平成19年度　卒業研究論文

静的解析に基づく侵入検知システム

指導教官
齋藤 彰一 准教授

名古屋工業大学 情報工学科
平成16年度入学 16115102番

鶴田 浩史

目 次

第1章

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。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ており、セキュリティホールを悪用することで、不正アクセスを行うワームなどによる被害が発生している。なかでも、バッファオーバーフロー攻撃やフォーマットストリング攻撃による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のに侵入検知システムが有効である。検知する対象や目的によってさまざまな検知手法が存在する。一般的な方法としてパターンマッチングとよばれる検知手法が用いられている。この検知手法は、ネットワークを流れるパケットと攻撃の特徴を記述したパターンの情報を持っているシグネチャを比較する手法である。パターンマッチングによる検知は既知の攻撃パターンによる攻撃に対しては、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無い攻撃パターンによる攻撃は検知できないという問題がある。

最近では、正常な運用時とは違う状態を異常と判断する異常検知と呼ばれる手法が用いられてきている。この検知手法は、正常な運用時の状態をパターンとして持ち、正常な運用時とは異なる状態を検知したとき異常と判断する手法である。異常検知では、未知のセキュリティホールに対する攻撃であっても検知することができるため、現在、研究が盛んに行われている。

そこで、本論文では、ライブラリ関数の実行順序とシステムコールの群を用いた異

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により得られた情報を利用する。具体的には、システムコールを発行しているライブラリ関数の呼び出し元であるユーザ関数の実行位置の情報、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情報を正常な運用時の情報として利用し、異常な動作を判定する。

第2章

不正アクセスとその対策

2.1　不正アクセス

不正アクセスとは、あるコンピュータに対して正規のアクセス権限を持っていない人が、システムの脆弱性 (セキュリティホールなど) を利用して不正にコンピュータにアクセスしようとすることである。この不正アクセスの対策として、ファイアウォールや侵入検知システム等がある。

2.2　不正アクセスの被害

不正アクセスにより、さまざまな問題が発生している。たとえば、サーバが過負荷になりサービスの提供が阻害されるという場合である。また、Web ページの改竄により、公開されている情報を書き換えられる。別の偽 Web ページへ誘引され、個人情報などが搾取される。管理者権限を乗っ取られ、重要な情報が盗まれたり、改竄されたりする被害が多数発生している。

2.3　不正アクセスの種類

不正アクセスには、さまざまな方法が存在する。不正アクセスは、セキュリティホールを利用した方法が大部分を占める。そこで、本節では、セキュリティホールについて述べ、そのセキュリティホールを悪用した不正アクセスの代表的な例、バッファオー

バフロー攻撃、フォーマットストリング攻撃について述べる。

2.3.1 セキュリティホール

セキュリティホールとは、コンピュータソフトウェアの欠陥のひとつで、本来操作できないはずの操作 (権限のないユーザが権限を超えた操作を実行できる等) ができてしまったり、見えるべきではない情報が第三者に見えてしまうような不具合のことである。

2.3.2 バッファオーバフロー攻撃

バッファオーバフローは、セキュリティホールの存在によってバッファ領域を上書きしてしまい、誤動作を引き起こすものである。例えば、C 言語で図 2.1 のようにプログラムを書いた場合、関数内で定義された変数は通常、扱うデータを一時的に格納するために記憶領域をスタック上に用意する。

```
f1(void){
    char str[32];

    strcpy(str, "ABCDEFGHIJKLMNOPQRST");
    f2(str);

    return;
}

f2(char *p){
    char s[16];

    strcpy(s, p);        ---------- (1)

    return;
}
```

図 2.1: サンプルプログラム

関数 f1() を実行中、関数 f2() を呼び出す手順は、関数 f1() へのリターンアドレスを一時的にスタック (図 2.2(a) 参照) に格納し、その後に関数 f2() の処理を実行する。こ

こで、関数 f2() で使用される変数の格納領域が、スタックの f1 のリターンアドレスの上位アドレスに格納される (図 2.2(b) 参照)。関数 f2() の (1) では、確保した記憶領域の大きさを超えた文字列でないか確認せずに、データをコピーしている。

関数 f2() で確保した記憶領域の大きさを超えないデータをコピーした場合は、プログラムは正常に動作する。しかし、図 2.1 のプログラムのように 20 文字のデータを 16 文字分しか確保していない領域にコピーした場合、関数 f2() の変数の格納領域を超えてしまい、関数 f1() へのリターンアドレスが破壊され、図 2.2(c) のようになる。これをバッファオーバフローという。

関数 f2() の処理が終了すると、関数 f1() へのリターンアドレスを参照して、関数 f1() へ処理が戻るはずが、バッファオーバフローにより上書きされた値を参照するので、意図しない動作を引き起こしてしまう。

