社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2007－OS－105 (12)
2007／4／5

実行コンテキストに応じたポリシー指定が可能なサンドボックス

塩谷 知宏† 大山 恵弘‡ 岩崎 英哉‡

†電気通信大学大学院 電気通信学研究科 情報工学専攻
‡電気通信大学 情報工学科
E-mail: shioya@ipl.cs.uec.ac.jp, {oyama, iwasaki}@cs.uec.ac.jp

ソフトウェアに潜在する脆弱性を突いた攻撃から，システムの被害を最小限に留めるために，サンドボックスという仕組みがある．サンドボックスは，プログラムの正常動作に必要な資源へのアクセス権限を，予めポリシーとして与えておくことで，攻撃が成功しても本来プログラムが必要としない資源へのアクセスを禁止できる．しかし，既存の多くのサンドボックスには，プロセスの実行開始から終了まで，同一のポリシーしか適用できないため，実行段階，すなわちコンテキストによっては余分な権限が与えられてしまう．そこで本研究では，プロセスが実行しているユーザ関数を調べて，動的にポリシーを切り替えることが可能なサンドボックスを提案する．実装は Linux カーネルモジュールを用い，システムコールをフックすることでポリシーを適用する．実験により，従来よりも細かいアクセス制限ができ，その有効性を確認した．

A Sandbox with Dynamic Policy based on the Execution Context of the Target Application

Tomohiro Shioya Yoshihiro Oyama Hideya Iwasaki

Department of Computer Science, Graduate School of Electro-Communications,
University of Electro-Communications
E-mail: shioya@ipl.cs.uec.ac.jp, {oyama, iwasaki}@cs.uec.ac.jp

We propose a new sandbox system that dynamically changes its behavior based on the application's execution context. Our system allows users to give different policies, each of which specifies permitted system calls, depending on the user functions in which the target application is now executing. With our system, the target application can be given less privilege than existing, single-policy sandbox systems. We implemented the proposed sandbox by using LKM (Loadable Kernel Module) of the Linux that hooks the system call issued by the application process. We also demonstrate the effectiveness of the proposed sandbox via some experiments.

1 はじめに

悪意のあるユーザが，アプリケーションの脆弱性を突いた攻撃を行い，システムの制御を奪ってしまったり，本来閲覧できないファイルを盗み見たりする不正アクセスが問題となっている．アプリケーションの脆弱性の原因は，アプリケーション作成時に混入したバグなどであり，それらのすべてを実行前に発見し修正することは非常に困難である．そこで，アプリケーションを安全に実行するための仕組みとしてサンドボックスがある．

サンドボックスは，アプリケーションの実行時に，外側からアプリケーションの動作を監視し，ユーザの意図に合わない動作を検出，防止する．通常，アプリケーションに与えるアクセス権の仕様はユーザが記述し，サンドボックスに伝える．以降では、この仕様をポリシーと呼び、ポリシーが記述されているファイルをポリシーファイルと呼ぶ．サンドボックスは，アプリケーションによる計算資源の操作を実行する前に，ポリシーとその操作を比較する。ポリシーで許された操作は実行を許可し，許されていない操作は，実行を拒否する．サンドボックスを利用することによって，アプリケーションにたとえ脆弱性があり不正アクセスをされたとしても，その被害の範囲を限定することができる．

アプリケーションは通常，実行状態に応じてアクセスする資源が異なるが，既存の多くのサンドボックスは，アプリケーションの実行開始から終了まで単一のポリシーを適用する．このことは，意図された実行に最低限必要な権限や資源のみをプログラムやユーザに与えるという最小特権の原則の観点からは，望ましくない．POP サーバを例にとってこれを説明する．

