンスモニタシステム，情報処理学会コンピュータシステム・シンポジウム論文集, pp. 61–68, (2002).