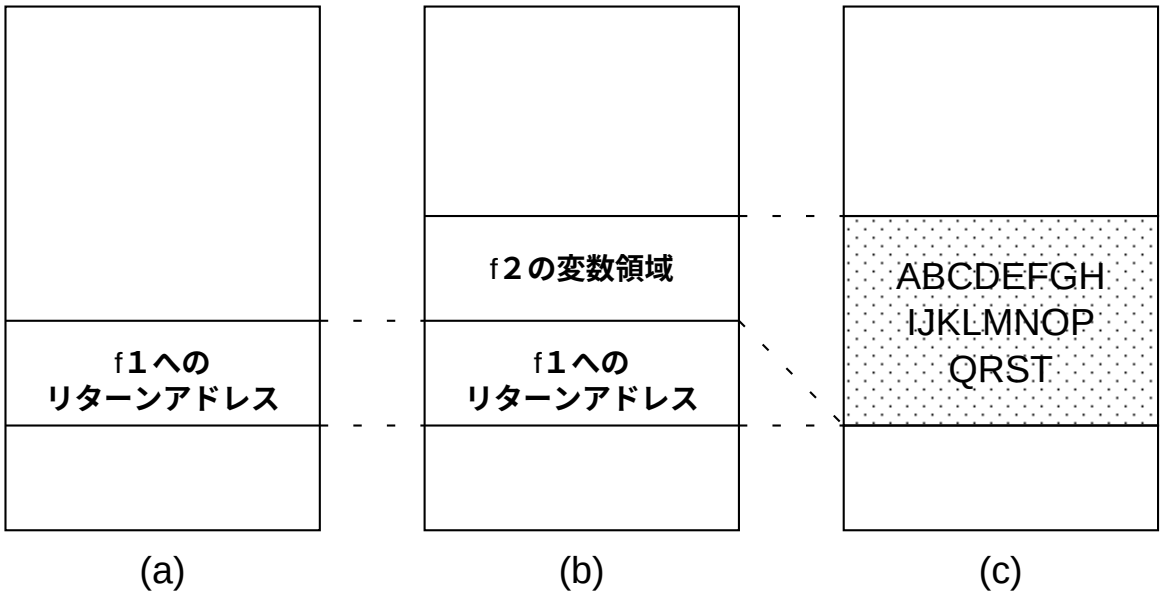

図 2.2: バッファオーバフローの仕組み

このような性質を利用し、悪意を持ったユーザは、自分の望む動作をするコードをふくみ、さらに、リターンアドレスを意図した値に書き換えるように細工したデータを作成する。これをプログラムに与えることによって、挿入したコードに動作が移るようにすることが可能となる。このような操作により、プログラムの動作を変えること

で、任意のコードを実行させることができる攻撃をバッファオーバフロー攻撃とよぶ。

バッファオーバフローは潜在的に大量に存在する問題であり、もっとも重大なセキュリティホールのひとつと考えられている。そのため、バッファオーバフローの脆弱性が発見されると、優先的に脆弱性を修正される。

2.3.3 フォーマットストリング攻撃

フォーマットストリングは、書式文字列とも呼び、printf 関数等のフォーマット関数によって使用されるものである。例えば、printf 関数で、printf(”%s”, string) と記述するように第１引数にフォーマットストリングのアドレスを指定するのではなく、printf(string) のように間違えて記述してしまい第 1 引数に文字列のアドレスを指定するとする。このように間違えて記述した場合に、”aaaaaaaa %08x %08x”等の文字列を利用し、”%08x %08x”で出力される値から、”aaaaaaaa”の格納アドレスの情報を取得する。このアドレスの情報をもとにリターンアドレスを書き換えることで攻撃ができる。

このように C 言語で printf 関数のように、”%d”、”%f”、”%n”などの書式記述子を利用した文字列データを生成する関数を利用している箇所を狙い、悪意を持ったユーザが望む動作をするプログラムコードを実行させる攻撃をフォーマットストリング攻撃とよぶ。

2.4 ファイアウォール

ファイアウォールとは、内部のコンピュータネットワークを外部のネットワークからの侵入を防ぐシステムである。ファイアウォールは、IP アドレスやポート番号を基に外部のネットワークからのパケットを監視し、不正なアクセスを検出・遮断する機能を実現している。

ファイアウォールは、許可されたアクセスを利用して行われる攻撃を防ぐことはできない。これには、許可されたアクセスで利用できるサービスにバッファオーバフローの脆弱性等がある場合、ファイアウォールを設置しただけでは防ぐことはできないた

めである。

ファイアウォールは、パケットフィルタリング型ファイアウォール、プロキシ型ファイアウォールに大別される。以下、2つの型のファイアウォールについて述べる。

2.4.1　パケットフィルタリング型ファイアウォール

パケットフィルタリング型ファイアウォールは、OSI参照モデルにおけるネットワーク層やトランスポート層で、外部のネットワークで流れるパケットの情報をもとに許可／不許可を判断するものである。このタイプのファイアウォールは、ルータやＬ３スイッチなどで代替することもできる。

送信元や送信先のIPアドレス、ポート番号などを監視し、それがあらかじめ設定されているIPアドレス、アプリケーションなどを判断し、通信を制御する。仕組みが単純なため、高速に動作するが、ルールの設定が煩雑になりやすいといった問題がある。

また、パケットフィルタリング型には、スタティックなフィルタリング方式とダイナミックなフィルタリング方式に分類される。スタティックなパケットフィルタリング方式は、内部のネットワークと外部のネットワークで双方向に通信を行う場合において、内部から外部へ行われる通信と外部から内部へ行われる通信の双方を許可する必要がある方式である。

ダイナミックなパケットフィルタリング方式は、内部のネットワークから外部のネットワークへ行われる通信を許可し、その通信への応答に関してのみ、外部からの通信を許可する、といった動作を自動的に行う方式である。

2.4.2　プロキシ型ファイアウォール

プロキシ型ファイアウォールは、プロキシサーバと同様の機能で、通信の制御を行うものである。なお、プロキシサーバは、代理サーバともいわれ、通信を代理接続する機能を有する。また、プロキシ型ファイアウォールは、サーキットレベルゲートウェイ型ファイアウォールとアプリケーションゲートウェイ型ファイアウォールに分類される。サーキットレベルゲートウェイ型ファイアウォールは、トランスポート層で、通

信の代替、処理を行う方式である。内部のネットワークから外部のネットワークへ接続する場合、内部からの通信に対して、サーキットレベルゲートウェイは、IP アドレスとポート番号を振り替え、外部と通信を行う。

アプリケーションゲートウェイ型ファイアウォールは、サーキットレベルゲートウェイと同様の機能を有しているが、トランスポート層で通信の代替をするのではなく、アプリケーション層で通信の代替、処理を行う方式である。このタイプは、アプリケーションの通信の中身も検査することができるが、動作が重くなるという問題がある。

2.5 侵入検知システム

侵入検知システム (IDS:Intrusion Detection System) は、不正アクセスやサービス拒否攻撃等のパケットが監視対象のネットワークに流れていたり、監視対象サーバにアクセスしている状態を、検知し警告するシステムである。侵入検知システムを大別すると、ネットワーク型侵入検知システム、ホスト型侵入検知システムの 2 種類に分けられる。以下、2 種類の侵入検知システムについて述べる。

