RL-001

ライブラリ関数毎のシステムコール監視による侵入検知システムの開発

鶴田浩史† 槙本裕司†
齋藤彰一† 松尾啓志†

プログラムの脆弱性を悪用した攻撃による被害が問題となっている．このような攻撃は，パターンマッチングによる検知が用いられてきた．しかし，この検知手法は，未知の攻撃を防ぐことができない．最近では，未知の攻撃を防ぐ異常検知を用いた侵入検知システムが研究されている．本論文では，ライブラリ関数の呼び出しアドレス，実行順序とシステムコールの情報を用いた侵入検知システムを提案する．提案システムでは，システムコールの呼び出し時にライブラリ関数の実行アドレス，実行順序，及び，システムコールの情報を確認することで，異常を検知する．実験では，検知にかかるオーバヘッドを測定した．

Implementation of Intrusion Detection System Based on Observing System-Call classified by each Library Function

Koji Tsuruta,† Yuji Makimoto ,† Shoichi Saito†
and Hiroshi Matsuo†

In recently, an intrusion detection system(IDS) with anomaly detection to prevent unknown attacks is researched. A pattern matching has been used by current IDS. But the pattern matching can't detect unknown attacks. In this paper, we propose the new IDS to detect unknown attacks. The proposed IDS classifies a system call by each library function, and observes system calls. A system call executed from each library function is limited to classified system call. Even if any worm can make an intrusion into the system, the worm couldn't execute an arbitrary system call. In the experiment, an overhead time of the proposed IDS is described.

1. はじめに

インターネットの普及と共に，システムに被害を与える不正アクセスが多く発生している．この原因の１つに，プログラムの脆弱性となるセキュリティホールが数多く発見され，．不正アクセスを行うワームなどがそれらのセキュリティホールを悪用しているためである．特に，バッファオーバーフロー攻撃の侵入を目的とした攻撃は，深刻な被害を与えることが多い．

セキュリティホールを悪用した攻撃による異常を検知するためには侵入検知システムが有効である．一般的には不正検知とよばれる検知手法が用いられている．この手法は，攻撃の特徴を記述したパターンの情報を記したシグネチャを攻撃の特徴とマッチングさせることで攻撃を検知する．不正検知では攻撃パターンに存在する既知の攻撃に対しては，高い検出率を示すが，未知のセキュリティホールに対する攻撃やシグネチャに存在しない攻撃は検知できないという問題がある．

最近では，正常な動作時と異なる状態を異常と判断する異常検知による検知手法が用いられている．この手法は，正常な動作時の状態をパターンとして持ち，正常な動作時と異なる状態を検知したとき異常と判断する．異常検知による検知手法は，未知のセキュリティホールに対する攻撃でも検知することができるため，現在，研究[1)2)3)4)]が盛んに行われている．

本稿では，バッファオーバフロー攻撃を検知の対象として，異常検知による検知手法について述べる．特に，ライブラリ関数の呼び出しアドレス，ライブラリ関数の実行順序とシステムコールを用いた異常検知による侵入検知システムを提案する．提案するシステムは監視の対象とする実行ファイルを静的解析することにより得られた情報を基に，実際に動作した結果と比較することで正常か異常かを判定する．具体的には，1) システムコールを呼び出すライブラリ関数の呼び出し元であるユーザ関数の実行アドレスと 2) ライブ

† 名古屋工業大学
Nagoya Institute of Technology

ラリ関数の実行順序と，3) 各ライブラリ関数が呼び出す可能性のあるシステムコールのリストの 3 種類を正常な動作時の情報として用いる．これらを実際の動作中に検査比較することで，異常な動作を判定する．プログラムの動作を判定する情報の 1 つであるライブラリ関数が呼び出す可能性のあるシステムコールの情報は，実行中に変化することはない．このため，ライブラリ関数の内部まで検査する必要がなく，検査に要する時間が短くて済むのが本システムの特徴である．

2 章では既存の侵入検知システムについて述べる．3 章では提案手法について述べる．4 章では実験結果を示し，実験結果に対する考察について述べる．5 章ではまとめについて述べる．

2. 既存の侵入検知システム

静的解析を用いた侵入検知システムでは，ソースコードやバイナリコードを解析することにより，正常な動作と異常な動作を判断するパターン（以降，規則と呼ぶ）を作成する．この手法では，ソースコードやバイナリコードに基づくすべてのパターンを規則化することができる．そのため，すべてのパターンの情報を持った規則と比較するだけで正常と異常を判断できる．したがって，静的解析で作成された規則では false positive が発生しないという特徴がある．

