Windowsにおけるリファレンスモニタの実現

神田 勝規[1]　大山 恵弘[2]　光山 義紀[3]　加藤 和彦[4,5]

概要

インターネットに代表されるオープンなネットワーク環境で流通するソフトウェアを安全に実行するためのシステム SoftwarePot の研究開発を進めている．SoftwarePot はソフトウェアを仮想的な環境（サンドボックス) で実行するシステムである．SoftwarePot はソフトウェアが発行するシステムコールを捕捉し、検査及び書き換えを行うことによってサンドボックスを構築している．システムコールの捕捉を実現するには、OS に依存した機能を使用しなければならず，移植性が低くなる要因となっている．本論文では，SoftwarePot の新ソフトウェアアーキテクチャについて述べる．SoftwarePot からリファレンスモニタシステムとして分離を行い，移植性の向上を図っている．Windows への SoftwarePot の移植を通して，新アーキテクチャの有効性の検証を行った．

The implementation scheme of a reference monitor on Windows

Katsunori Kanda[1] Yoshihiro Oyama[2] Kohyama Yoshinori[3] Kazuhiko Kato[4,5]

Abstract

We are developing the SoftwarePot system to safely execute the softwares circulated in the open network. The SoftwarePot provides the virtual execution environment for the circulated software. The virtual environment is implemented by the system call interception. In this paper we propose a design to supporte multiplatform and an implementation scheme on the Windows.

1　はじめに

HTTP、FTP、SMTP 等を利用し、多くのソフトウェアが流通するようになった．しかし、電子メールに添付されるウイルスのように、ソフトウェアの中には利用者の意図とは異なる動作をするものがある．こういったソフトウェアの中には、ファイルを不正に改竄したり、機密データを漏洩させるものがある． 悪意を持った作者によって作られたこのようなソフトウェアから計算機資源を守るために、資源へのアクセスが制限された実行環境を作る研究がこれまでに行なわれてきた．我々は、ソフトウェアを流通させる用途に適したサンドボックスシステム SoftwarePot を提案している [4, 5, 6, 7]. SoftwarePot はソフトウェアを仮想的な実行環境に閉じ込めて実行することができる．ソフトウェアは明示的な指示が与えられない限り、仮想的なファイルシステム外のファイルにアクセ

[1]筑波大学大学院博士課程システム情報工学研究科
University of Tsukuba Graduate School, Doctoral Program System & Information Engineering
[2]東京大学大学院情報理工学研究科
University of Tokyo Graduate School of Information Science and Technology
[3]有限会社バクサリー
BUCSALY Inc.
[4]筑波大学 電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba
[5]科学技術振興事業団 さきがけ研究 21
Japan Science and Technology Corporation, Presto