2.5.1 ネットワーク型侵入検知システム

ネットワーク型侵入検知システム (NIDS:Network Based Intrusion Detection System) は、接続しているネットワークセグメントに流れるトラフィックを収集し、プロトコルのヘッダ部分やデータ内容を解析するシステムである。トラフィックを収集・解析することで、システムへの不正アクセス、サービス拒否攻撃、ウイルスやワーム等の攻撃、組織内からの不正行為を検知することができる。

NIDS は、攻撃パターンのデータベースを持っており、パケットのデータ部分に含まれる特定のパターンとマッチングさせることで不正通信を判断する (パターンマッチングによる検知)。過去に認識された攻撃パターンをデータベース化したものをシグネチャと呼ぶ。

また、パターンマッチングによる検知の他に、異常な通信を検出する方式がある。この方式は、IDS を一定期間運用して通常運用時のログイン時間帯、利用されるアプリ

ケーション、トラフィック量などの統計情報を作成し、通常運用時とは違う状態を検知した場合、不正通信と判断する。このような検出方式は、異常検知 (Anomaly Detection) といわれる。

NIDS の利点は、監視対象がネットワークとなるため、同一ネットワーク上の複数のシステムの監視をすることができること、監視対象となるシステムに負荷を与えないこと、どのような OS でも監視できることである。しかし、ネットワーク通信を行わず、ローカル環境のみで行われた攻撃の検知はできない。また、トラフィック内のデータ内容を解析することによって不正通信を検知するため、通信内容が暗号化されている場合、解析することができない。

2.5.2 ホスト型侵入検知システム

ホスト型侵入検知システム (HIDS:Host Based Intrusion Detection System) は、検知対象のシステムにインストールする形で導入される。OS が出力するログ、システムログ、ファイル改竄、サービスの実行時に発行されるシステムコールなどの監視をすることによって攻撃を検知する。

HIDS の利点は、ネットワーク環境などの外部的要因の影響が少なく、ギガビット環境などのトラフィック量が多い場合でも取りこぼしが発生せず、暗号化通信の影響もない。また、ローカル環境の攻撃も検知できる。しかし、導入されたシステム以外の攻撃は、検知できないため、検知対象とするシステム全てに導入する必要がある。

第3章

既存の手法と提案手法

実行プログラムを静的解析することで規則を作成する手法の特徴、問題点を述べる。そのあと、静的解析による提案手法について述べる。

3.1 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規則を作成する。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動作をパターン化することができる。そのため、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、未知の攻撃を検知できたり、正常であるのに異常であると判断されること (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 [2] が提案した手法は、システムコールの呼出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、考えうる状態の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 [3] が提案した手法は、システムコールが発行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、システムコールが発行されたとき、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 [4] は、Wagner らの手法であるシステムコールの呼出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。

槙本らの手法 [5] 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実行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。

3.2 提案手法

本論文では、静的解析を用いた規則に基づく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、実行ファイルの正常な動作をすべて網羅しているので、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。また、槙本らのシステムの一部である静的解析による規則で確認を行う機構を拡

張することで、提案する侵入検知システムを実現している。

提案手法の構成を図 3.1 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の情報を基に監視処理を行う。監視処理は、システムコールが正常な動作であるかを判断する処理である。詳細は、4 章で述べる。

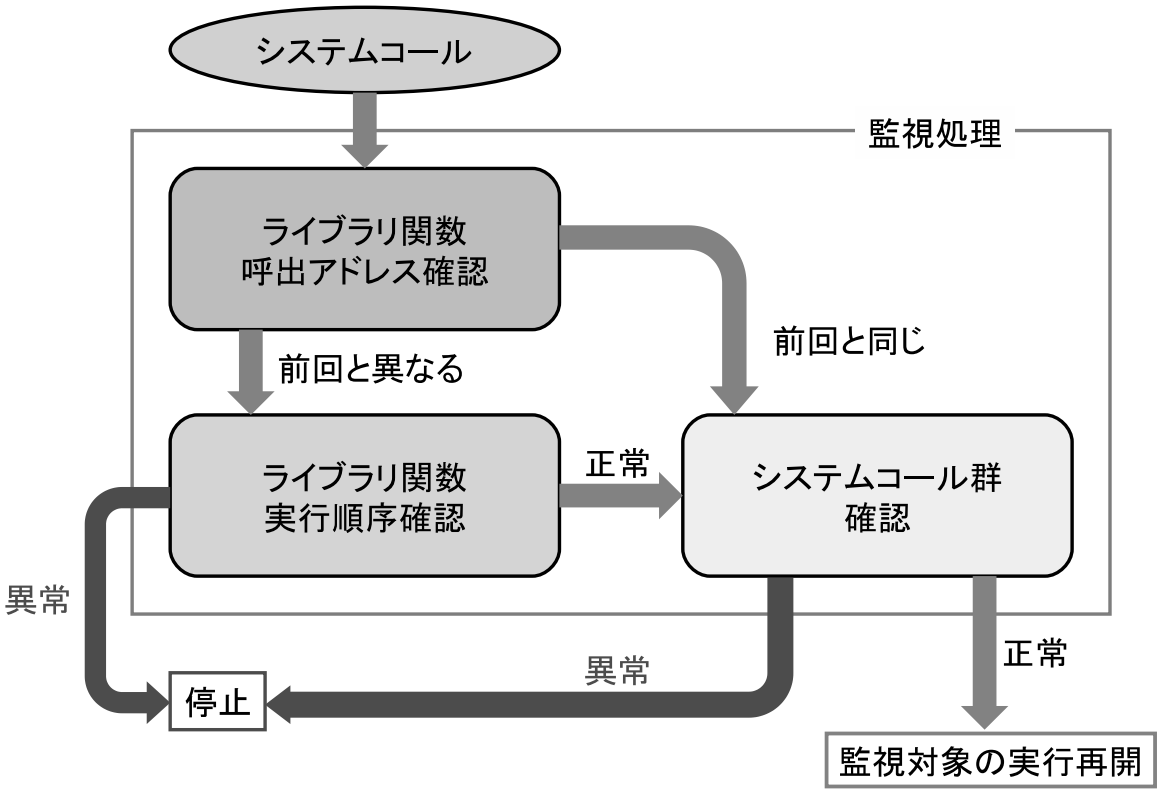

図 3.1: 監視処理の流れ

提案する方法では、システムコールの発行時を監視のタイミングとする。システムコールを呼び出した時、その呼び出し元となったライブラリ関数を呼び出しているユーザ関数の実行順を確認することでプログラム実行の全体の流れを把握する。