静的解析を用いた侵入検知システムは，今日までにさまざまな手法が提案されている．Wagner らが提案した手法[1] は，システムコールの呼び出し順序のみを確認する手法である．この手法は，システムコールの情報だけで，プログラムの制御を推定しているので，状態数が非常に多くなる．そのため，実行時の検知にかかるオーバヘッドが極めて大きくなるという問題がある．

Feng らが提案した手法[2] は，システムコールが呼び出されたときのスタックの情報を用いる手法である．この手法は，システムコールが呼び出されたとき，スタックに積まれている情報を取得し，Virtual Stack List と呼ばれるリターンアドレスのみが積まれたスタックを作成する．プログラムの実行時に，現在の Virtual Stack List と 1 つ前の Virtual Stack List をスタック最下位から比較を行い，Virtual Path と呼ばれる関数が遷移する際のリターンアドレスの列を作成している．

阿部らの手法[3] は，Wagner らの手法であるシステムコールの呼び出し順序を確認する手法を改良した手法である．Wagner らの方法は，システムコールの情報のみを用いてプログラムの制御を非決定的に把握する．そのため，オーバヘッドの増加と検知精度を低下させることになる．そこで，彼らの方法は，実行時のスタックの情報を用いることで制御を決定的に把握する．具体的には，前回のシステムコールが呼ばれてから今回のシステムコールが呼ばれるまでの関数遷移を確認し，関数遷移が確認できない場合は異常と判定する方法である．また，このままでは，オーバヘッドが大きすぎるので，オーバヘッドを削減するアルゴリズムを実装している．このアルゴリズムは，関数遷移のパスを探索履歴として保存することで以前に行った探索を省略することが可能になるものである．このアルゴリズムにより，オーバヘッドを 2 倍程度に抑えることができたと述べている．

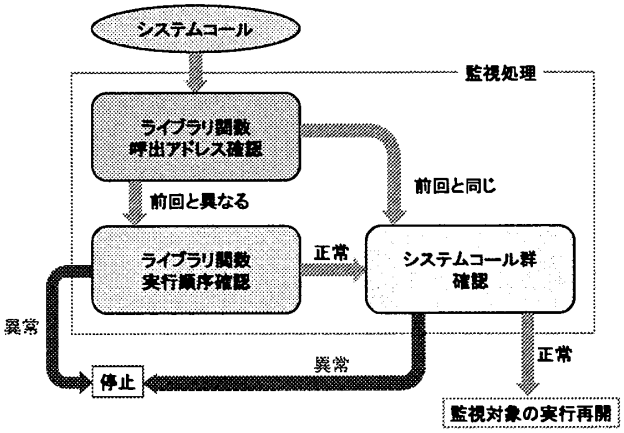

図 1　監視処理の流れ

槙本らの手法[4] は，学習と静的解析を組み合わせた侵入検知システムである．この方法は，プログラムの実行の学習により作成したシステムコールと呼び出し位置を組にした情報をもとにシステムコールの呼び出し順序の規則と，実行ファイルの静的解析により作成された規則である，ユーザ関数実行規則，スタックリスト確認規則を用いる．具体的には，学習済みの動作は，学習により作成されたシステムコールの呼び出し順序の規則を確認するため，静的解析よりもオーバヘッドが小さくてすむ．また，学習漏れに対しての処理は，静的解析により作成された規則を確認することで異常を検知できる．

3. 提案手法

本稿では，静的解析で作成した規則に基づいて動的な解析を行う侵入検知システムを提案する．静的解析により作成された規則は，監視の対象となる実行ファイルのバイナリコードをあらかじめ解析して作成するため，実行ファイルの正常な動作のパターンをすべて網羅できる．そのため，学習に基づく侵入検知システムのような学習漏れなどによる false positive が発生しない．

提案手法の構成を図 1 に示す．プログラムの実行時に呼び出されたシステムコールを基に監視処理を行う．監視処理は，呼び出されたシステムコールを静的解析であらかじめ作成した規則と比較することで正常な動作であるかを判定する処理である．監視処理の詳細は，

4 章で述べる．

監視システムで検知を行う監視処理では，ライブラリ関数の呼び出しアドレスの規則，ライブラリ関数の実行順序の規則，システムコール群の規則の 3 つの規則を用いる．監視処理で用いるシステムコール群の規則が本システムの特徴である．これらの規則はそれぞれ 4 章で述べるライブラリ関数呼び出しアドレス確認処理，ライブラリ関数実行順序確認処理，システムコール確認処理に用いる．以下，監視処理に用いる規則の特徴について述べる．