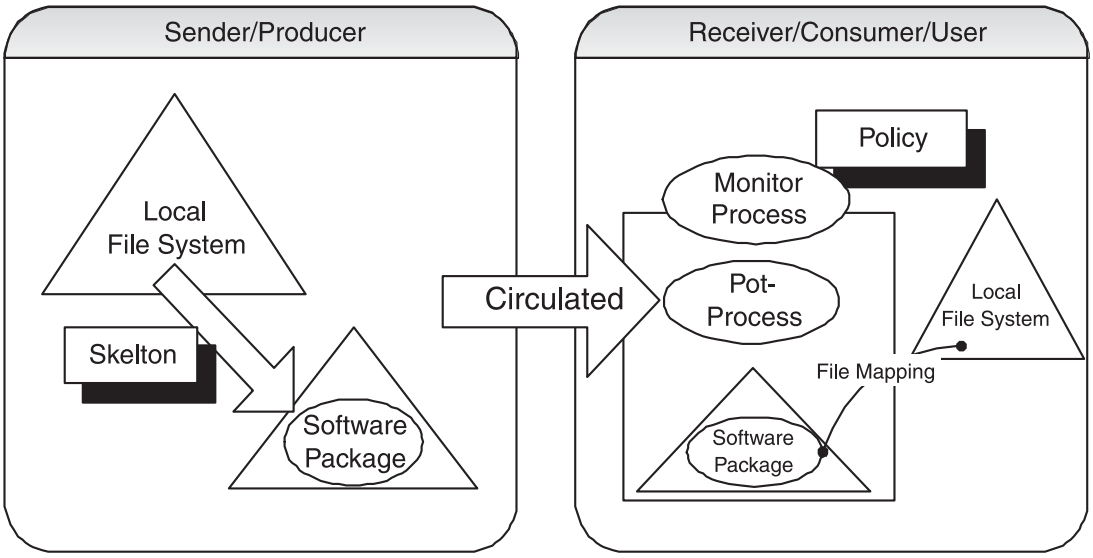

図 1: SoftwarePot の利用の様子

スすることはできない. ソフトウェアパッケージ開発者が作成したパッケージは仮想的なファイルシステムの構築に必要な情報を含むアーカイブファイルとして配布される. パッケージを入手したユーザーは、ファイルへのアクセスや通信といった計算機資源へのアクセスを制限することができる.

文献 [4, 5, 6, 7] で提案されている SoftwarePot は計算機資源へのアクセス制御、仮想的なファイルシステムの構築をシステムコール引数の検査と書き換えによって実現している. これまで、SoftwarePot は Linux と Solaris において実装されてきた. システムコールの捕捉機構 (リファレンスモニタ) は、OS の機能に依存した実装となっており移植性が低い. 特に Windows 上でリファレンスモニタを作成するには、OS の基本機能の違いが問題となる. また、Windows の Win32 API は UNIX のシステムコールのように OS カーネルのサービスを利用するための唯一のインターフェースではなく、ライブラリコールであるため、Win32 API を捕捉する機構を作成した場合、この捕捉機構を回避する余地が残ってしまう. そのため、Windows 上でのリファレンスモニタは Cygwin によって提供される API のみを用いて構築されたアプリケーションを対象とする. この結果、Windows におけるリファレンスモニタによって安全性を確保することは困難である. しかし、ソフトウェアのインストレーションサポートや隔離実行環境の提供といった有用な利用方法が存在すると考える.

本論文は以下のように構成される. まず、2 章において SoftwarePot が提供する機能と現在の実装方法の概要を述べる.3 章においてリファレンスモニタの設計を述べる. 4 章で、リファレンスモニタの Windows 上の実現方法を述べる. 5 章で関連研究を述べる. 6 章でまとめと今後の課題を述べる.

2 SoftwarePot システムの概要

SoftwarePot 上で実行されるソフトウェアは、仮想的な実行環境 (**ポット空間**) の構築に必要なメタデータを含むアーカイブファイル (**ポットファイル**) として配布される. ポット空間とは、仮想的なファイルシステムを持ちネットワーク資源へのアクセスが制限された実行環境である. ポットファイルを入手したユーザーはポット空間の中で、ソフトウェアを実行する. この時、ユーザーはソフトウェアがどのように実行されるべきかをセキュリティポリシーファイルとして与えることができる. セキュリティポリシーファイルには、ソフトウェアが使用でき

るシステムコールやアクセス可能なネットワーク、ファイルマッピングのポリシーが記述される. ファイルマッピングとは、ローカルのファイルシステムの一部をポット空間内の仮想的なファイルシステムから参照できるようにする操作である. ファイルマッピングが行なわれる典型的なファイルは共有ライブラリやソフトウェアの実行結果を保存するためのディレクトリである.(図 1)

ポット空間はソフトウェアの発行するシステムコールを捕捉し、その引数を検査及び書き換えることによって実現されている. システムコール捕捉機構の実装方法は、対象とするオペレーティングシステムに強く依存する. そのため、SoftwarePot では移植性を考慮し、システムコール捕捉機構を SoftwarePot から独立したシステム「refmon」として実装している. SoftwarePot は、refmon の提供する API を通してポットプロセスの監視を行なう. 従って、システムコール捕捉機構の実装方式の違いによって、SoftwarePot の実装を変更する必要がない.

3 リファレンスモニタの構造

アプリケーションの動作中に発生するイベントを捕捉し任意の処理を行なうシステムをリファレンスモニタシステムと呼ぶ. リファレンスモニタシステムにおいて、アプリケーションを監視する機構をモニタと呼ぶ. リファレンスモニタシステムによって捕捉される典型的なイベントは、システムコール呼び出しであり、refmon はリファレンスモニタシステムの一種である. 捕捉できるイベントによってさまざまな種類のリファレンスモニタシステムが考えられるが、それらは共通して以下の機能を持つ.