また、ユーザ関数の実行順序を確認することでシステムコールの呼び出し順序を確認しなくてもプログラムの正当性を確認することができると考える。このため、システムコールの実行順は、規則に含んでいない。

プログラム実行時のシステムコールを解析して不正アクセスを判定する機構として、ライブラリ関数の呼出アドレス確認機構、ライブラリ関数の実行順序確認機構、シス

テムコール群確認機構がある。また、不正アクセスと判断された場合、KILL システムコールを実行し、プロセスの実行を強制終了させることで、不正な動作を実行させないようにしている。

監視システムで用いる規則はライブラリ関数の呼出アドレスの規則、ライブラリ関数の呼出順序の規則、システムコール群の規則の 3 つを用いる。これらの規則はそれぞれ 4 章の実装で述べるライブラリ関数呼出アドレス確認機構、ライブラリ関数実行順序確認機構、システムコール確認機構に用いる。

提案手法で用いられる 3 つの規則はバイナリコード (実行ファイル) を逆アセンブルした結果を静的解析することで作成する。

3.2.1 ライブラリ関数の呼出アドレスの規則

ライブラリ関数の呼出アドレスの規則は、システムコールを呼び出している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の集合である。

本規則は、ユーザ関数から関数の呼び出しがある度に 1 つの規則を作成する。この 1 つの規則をブロックと呼ぶ。本規則の構成は、ブロックの ID 番号及びスタックの情報、すなわち、ユーザ関数へのリターンアドレスの情報をもとに得られる関数の実行アドレスの情報から構成されている。

3.2.2 ライブラリ関数の実行順序の規則

ライブラリ関数の実行順序の規則は、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元となるユーザ関数から今回の呼び出し元となるユーザ関数への遷移をまとめたものである。

本規則は、ライブラリ関数の呼出アドレスの規則と同様に、ユーザ関数から関数の呼び出しがある度に 1 つの規則を作成する。また、ライブラリ関数の呼出アドレスの規則とライブラリ関数の実行順序の規則の適応範囲は、ユーザ関数の定義の最初から最後までである。本規則の構成は、ユーザ関数の実行アドレスの情報とユーザ関数から遷移する可能性のある関数への相対アドレスから構成されている。本規則を、ライ

ブラリ関数の呼出アドレスの規則と対応するブロックにそれぞれ追加することで、規則を 1 つに抑えることができる。

3.2.3 システムコール群の規則

システムコール群の規則は、各ライブラリ関数が実行する可能性のあるシステムコールの情報を持った規則である。本規則は、ライブラリ関数が何回同じシステムコールを実行したとしても、１回とカウントしている。従って、システムコールの種類の情報から構成されている。また、この規則の適応範囲は、ライブラリ関数の定義の最初から最後までである。

ユーザ関数とライブラリ関数が、図 3.2 のように実行されるとし、具体例を述べる。

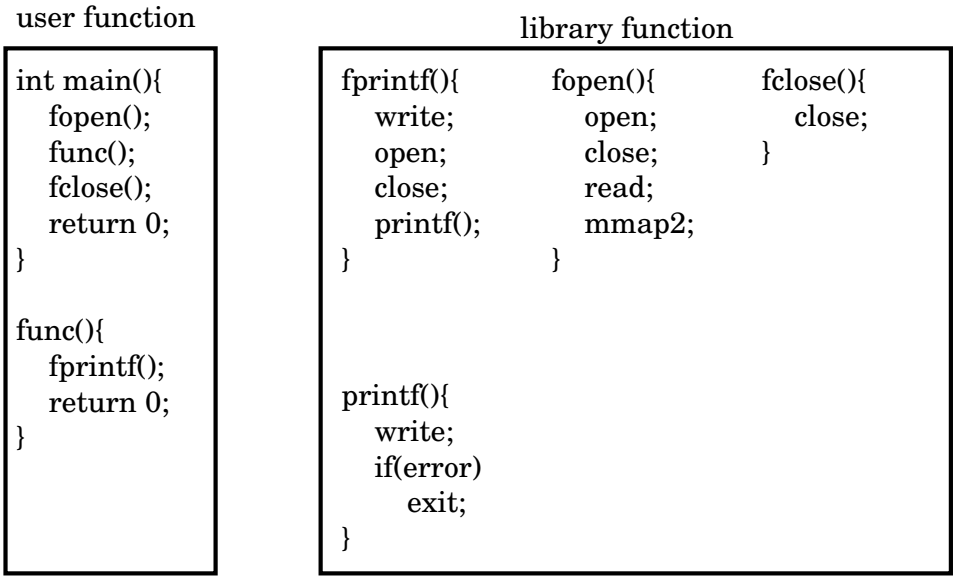

図 3.2: ユーザ関数とライブラリ関数の関係

user function はユーザ関数、liburary function はライブラリ関数を示す。また、ユーザ関数である、main() 及び func() からはライブラリ関数である、fopen()、fclose()、fprintf() を呼び出していることを表している。

library function の fprintf()、fopen()、fclose()、printf() はライブラリ関数を示す。ライブラリ関数である、fprintf()、fopen()、fclose() および printf() からはシステムコー