3.1 ライブラリ関数の呼び出しアドレスの規則

本規則は，ユーザ関数の実行アドレスで，システムコールを含むライブラリ関数の呼び出し元アドレスのリストである．これによって，未知のアドレスからの関数呼び出しを異常であると検知できる．

3.2 ライブラリ関数の実行順序の規則

本規則は，ユーザ関数内でライブラリ関数を呼び出す順序を表している．本システムでは，各システムコール呼び出し毎に，システムコールを呼び出したライブラリ関数の実行順序を確認する．これによって，ユーザ関数内での実行の正確性を保証する．このため，1 つ前に確認したユーザ関数内の実行アドレスから今回のユーザ関数内への実行アドレスへの処理が遷移する情報を持つ．ライブラリ関数の実行順序を確認することで，未知の関数への遷移を異常であると検知することができる．

3.3 システムコール群の規則

本規則は，各ライブラリ関数が呼び出す可能性のあるシステムコールと，ライブラリ関数内で当該システムコールの呼び出し元アドレスを取りまとめたものである．本規則は，各ライブラリ関数毎に定義する．本規則の特徴として，呼び出す可能性のあるシステムコールの種類と呼び出しアドレスを扱い，その実行順序を規則化しないことがあげられる．これにより，実行時の動作確認処理をシステムコールの比較のみに限定することができ，状態遷移の確認を要しないことから確認処理の高速化が可能となる．しかし，実行順序の情報を持たないことから，異常な順序でシステムコールが呼び出された場合に検知することができない．以下，この規則の正当化について述べる．

本規則が有しない情報は，各ライブラリ関数から呼び出されるシステムコールのライブラリ関数内での順序である．ライブラリ関数そのものの順序は，先に述べたライブラリ関数の実行順序の規則によって規定されている．したがって，本規則を適用した環境で異常な動作を引き起こすには，ライブラリ内の欠陥を利用しての攻撃が必要である．また，どのような種類のシステムコールを呼び出すかは規則化されていることから，未知のシステムコールの呼び出しは検知できる．つまり，ライブラリ関数への攻撃によって可能となる本規則で検知できない処理は，各ライブラリ関数に予めプログラムされたシステムコールのみを利用した処理である．

まず，ライブラリが信頼できる場合は，問題はない．そこで，ライブラリに欠陥があり完全に信用できない場合を想定する．ライブラリ関数には，システムコールと一対一に対応しているものが多数ある．これらのライブラリ関数を攻撃したとしても，利用できるシステムコールが限定されるため，攻撃者にとって得るものがない．また，これらの関数は構造が単純であるため，攻撃そのものが困難である．したがって，fopen() など，ライブラリ関数内で複数のシステムコールを呼び出し，比較的複雑な処理を実行するライブラリ関数が攻撃の対象となる．ここで，攻撃を検知するために，システムコールの呼び出し元アドレスを用いる．呼び出し元アドレスは，スタックに積まれている．本システムでは，ライブラリ関数の呼び出しアドレス確認処理と同時に，システムコール呼び出し元アドレスの確認を行う．これによって，攻撃者がライブラリが配置されたメモリ空間以外からシステムコールを呼び出したことを検知する．ただし，本手法は，スタックが正常であるという前提がある．スタックが置き換えられ，呼び出し元アドレスが偽装された場合の検知手法は今後の課題である．

4. 実　　装

提案システムでは，異常な動作を判定する処理として，図 1 に示すようにライブラリ関数呼び出しアドレス確認処理，ライブラリ関数実行順序確認処理，システムコール群確認処理の 3 つの処理でそれぞれ規則と比較し，監視対象となるプログラムが正常であることを確認する．これらの 3 つの処理で異常であると判断された場合，監視対象となるプログラムの実行を終了させることで異常な動作を実行できないようにする．これらの処理について順次述べる．

4.1 ライブラリ関数呼び出しアドレス確認処理

ライブラリ関数の呼び出しアドレスの確認処理では，システムコールを呼び出しているライブラリ関数を実行するユーザ関数の実行アドレスとライブラリ関数の呼出アドレスの規則を比較し確認する処理である．

まず，システムコールが呼び出されると，CPU は特権モードに移行し，使用するスタックをユーザスタックからカーネルスタックに切り替える．スタックには，システムコールの引数，システムコール番号，システムコール呼び出し前のレジスタの値が積まれる．このとき，ユーザスタックに積まれているベースポインタの値を参照して，ユーザ関数への戻りアドレスを取得し，ユーザ関数への戻りアドレスをもとにライブラリ関数の呼び出しアドレスの確認を行う．