- イベント捕捉機構
- 監視強制機構
- 実行制御機構 (継続・中止・強制終了)

イベント捕捉機構とは、アプリケーションが実行中に発生するイベントを捕捉しそのイベントに関する情報をモニタへ通知する機構である. 監視強制機構とは、アプリケーションがモニタの監視から逃れられないようにする機構である. アプリケーションが実行中に子プロセスを作成した場合、子プロセスもモニタの監視下に置かれ、アプリケーションの挙動がすべて監視されることが保証できる. 実行制御機構とは、イベント捕捉後アプリケーションの実行を継続するか中止するか、強制終了するかを制御する機構である.

refmon が対象とするイベントは、システムコール呼び出しである. refmon は、上記の 3 つの機構に加え、システムコール引数の書き換えを支援する機構を持つ. これは、SoftwarePot の仮想的なファイルシステムを実現するためである.

モニタをユーザープロセスとして実装するのか、カーネルモジュールやカーネルパッチといった形でカーネルの一部として実装するのかでリファレンスモニタを分類することができる. カーネル内に実装されたモニタは一般に、ユーザープロセスとして実装されたモニタと比べて高速に動作する. しかし、カーネルレベルでの開発はユーザーレベルのライブラリを利用することができないことや、デバッグが困難であるため開発コストが高い. 機能拡張を容易にするため、refmon はユーザーレベル実装となっている[1].

3.1 基本動作

refmon が動作する様子を open システムコールを例に説明する (図 2). (1) アプリケーションプロセスが open システムコールを発行し、refmon によって捕捉される. (2)refmon はモニタに open システムコールが発行されたことを通知する. (3) モニタプロセスは refmon から与えられた情報をもとに何らかの処理を行ない実行の継続許可を出す. (4) モニタプロセスからの通知を受け open システムコールの処理を継続する. (5)OS kernel が open システムコールの処理を終えると再び refmon によって捕捉

[1] Linux 版の refmon は、現在、カーネルモジュールを利用してシステムコール捕捉を実現している. しかし、今後 ptrace() を用いた実装を行なう予定である.

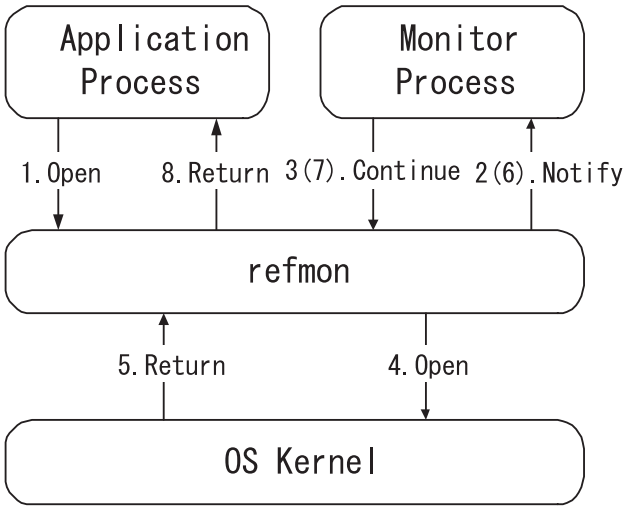

図 2: refmon の動作する様子

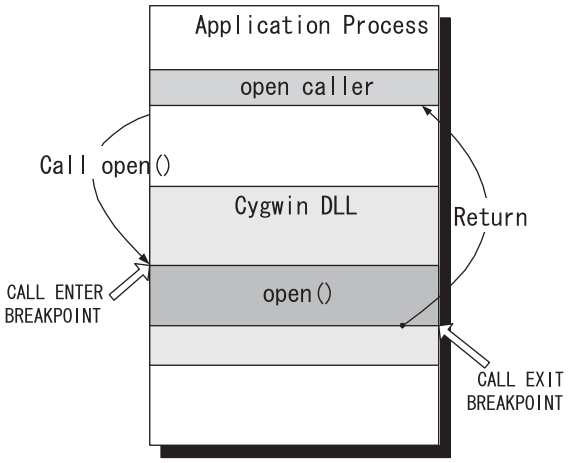

図 3: Windows においてのアプリケーションプロセスのアドレス空間

される．(6) モニタプロセスに open システムコールが終了したことを通知する．(7) モニタプロセスは何らかの処理を行ない open システムコールの実行の継続許可を出す．(8)refmon はモニタプロセスからの通知を受けアプリケーションプロセスへ open システムコールの処理結果を返す．