ルである、write、open、close、read、mmap2、exit を実行していることを表している。

このようなユーザ関数とライブラリ関数が定義されていたとき、システムコール群の規則は、図 3.3 の system call rule のようになる。なお、ライブラリ関数のみを解析して作成する。

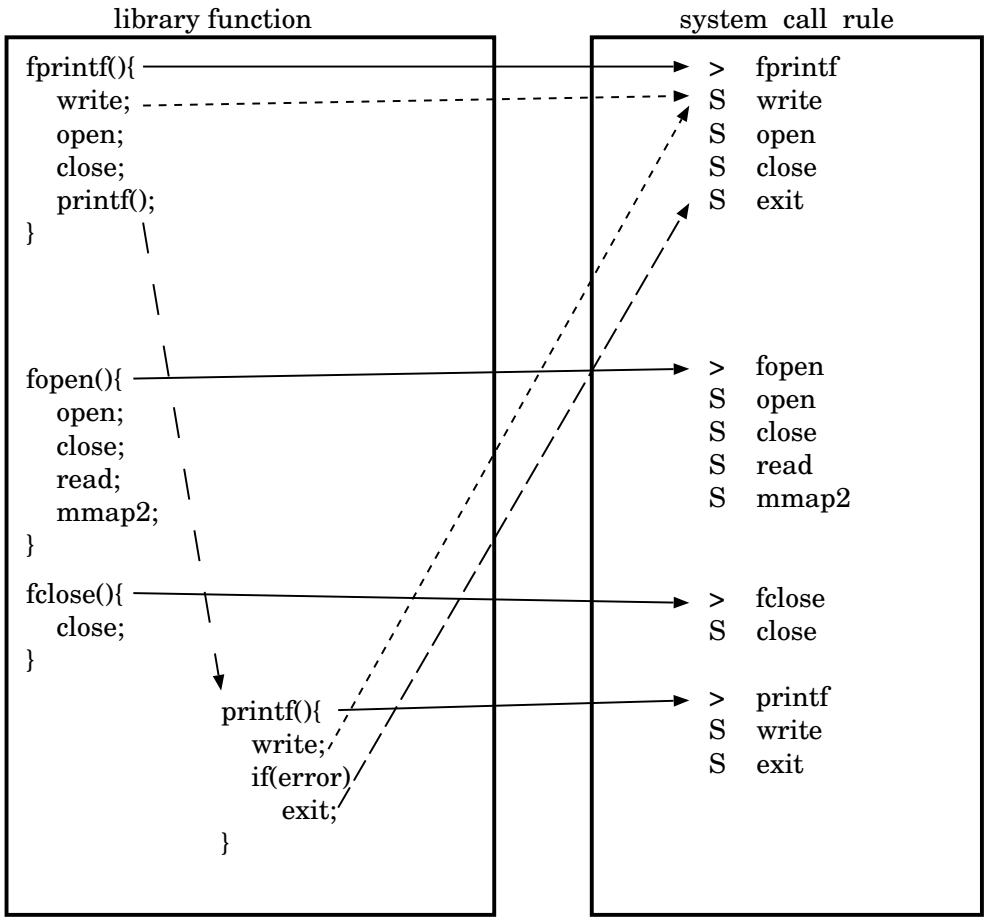

図 3.3: システムコール群の規則の関係

system call rule の「>」はライブラリ関数の先頭アドレス、「S」はライブラリ関数が実行するシステムコールであることを表している。ここでは名称を明記しているが、実際には、アドレスで表す。

ライブラリ関数 fprintf() は、システムコールだけでなく、ライブラリ関数 printf() を呼び出しているので、printf() 関数のシステムコールも fprintf() 関数の解析結果として

規則を作成する。ただし、fprintf() 関数でも、printf() 関数でも write システムコールが実行されているが種類だけの情報を規則として持つので、write システムコールは重複して規則に追加することはない。また、printf() 関数もライブラリ関数であるので、解析を行い規則を作成する。

このように、あるライブラリ関数がシステムコールだけでなく、他のライブラリ関数を呼び出している場合、そのライブラリ関数を解析し、他に呼び出すライブラリ関数やシステムコールがなくなるまで解析を行う。すべてのライブラリ関数に対して、同様に解析を行い、システムコール群の規則として作成する。

しかし、すべてのライブラリ関数の解析結果の情報を規則としているので、ユーザ関数から実行されないライブラリ関数はまったく比較に使用されない。そこで、ユーザ関数から呼び出されないライブラリ関数をシステムコール群の規則から除き、規則を適正化した。適正化したシステムコール群の規則のライブラリ関数をエントリ関数と呼ぶ。具体的には、printf() 関数は、ユーザ関数から呼ばれないので、解析の対象から除かれる。このように、比較に行われないライブラリ関数を除くことで、図 3.4 のように規則を作成した。また、適正化することで、システムコール群の規則のサイズを 177KB から 6KB に削減することができた。また、これにより検知にかかる時間を削減することもできた。実際に wc を静的解析して作成したシステムコール群の規則を付録に示す。