また，システムコールが呼び出されたとき，1 つ前に呼ばれたシステムコールと同じライブラリ関数から

表1 実験結果

対象プログラム		実行時間 [秒]	規則の大きさ [KB]	増加の割合
wc	監視システムなし	1.681	-	1.00
	監視システムあり	1.706	15	1.02
gzip	監視システムなし	1.328	-	1.00
	監視システムあり	1.346	25	1.02

呼ばれているかを比較する．もし，同じライブラリ関数の場合は，システムコールの確認を行う処理に遷移し，違うライブラリ関数の場合は，ライブラリ関数の実行順序の確認を行う処理に制御を移す．

4.2 ライブラリ関数実行順序確認処理

ライブラリ関数の実行順序の確認処理は，今回確認するライブラリ関数の呼び出しアドレスと1つ前に確認したライブラリ関数の呼び出しアドレスの順序関係を確認する処理である．この処理では，今回システムコールを呼び出したライブラリ関数の呼び出しアドレスが，前回システムコールを呼び出したライブラリ関数から遷移できることを規則で確認する．なお，この処理は，今回システムコールを呼び出したライブラリ関数の呼び出しアドレスと前回システムコールを呼び出したライブラリ関数の呼び出しアドレスが異なる場合のみ実行する．

4.3 システムコール群確認処理

システムコール群の確認処理は，ライブラリ関数呼び出しアドレス確認処理で確認したライブラリ関数の呼び出しアドレスから呼ばれるライブラリ関数が呼び出すべきシステムコールであることを確認する処理である．システムコール群を確認することで，不正なシステムコールの呼び出しを検知することができる．

5. 評　　価

本章では，4章の内容をLinux上でカーネルを改良して実装した．実装した異常検知システムを用いた結果および，それに対する考察を述べる．

5.1 評価環境

実験環境は，Pentium 4（3.4GHz）上で動作しているFedora Core 5であり，カーネル2.6.17.8である．実験に用いられる異常検知システムはC言語で記述した．異常検知システムは，監視対象のプログラムの実行を監視する．監視の対象となるプログラムはGNUのwcとgzipを使用した．これを用いて，それぞれ約15MBのテキストファイルにおいて，動作の実行時間を計測した．

5.2 結　　果

実験は，監視システムを使用しない場合と監視システムを使用した場合の実行時間を計測した．その結果を，表1に示す．増加の割合は，まったく監視しない場合を1として正規化している．システムコール1回あたりの処理時間は約26μsである．システムコールの呼び出し回数はwcでは約980回，gzipでは約720回であった．そのため，処理全体としてそれぞれ約25ms，約18ms増加した．これは，規則のサイズが15KBや25KBに抑えることができたため，比較にかかる時間が小さくなったと考えられる．

また，ライブラリ関数から呼び出されるシステムコールを，ライブラリ関数から呼び出されないシステムコールに変更して実行した．この結果，ライブラリ関数の呼出アドレスの規則と実行順序の規則は正常と判断されたが，システムコール群の規則に呼び出されたシステムコールが存在しないため異常と通知され，それ以降の実行を停止させることができ異常を検知することを確認した．

6. ま と め

システムコールが呼び出されたタイミングでプログラムの制御を監視するシステムを提案した．システムコールが呼び出された時の情報を取得し，この情報からライブラリ関数の呼び出しアドレスを確認することで，関数の実行位置を特定し，プログラムの流れを把握することができる手法である．本システムの実験より，ユーザ関数が呼び出すライブラリ関数とシステムコール群のみを検査することにより，検知に要する時間が微増であることを確認した．今後の課題として，より規模の大きなプログラムに適用できることを確認する予定である．

参 考 文 献

1) Wagner, D. and Soto, P.: Mimicry Attacks on Host-Based Intrusion Detection System, *ACM Conference on Computer and Communications Security* (2002).
2) Feng, H.H., M.Lolesnikov, O., Fogla, P., Lee, W. and Gong, W.: Anomaly Detection Using Call Stack Information, *IEEE Security and Privacy 2003* (2003).
3) 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，　Vol.45, No.SIG 3(ACS 5) (2004).
4) 槙本裕司，齋藤彰一，松尾啓志：システムコールの実行順と実行位置に基づく侵入検知システムの実現，情報処理学会研究会報告システムウェアとオペレーティング・システム，　No.2007-OS-106 (2007).