POP サーバは，ソケットの作成，クライアントとの接続の確立，パスワードファイルの読み込み，メールスプールへの読み書き，シグナルの設定など，様々な計算資源に対する操作を行う．その操作は，POP サーバの実行状態に応じて異なる．POP サーバの状態遷移は，図 1 に示すように，クライアントからの接続要求を受け付ける start 状態，ユーザ名を取得し認証の準備をする auth 状態，パスワードを受け取り認証処理をする auth2 状態，メールの閲覧や削除を行う transaction 状態，メールスプールファイルを更新する update 状態，および終了処理を行う terminated 状態に分けられる．ここで，たとえばクライアントと接続を確立する (accept システムコールを行う) のは start 状態だけであって，その他の状態では accept は決して行わない．また，ファイルアクセスについても，start 状態では /etc/hosts に，auth 状態では /etc/passwd に，ユーザが shioya だった時，transaction 状態ではスプール /var/spool/shioya にあるファイルに読み込みモードで，update 状態ではスプールにあるファイルに読み書きモードでアクセス，というように，状態毎にアクセスする対象のファイルやアクセスのモードが異なる (図 2 参照)．しかし，従来のサンドボックスのように，単一のポリシーしか適用できなければ，アプリケーションの実行で必要なすべての資源へのアクセスの許可を，すべての実行状態で共通に与えることしかできないので，それぞれの状態にとっては過大な権限まで与えられることになる．たとえば，図 2 の上半分に示すように transaction 状態においてもパスワードファイル /etc/passwd へアクセスすることができてしまう．

そこで，本研究では，アプリケーションの実行コンテキスト単位で動的にポリシーを変更し，適用できるサンドボックスを実現する．提案するサンドボックスを利用すれば，図 2 の下半分に示すように，POP サーバの状態毎にアクセス可能なファイルを分割することができる．アプリケーションの実行コンテキストは，プログラムの関数単位とした．また，アプリケーションが発行するシステムコールの実行を監視することで，ポリシーを適用する．実装は，Linux カーネル上に行った．

2 章では関連研究について述べる．3 章では提案機構の設計とポリシーファイルについて説明し，4 章では提案機構の実装と評価について述べ，5 章では提案機構の評価を述べる．そして，6 章で本提案をまとめる．

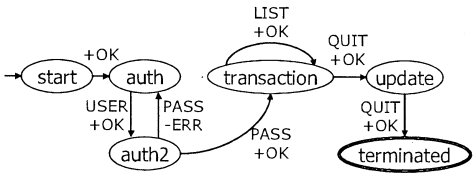

図 1: POP サーバの状態遷移

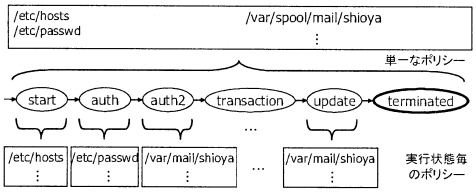

図 2: POP サーバの実行状態とポリシーの関係

2 関連研究

プロセスの実行状態によって，動的にアクセス権限を切り替えたり，モジュールの読み込みなどのプログラムの変化によってポリシーを切り替えるサンドボックスがいくつか研究されている．

Privtrans[2] は，監視するプログラムと権限が必要な関数呼び出しやデータアクセスを記述したファイルから，Monitor と Slave という 2 つのプログラムを自動生成する機構である．Slave では，監視するプログラムを実行させ，特別な権限が必要な処理は，Monitor が行うことで，権限を分離している．重要な資源は Monitor 側に持たせることで，その資源にアクセスする関数をチェックすることが可能である．しかし，プログラムの実行コンテキストによってポリシーの動的変更はできない．

SubDomain[5] は，プロセスが exec システムコールを呼び出す際に，ポリシーの切替えを行うサンドボックスである．また，監視するプログラムを変更し，change_hat というシステムコールを挿入することにより，ポリシーの変更を動的に行うことも可能である．本研究では，監視するプログラムの改変なしで，任意のユーザ関数毎にポリシーを切替えることが可能である．

阿部らの研究 [8] では，本研究と同様，関数毎にポリシーを動的に切替えることができる．関数呼び出しを監視するために，プログラムの実行前にポリシーを切替える呼び出し命令をトラップ命令に置

き換える．ポリシーの切替えは，制限するシステムコールが実行される前に行うため，ポリシー切替えが無駄になることもある．本研究では，ユーザプロセスからシステムコールが呼ばれ，カーネル中でシステムコールルーチンが実行される前に，必要であればポリシーを切り替える．関数呼び出しの履歴は，ユーザスタックの関数フレーム中にある戻り番地から求める．