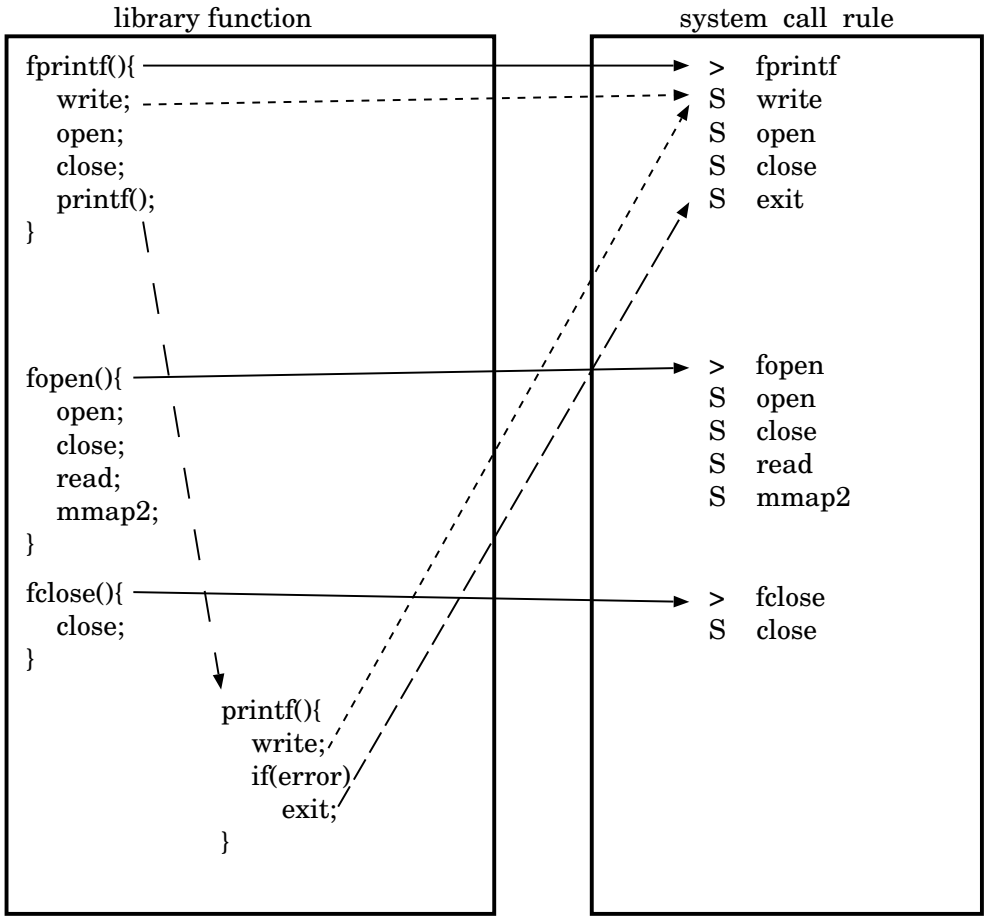

図 3.4: システムコール群の規則の削減

第4章

実装

3.2 節で述べた規則を用いて図 3.1 に示した監視システムを ptrace を用いて実装した。本章では、システムコールを用いた監視システムの実装について述べる。

4.1 システムコールのフック

ptrace を用いて、監視対象のプログラムがシステムコールを発行したときの情報を監視システムが取得する手順を図 4.1 に示す。監視を開始する手順は、1) 検知プログラム (親プロセス) が監視対象のプロセス (子プロセス) を作成する。2) 子プロセスは、ptrace を用いて、親プロセスに自分が監視対象のプロセスであることを通知する。3) 通知の後、監視対象のプログラムを実行する。システムコールをフックするには、4) 監視対象のプログラムは、システムコールを発行するたびに、ptrace を用いて、親プロセスにシステムコールを発行したことを通知する。5) 親プロセスは、監視対象のプログラムがシステムコールを発行した通知を受け取ると ptrace を用いて、監視対象のプログラムの実行を一時停止させ、PTRACE_SYSCALL が指定されたときの動作を確認する。6) そのときのシステムコールの情報を取得し、検知プログラムを実行する。検知プログラムの詳細は 4.2 で述べる。7) 検証が正しく終了したら、親プロセスは、監視対象のプログラムの実行を再開させる。以上の事を監視対象のプログラムが終了するまで繰り返す。また、異常が検知された場合、監視対象のプログラムを強制終了させ、危険な動作を実行させないようにしている。