図 2 の例では、open システムコールの実行が許可されているが、例えば、セキュリティポリシーで書き込みが許可されてないファイルに対して書き込みモードでファイルを開こうとしている場合など、実行の中止を行なうことができる．中止命令が発行された場合、OS カーネルへ処理が移ることなくアプリケーションプロセスへエラーコードが返されるため、アプリケーションからは OS カーネルへ処理が移ったのかどうかを判断することはできない．

4 Windows におけるリファレンスモニタの実現

Windows において Win32 API コールを捕捉してサンドボックスを構築することは困難である．何故なら、Win32 API コールを捕捉することは UNIX においてライブラリコールを捕捉することと同等であり、モニタ監視を逃れて資源にアクセスすることが可能だからである．このような問題から、対象とするアプリケーションが利用できるライブラリを Cygwin [1] に制限することにした．Cygwin は、POSIX 関数を Windows 上に提供するライブラリである．Cygwin をサポートするライブラリとして選んだ理由は、これまで UNIX 環境で開発されてきた SoftwarePot を Windows 上へ移植する際に作業が容易になると考えたからである．

同プロトタイプでは、未知の攻撃から計算機資源を保護することはできないが、ソフトウェアの隔離実行や、インストレーションをサポートするツールといった有用な利用方法は存在すると考えられる．

4.1 基本機能の実現

リファレンスモニタの基本機能である、イベント捕捉機構、監視強制機構、実行制御機構の実現方法について述べる．

はじめに、イベント捕捉機構の実現方法を説明する．Windows 版 refmon で捕捉するイベントは Cygwin DLL によって提供される POSIX 関数呼び出しである．POSIX 関数の捕捉には、Windows によって提供されるブレイクポイント機構を利用している．関数の捕捉には呼び出し直前 (CALL ENTER) の捕捉と呼び出し終了 (CALL EXIT) の捕捉の二種類が存在する．呼び出し直前を捕捉するためには、関数の

先頭に位置するインストラクションをブレイクポイントに置き変えればよい．また、呼び出し終了を捕捉するためには ret インストラクションをブレイクポイントに置き変えればよい (図 3)．ブレイクポイントを設置するためには、ブレイクポイントを設置したい関数の先頭アドレスをアプリケーションプロセスのアドレス空間内から見つけ出さなければならない．これは、DLL の先頭アドレスから関数の先頭までの相対アドレスはいつも同じになることを利用する．モニタプロセス上に Cygwin DLL をロードし、Cygwin DLL がロードされた先頭アドレスと関数の先頭アドレスの相対アドレスを求める．Cygwin DLL の先頭アドレスを取得するには、ToolHelp API を用いる．この API を用いると関数の先頭アドレスも取得することができる．しかし、ret インストラクションのアドレスを取得する API は提供されていない．現在の実装では、関数の先頭アドレスから逆アセンブルを行ない、ret インストラクションのアドレスを計算している．

次に、監視強制機構の実現方法を説明する．Windows においてブレイクポイント機構を利用するためには、アプリケーションプロセスがデバッグモードで起動されていなければならない．デバッグモードで起動されたプロセスに対して、モニタプロセスはデバッグイベントの監視を行なうことができる．デバッグイベントには、ブレイクポイントでの停止イベント、シングルステップ実行イベント、プロセスの生成イベント等がある．アプリケーションが子プロセスの作成を行なった場合、モニタプロセスは自身をデバッガプロセスとして子プロセスへアタッチ処理を行なう．この処理によって、モニタプロセスは子プロセスのデバッグイベントも監視することができるようになる．

次に、実行制御機構の実現方法を説明する．実行制御機構は Win32 API のデバッグ機構の一部として提供されているため、その機構を利用している．アプリケーションプロセスはデバッグイベントを発行すると、その実行を停止しモニタプロセスからの指示待ち状態になる．モニタプロセスは、Win32 API を通してアプリケーションプロセスの実行を制御することが可能である．

5 関連研究

文献 [3] のシステムは、UNIX 環境でユーザレベルのリファレンスモニタシステムの構築を支援する．このシステムの特徴は、システムコールが機能ごとにいくつかのグループに分類されており、リファレンスモニタからグループに対する操作を行うことができる点である．この機構によって、同等の機能でありながらシステムコール識別子が OS によって異なるようなシステムコールを扱いやすくなる．ポータビリティを考慮した場合も非常に有用な機構であるため、今後我々のシステムにも取り入れたいと考えている．