セキュリティポリシーの記述を簡略化することを目的とした品川らの研究 [4] では，サーバプログラムの実行をフェーズに分けている．フェーズとは，例えばネットワークを通してメッセージを受け取る前の初期化フェーズや，メッセージを受け取った後のプロトコル処理フェーズなどである．この研究では，ネットワークを通したメッセージを受け取る前では，攻撃を受ける危険性がないことから，そのフェーズのポリシー記述を簡略化し，メッセージを受け取る前後でポリシーを切り変える．本機構では，任意のユーザ関数定義によってポリシーの切替えが可能である．

3 提案機構

3.1 基本設計

本研究は，プロセスの実行状態に応じて異なるポリシーを与えることができるようなサンドボックスを提案する．このサンドボックスは，実行状態が変化すると動的にポリシーを切りかえる．ここで，プロセスの「実行状態」とは，そのプロセスが現在実行している (C の) 関数に至るまでの関数呼び出しの履歴とする．そうするのは，プロセスの実行状態は，通常，アプリケーションのプログラムの中で，ユーザ定義関数としてコーディングされているためである．以後，このような関数呼び出しの履歴による実行状態のことを「実行コンテキスト」と呼ぶことにする．

POP サーバのひとつのインプリメンテーションである Qpopper4.0.4 を例にとって，実行コンテキストを説明する．

1 章で述べたように，POP サーバの動作は，いくつかの実行状態に分けることができる．表 1 は，Qpopper の実行状態とユーザ関数定義の関係を示している．また，transaction 状態では，機能毎に関数定義がされている．このことから，ユーザ定義関数

表 1: Qpopper での実行状態とユーザ定義関数の関係

実行状態	関数定義	処理内容
start	pop_init()	設定ファイル読み込みなど
auth	pop_user()	ユーザ情報の取得
auth2	pop_pass()	ユーザの認証
transaction	pop_list()	LIST コマンドの処理
	pop_retr()	RETR コマンドの処理
	pop_dele()	DELE コマンドの処理
	...	...
update	pop_updt()	メールスプールを更新
terminated	pop_quit()	終了処理

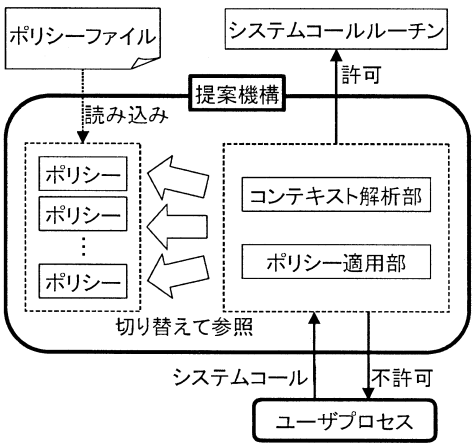

図 3: 提案機構の基本設計

の呼び出し履歴を見ることで，その実行状態を把握できることがわかる．よって，本研究での実行コンテキストは，ユーザ定義関数の呼び出し履歴とし，その履歴によって動的にポリシーを切り替えることができる．

本サンドボックスは，プロセスによるシステムコールの発行時にポリシーをチェックして，そのシステムコール実行の可否を決定する．そうする理由は，ユーザプロセスが計算機資源を利用するには必ずシステムコールを経由するので，システムコールの入口にチェック機構を置くのが最も合理的であるためである．

図 3 に，本研究が提案するサンドボックスの全体像を示す．提案機構のサンドボックス処理は，実行コンテキスト解析部とポリシー適用部からなる．実行コンテキスト解析部は，システムコールを発行したときのユーザ定義関数の呼び出し履歴を解析する．

```
 1   main() {
 2     brk()
 3     open("/dev/null", 01102, 0666)
 4     clone(_, 0, 0x01200011, _)
 5     exit_group(0)
 6     setsid()
 7     chdir("/")
 8     fstat64(_, _)
 9     socket(2, 1, 0)
10     bind(_, {
11                 0:(int) ;
12                 4:(unsigned short)HTONS(110);
13                 6:(int)INET_ADDR("0.0.0.0");
14               }, 16)
15     ...
16   }
17   pop_init() {
18     uname(_)
19     gettimeofday(_, _)
20     getpid()
21     open("/etc/host.conf", 0)
22     ...
23   }
24   pop_user() {
25     open("/etc/passwd", 0)
26     rt_sigaction(14, _, NULL, 8)
27     ...
28   }
29   pop_pass(char* p) {
30     open("/var/mail/" ++ p[34–38], 0)
31     ...
32   }
33  ...
```