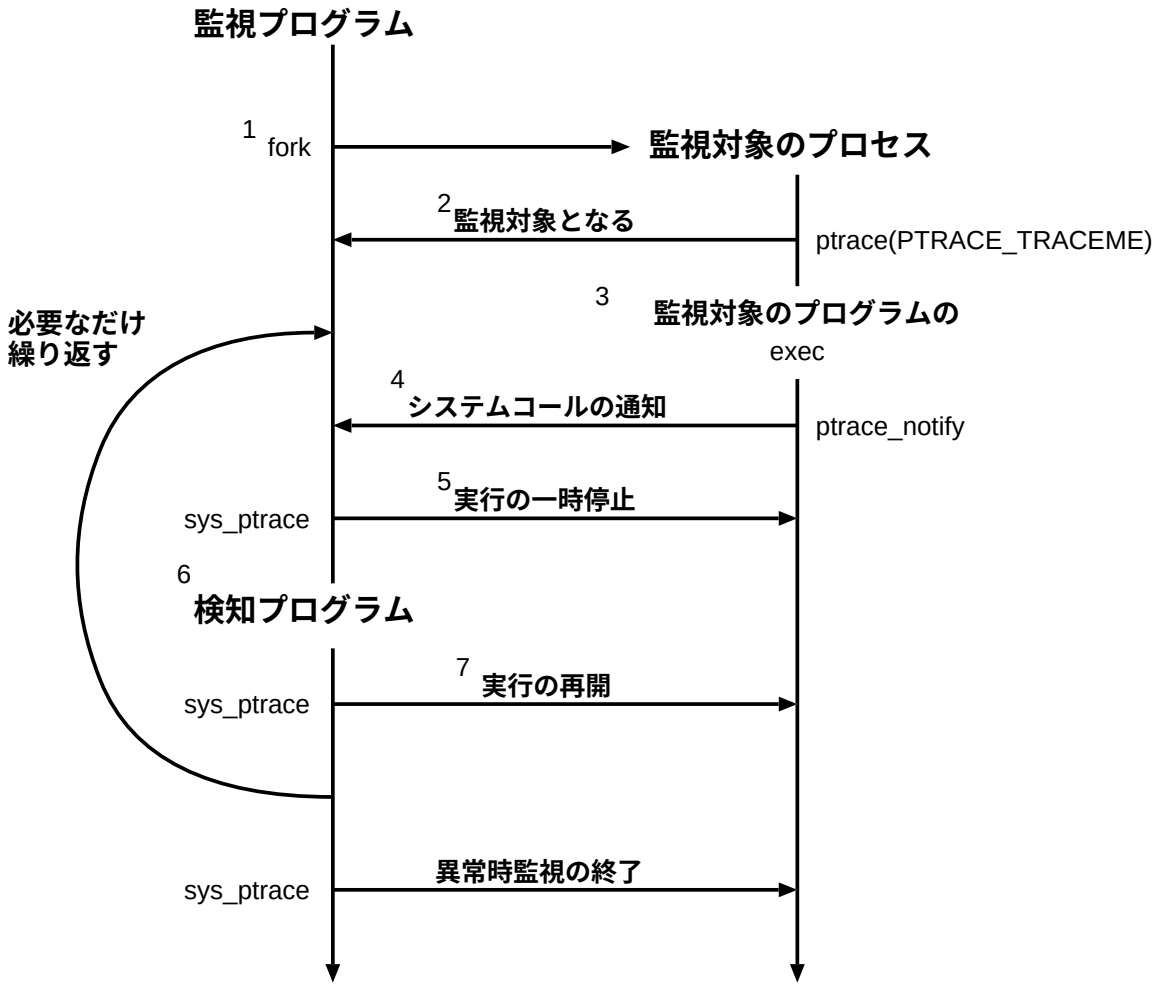

図 4.1: ptrace を用いたフックの流れ

4.2 検知プログラム

検知プログラムには、ライブラリ関数呼出アドレス確認機構、ライブラリ関数実行順序確認機構、システムコール群確認機構の 3 つの機構があることを 3.2 節で述べた。これら 3 つの機構について順次述べる。

4.2.1 ライブラリ関数呼出アドレス確認機構

ライブラリ関数の呼出アドレスの確認機構は、システムコールを呼び出しているライブラリ関数の呼出アドレスとライブラリ関数の呼出アドレスの規則と比較する機構である。

システムコールが発行されると、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、図 4.2 のように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。

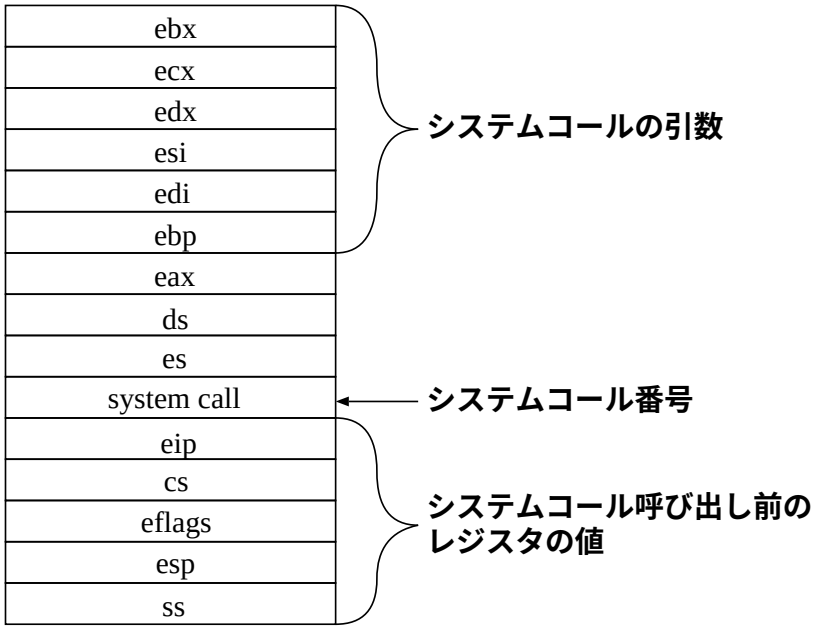

図 4.2: 特権モードに移行したときのスタックの状態

このとき、ユーザスタックに積まれている ebp 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出アドレスの確認を行う。この確認を行うことで、スタック内などから異常なアドレスからのライブラリ関数の呼び出しを異常であると検知できる。

また、システムコールが実行されたとき、前回呼ばれたシステムコールと同じライブラリ関数から呼ばれているかを比較し、同じライブラリ関数の場合、システムコー

ルの確認を行う機構に遷移し、違うライブラリ関数の場合、ライブラリ関数の実行順序の確認を行う機構に遷移する。

4.2.2　ライブラリ関数実行順序確認機構

ライブラリ関数実行順序確認機構は、ライブラリ関数の呼出アドレスが前回のライブラリ関数の呼出アドレスの順序関係を確認する機構である。今回システムコールを呼び出したライブラリ関数の呼出アドレスが、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確認する。この確認を行うことで、正常な実行順とは異なる実行順で実行するライブラリ関数を異常であると検知できる。

なお、この機構は、今回システムコールを呼び出したライブラリ関数の呼出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出アドレスが異なる場合のみ実行する。

4.2.3　システムコール群確認機構

システムコール群確認機構は、ライブラリ関数呼出アドレス確認機構で確認したライブラリ関数の呼出アドレスから呼ばれるライブラリ関数にシステムコールが実行されていることを確認する機構である。今回実行するシステムコールが実行されるべきであるライブラリ関数に含まれていない場合、異常であると検知し、監視対象のプログラムを終了させる。

第5章

評価

本章では、4 章の内容を Linux 上に実装し、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

実験環境を表 5.1 に示す。

表 5.1: 実験環境

OS	:	Fedora Core 5
CPU	:	Pentium 4（2.40GHz）
Memory	:	512MB
Kernel	:	version 2.6.16

実験に用いられる異常検知システムは C 言語で記述されている。異常検知システムは、4 章で述べたように Linux の ptrace を用いて監視対象のプログラムの実行を監視する。監視の対象となる実行ファイルは静的にリンクされたものを使用した。