SysGuard [8] はシステムコールを捕捉し、Guard と呼ばれるソフトウェアモジュールによってアクセス制御を行なう．Guard はユーザーによって自由に開発を行なうことができるため柔軟性のあるアクセス制御を行なうことができる．しかし、Guard はカーネル空間内で実行されるソフトウェアモジュールであるため、ポータビリティの確保が問題となる．文献 [9] において OS 固有の構造を抽象化しポータビリティを高めようと試みていると述べられている．我々のシステムは、Guard と同様にソフトウェアモジュールによってアクセス制御を行なうことができるため、柔軟性が高い．また、ユーザーレベルでソフトウェアモジュールを作成することが可能なため、SysGuard のような OS 固有の構造による問題は起らない．しかし、Linux の socketcall() システムコールのようなシステムコールインターフェイスの OS ごとの差異による問題は SysGuard と同様に生じる．

Entropia [2] は、インターネット上の遊休計算機の余剰計算能力を利用して分散計算を行なう Windows 用のシステムである．計算に参加する計算機にはネイティブコードが配布されて実行される．Entropia は配布されたネイティブコードから計算機上の資源を保護するためのサンドボックスを備えている．Entropia のサンドボックスはバイナリ変換によってファイルへのアクセスといった主要な Win32 API を捕捉し

実現されている.

6 まとめ

SoftwarePot システムの移植性を高めるために、リファレンスモニタシステム refmon の設計と実装を行なった. 現在、Windows 版 refmon のプロトタイプの作成は終了し UNIX 版 refmon への統合を進めている. 今後の課題として、システムの性能評価が第一に挙げられる. 特に、Windows 版 refmon によって生じるオーバーヘッドの計測を行ないたい. 次に取り組みたい課題として、文献 [3] で用いられているようなシステムコールをグループごとにまとめる手法を我々のシステムにも取り入れたい. この機構を取り入れることによって、Windows 版 refmon を Win 32 API に対応させた場合にも refmon を用いて構築されたリファレンスモニタに高い移植性を提供できると思われる. 現在我々のシステムは Cygwin DLL のみを対象としているが、将来的には Entropia と同様に Win32 API に対応したい. Win32 API は、kernel32.dll といくつかの DLL によって提供されている. これら Win32 API を提供する DLL を提案システムが Cygwin DLL に対して用いた手法を適用することによって Win32 API に対応したリファレンスモニタは作成可能である.

参考文献

[1] http://cygwin.com.

[2] A. A. Chiem, B. Calder, S. Elbert, and K. Bhatia. Entropia: Architecture and Performance of an Enterprise Desktop Grid System. *Journal of Parallel Distributed Computing*, Vol. 63, No. 5, pp. 597–610, 2003.

[3] K. Jain and R. Sekar. User-level infrastructure for system call interposition: A platform for intrusion detection and confinement. In *Proc. Network and Distributed Systems Security Symposium*, 2000.

[4] K. Kato and Y. Oyama. SoftwarePot: An Encapsulated Transferable File Sysstem for Secure Software Circulation. In *Software Security - Theories and Systems*, Vol. 2609. Springer-Verlag, February 2003.

[5] K.Kato, Y.Oyama, K.Kanda, and K.Matsubara. Software Circulation using Sandboxed File Space—Previous Experience and New Approach. In *8th ECOOP Workshop on Mobile Object Systems*, Malaga, Spain, June 2002.

[6] 神田勝規, 大山恵弘, 加藤和彦. カーネルモジュールを用いた SoftwarePot の効率的な実装方式. 情報処理学会 研究報告システムソトウェアとオペレーティング・システム, 第 90 巻, pp. 81–86, 沖縄, 6 月 2002.

[7] 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. 第 5 回プログラミングおよび応用のシステムに関するワークショップ SPA'02, 3 月 2002.

[8] 榮樂恒太郎, 新城靖, 板野肯三. システム・コールに対するラッパ/リファレンス・モニタ SysGuard の設計と実現. 情報処理学会論文誌, Vol. 43, No. 6, pp. 1690–1701, 2002.

[9] 榮樂恒太郎, 新城靖, 樋爪真紀, 中田吉法, 板野肯三. 高い情報生存能力を実現するラッパ sysguard におけるガード・モジュールの開発環境. 第 4 回プログラミングおよび応用のシステムに関するワークショップ SPA2001, 2001.