図 4: Qpopper のポリシーファイル

ポリシー適用部は，実行コンテキスト解析部が求めた解析結果に対応するポリシーを参照し，システムコールの実行を判断する．参照するポリシーは，ポリシーファイルから提案機構に読み込んで，実行コンテキストによって切り替えられる形で保持する．

3.2 ポリシー定義

提案機構でのポリシー定義はアプリケーション毎に用意する．そこには，アプリケーションで使用されているユーザ定義関数毎に，そのユーザ定義関数の実行中に実行が許可されるシステムコール呼び出しを列挙する．それらのシステムコールは，ユーザ定義関数から直接呼ばれていても良いし，別の関数を経由して間接的に呼ばれていても構わない．ポリシー定義に記述されていないシステムコールは提案機構によって実行が拒否される．

ポリシー定義は，サンドボックスにより監視されるアプリケーションの開発者が記述することを想定している．開発者ならば，手元にソースコードがあり，ユーザ関数定義の中から呼び出されるシステムコールを把握することが可能であるため，必要最小限のシステムコールの実行を許可するようなポリシー定義の作成が期待できる．

ここで，Qpopper を例にとり，ポリシーの定義方法を説明する．図 4 に Qpopper のポリシー定義の一部を示す．

1 行目は，ユーザ定義関数 main に対するポリシー定義であることを示している．2 行目から 16 行目までが，ポリシー定義として列挙した実行を許可するシステムコールである．3 行目では，open("/dev/null", O_RDWR | O_CREAT | O_TRUNC, 0666) の実行を許す定義である．O_RDWR などの値は，ポリシー作成者であるアプリケーションの開発者が調べて記述する必要がある．

システムコールの引数で，特に注目する必要の無いものには，_ を書く．4 行目のような記述をすると，第一引数，および第四引数の値に関わらず，第二引数の値が 0，かつ第三引数が 0x01200011 である clone システムコールについてのアクセス権を与えることを意味する．

図 4 の 10 行目で定義している bind システムコールは，その第二引数に構造体へのポインタを取る．このような場合は，{} で囲んだ中に，構造体のメンバに対する値を記述できる．更に，この構造体のメンバの値には，htons や inet_addr でエンコードされるものが与えられる．このようなメンバに対しては，HTONS や INET_ADDR という組み込みのシンボルを記述できる．

アプリケーションによっては，実行中に得られた情報により，アクセスを許可する資源が定まるようなものがある．例えば，POP サーバの transaction 状態では，その前に認証を行ったユーザに対するメールスプールのディレクトリに対するアクセスだけを許可したい．このような時に，ユーザ定義関数の引数を参照することができる．Qpopper の例では，図 4 の 29 行目から 32 行目のように書くことができる．pop_pass 関数は，第一引数に POP 構造体 (図 5) へのポインタ p をとり，そのポインタが指す構造体のメンバ user のメールスプールを読み込み権限でオープンする．つまり，p->user が "shioya" であれば，"/var/mail/shioya" をオープンする．29 行目の記述は，pop_pass 関数の第一引数は，構造体へのポインタをとることを表す．そして，30 行目にある open システムコールのアクセス権の記述で，そのポイン

```
1   typedef char BOOL;{
2   struct POP{
3       BOOL debug;
4       BOOL xmitting;
5       BOOL bStats;
6       BOOL dirty;
7       BOOL bKerberos;
8       char* kerberos_service;
9       BOOL server_mode;
10      int check_lock_refresh;
11      char* myname;
12      char* myhost;
13      char* client;
14      char* ipaddr;
15      unsigned short ipport;
16      BOOL bDowncase_user;
17      BOOL bTrim_domain;
18      char* user;
19      ...
20  };
```

図 5: Qpopper で使用する POP 構造体

タが指す構造体のメンバを参照している．p[34–38] と書くことで，POP 構造体のメンバである user を参照できる．また，++ は文字列の連結を意味する．

4 実現

4.1 LKM の利用

サンドボックスの実装法には，大きく分けると以下の 4 つの方法がある．

1. アプリケーションの書き換え
2. ptrace システムコールの利用
3. カーネルの改造
4. Loadable Kernel Module (LKM) の利用

1 は，アプリケーションのコードを書き換えることによって，計算資源を操作する処理の前に，チェックコードを挿入するものである．この方法は，資源を操作する命令ごとにチェックを行えるので，細かい粒度のサンドボックスが実装できるが，実装が繁雑で難しい．2 は，親プロセスによって子プロセスの実行を監視できる ptrace システムコールを利用する方法である．この方法は，監視対象の子プロセスがシステムコールを呼んだり，シグナルを配送するたびに，子プロセスを停止させ，親プロセスでチェック処理を行う．実装しやすいが，親プロセスが介入するため，実行効率が大きく低下する．3 は，カーネルのコードを直接書き換えることで，実行を監視するものである．この方法は，カーネルのバージョンに強く依存する．4 は，システム起動時に動的にカーネルにコードを挿入する仕組みである．この方法は，サンドボックスをカーネルと切り離して実装でき，また 2 よりもオーバーヘッドを抑えることができる．3 と同様にカーネルのバージョンに依存するが，3 よりもバージョンアップに追従しやすい．よって，提案機構では，LKM を用いて実装を行った．Linux カーネルのバージョンは，2.6.8 である．

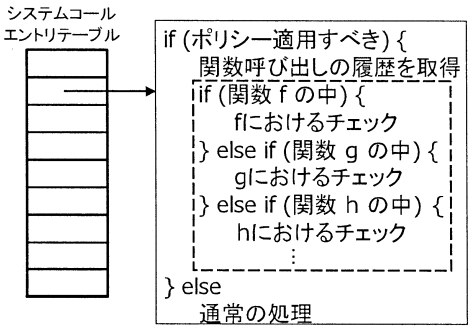

図 6: 提案機構が提供するサンドボックス処理の流れ

4.2 サンドボックス処理の流れ

図 6 に，提案機構が提供するサンドボックス処理の流れを示す．提案機構では，システムコール発行時にポリシー適用を行うために，システムコールエントリを書き換えることで，システムコールをフックする．Linux kernel 2.6 では，LKM からシステムコールエントリテーブルは参照できないようになっているが，シンボルマップを走査することでシステムコールエントリテーブルの開始アドレスを得ることができる．そこで，システムコールテーブルに，本機構によるポリシー適用処理を行う関数へのポインタを格納しておく．

提案機構によるポリシー適用の流れは以下のようである．

1. ユーザプロセスから発行されたシステムコールをフックし，提案機構によるポリシー適用処理を開始する．
2. システムコールを呼び出したユーザプロセスが，提案機構の監視対象であるかを判断する．

監視対象プロセスでなければ，本来のシステムコールルーチンを実行する．

3. 監視対象プロセスならば，システムコールを発行した時点でのユーザ定義関数呼び出しの履歴を，ユーザスタックの関数フレーム中の帰り番地から得る．

4. 得られたユーザ関数に対するアクセス権と照合し，そのシステムコールを許可する場合は，本来のシステムコールルーチンを実行する．拒否の場合は，提案機構がシステムコールの失敗を意味する値をユーザプロセスに返す．

上の4において，3で得た帰り番地と監視対象アプリケーションの名前表を用いてユーザ定義関数名を得る．アプリケーションの名前表は，事前にnmコマンドにより得ておき，ポリシー定義と共に保持しておく．

4.3 提案機構の構成

提案機構は2種類のLKMから成る．その構成を図7に示す．

1つは，システムコールをフックし，ユーザプロセスの実行コンテキストの情報を得るコンテキスト解析モジュールである．こちらは，ポリシーの情報を保持せず，したがってポリシーの適用はしない．それをするのは，もう1つのモジュールであるポリシー適用モジュールである．このように2つに分けた理由は，複数のアプリケーションを提案機構のサンドボックス内で動かす場合，アプリケーション毎にポリシー定義モジュールを用意すれば良いようにするためである．

5 評価

提案機構の有効性を検証するため以下のような実験を行った．本機構により，動的なポリシーを変更でき，攻撃を受けたユーザ関数によって与えられている権限のみ実行可能なことを確認した．

代表的なPOPサーバであるQpopperを改造し，メール受信の際に使用するコマンド処理部に以下のような脆弱性を意図的に入れて検証を行った．transaction状態において，サーバが持つメールの件数とバイト数を表示するLISTコマンドは本来引数に負

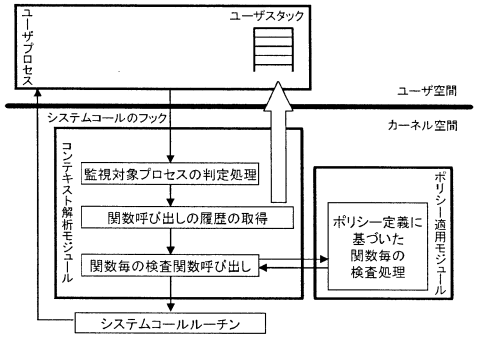

図7: 提案機構の実装

の値をとることは無く，負の値を引数としてコマンドを実行するとエラーとなる．その部分を改造し，負の値を引数としてLISTコマンドを実行すると/etc/passwdをopenするようにした．/etc/passwdは，auth状態では参照されるが，transaction状態には必要無い．このような脆弱性に対して，openシステムコールについてのみのポリシーを適用させたQpopperに攻撃を行ったところ，/etc/passwdのopenはパーミッションエラーとなった．この検証により，提案機構が実行のコンテキストによって，ポリシーを変更し，適用できていることが示された．

6 まとめ

プロセスのコンテキストによって，動的にポリシーを変更可能なサンドボックスを提案，実装した．また，実験によりその有効性を検証した．

提案機構は，ユーザ関数毎にアクセス権を記述することで，きめ細かいポリシーを作成している．その反面，1プロセスの実行開始から終了まで同一のポリシーを適用する従来のサンドボックスよりも，ポリシー作成が複雑になっている．今後の課題として，本機構のポリシー作成を支援する機構が必要である．

参考文献

[1] 大山恵弘：ネイティブコードのためのサンドボックスの技術，コンピュータソフトウェア，Vol. 20, No. 4, pp. 55–72 (2003).

[2] David Brumley and Dawn Song: Privtrans: Automatically Partitioning Programs for Privilege Separation.,

Proceedings of the 13th USENIX Security Symposium, (2004).

[3] Niels Provos: Improving Host Security with System Call Policies., *Proceedings of the 12th USENIX Security Symposium*, pp. 257–272 (2003).

[4] 品川高廣, 河野健二, 実行時のフェーズを考慮したセキュリティポリシー記述の簡略化, **先進的計算基板システムシンポジウム** *SACSIS2006*, pp. 295–503 (2006).

[5] Cowan, C., Beattie, S., Kroah-Hartman, G., Pu, C., Wagle, P. and Gligor, V., SubDomain: Parsimonious Server Security., *Proceedings of the 14th USENIX Security Symposium (LISA 2000)*, (2000).

[6] Erlingsson, Ú. and Schneider, F.B., SASI enforcement of security policies: a retrospective., *Proceedings of the 1999 workshop on New security paradigms*, pp. 87–95 (1999).

[7] 品川高廣, 河野健二, 高橋雅彦, 益田隆司, 拡張コンポーネントのためのカーネルによる細粒度軽量保護ドメインの実現, **情報処理学会論文誌**, Vol. 40, No. 6, pp. 2596–2606 (1999).

[8] 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, **コンピュータシステム・シンポジウム論文集**, Vol. 20, No. 3, pp. 61–68 (2003).