監視の対象となるプログラムは GNU の wc を使用した。wc は、対象となるファイルのバイト数、単語数、行数をカウントするプログラムである。これを用いて、約 15MB のテキストファイルにおいてバイト数、単語数、行数を出力する動作に要する時間を計測した。

5.2 結果・考察

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。監視システムを使用した場合は、全関数を監視した場合とエントリ関数のみを監視した場合を計測した。その結果を、表5.2に示す。倍率は、全く監視しない場合を1として正規化している。

表 5.2: 実験結果

監視システム	実行時間 (倍率)
なし	0.472 (1.00)
全関数を監視	0.715 (1.51)
エントリ関数のみを監視	0.525 (1.11)

wcの測定時間は、全関数を監視した場合場合は、全く監視しない場合に比べて約1.5倍になったのに対して、エントリ関数のみを監視することによって、約1.1倍にまで抑えることができた。

システムコール群の規則は、全ライブラリ関数の場合とエントリ関数のみの場合を比較すると、全ライブラリ関数が1118関数あるのに対して、エントリ関数は39関数しかない。全ライブラリ関数では比較に使用されないライブラリ関数は90%以上に及ぶため、対象となるライブラリ関数の探索にかかる時間がエントリ関数のみの場合に比べて大きくなったと考えられる。

なお、実行時に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書き換えて評価を行った場合、システムコールが発行されたときに意図しないシステムコールが発行されたと通知され、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が確認できた。

第6章

まとめ

システムコールが発行されたタイミングでプロセスを監視するシステムを提案した。システムコールが発行されたとき、監視対象のプロセスを一時停止させて、そのときのユーザスタックに積まれている情報を取得する。取得した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。また、実行ファイルを静的解析することで、正常な動作の場合のパターンを規則として作成する。そして、この規則を、監視対象のプログラムの実行時のシステムコール及びスタックと比較することによってプロセスの異常な動作を検知するシステムを実装した。

プログラムの流れを掴むための方法として、システムコールの実行位置ではなく、システムコールを発行したライブラリ関数を呼び出しているユーザ関数の実行順序を確認した。ユーザ関数の実行順序を確認することで、実行の遷移を確認することができ、プログラムの全体の流れを掴むことができた。

全ライブラリ関数の解析結果の情報からなる規則からユーザ関数が呼び出すライブラリ関数を解析結果を規則とすることでシステムコール群の規則のサイズを削減し、検知にかかる時間を大幅に短縮することができた。また、意図しないシステムコールが発行された場合、プログラムの動作を終了させることができ、異常を検知することが確認できた。

しかし、静的解析により作成した規則は、関数ポインタに未対応なので、関数ポインタから遷移したライブラリ関数から呼び出されるシステムコールの確認ができない。

したがって、関数ポインタから遷移したライブラリ関数を解析できるようにプログラムを改良し、その解析結果を規則に組み入れるようにすることで、さらに検知の精度をあげていく予定である。また、小さなプログラム (wc) でしか動作の確認と評価ができていないので他のプログラムやより規模の大きなプログラムに対して同様の検証ができるようにしていきたい。

謝辞

本研究のために、多大な御尽力を頂き、御指導を賜った名古屋工業大学の松尾啓志教授、津邑公暁准教授、齋藤彰一准教授、松井俊浩助教に深く感謝します。

また、本研究の際に多くの助言、協力をして頂いた齋藤研究室の方々に深く感謝致します。

参考文献

[1] 高橋浩和, 小田逸郎, 山幡為佐久 著:”Linux カーネル解読室 2.6”.

[2] David Wagner,Paolo Soto:”Mimicry Attacks on Host-Based Intrusion Detection System”, Proc.9th ACM Conference on Computer and Communications Security(2002).

[3] Henry Hanping Feng,Oleg M.Lolesnikov,Prahlad Fogla, Wenke Lee,Weibo Gong:”Anomaly Detection Using Call Stack Information”, IEEE Security and Privacy 2003(May 2003).

[4] 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.SIG3(ACS 5),pp.11-20.

[5] 槙本祐司, 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,pp.23-30(2007).

付録

作成した wc のシステムコール群の規則を図 1 に示す。

```
> 0804b558

> 0804b790
S 000000f0
S 0000010a
S 00000005
S 00000003
S 00000006
S 000000c3
S 0000005b
S 000000e0
S 0000010e
S 000000ae
S 000000af
S 000000fc
S 00000001
S 0000009e
S 00000092
S 000000c0
S 0000007d
S 000000c5
S 000000a3
S 000000b7
S 0000000d
S 00000004
S 00000066
S 000000dd
S 00000014
S 000000bf

> 0804d050

> 0804d12c

> 0804d4d4
S 00000092
S 0000005b
S 000000f0
S 000000e0
S 0000010e
S 000000ae
S 000000af
S 000000fc
S 00000001
S 0000009e
S 0000007d
S 000000c0
S 00000005
S 000000c5
S 00000006
S 00000003
S 000000a3
S 000000b7
S 0000010a
S 000000c3
S 000000bf
S 0000000d
S 00000004
S 00000066
S 000000dd
S 00000014

> 0804d4ec
S 000000b7
S 00000092
S 0000005b
S 000000f0
S 000000e0
S 0000010e
S 000000ae
S 000000af
S 000000fc
S 00000001
S 0000009e
S 0000007d
S 000000c0
S 00000005
S 000000c5
S 00000006
S 00000003
S 000000a3
S 0000000d
S 000000c3
S 00000004
S 00000066
S 000000dd
S 00000014
S 0000010a
S 000000bf

> 0804fedc
S 000000f0
S 0000005b
S 000000e0
S 0000010e
S 000000ae
S 000000af
S 000000fc
S 00000001
S 0000009e
S 00000092
S 000000c0
S 0000007d
S 00000005
S 000000c5
S 00000006
S 00000003
S 000000a3
S 000000b7
S 0000010a
S 000000c3
S 000000bf
S 0000000d
S 00000004
S 00000066
S 000000dd
S 00000014

> 080513b0
S 000000f0
S 000000e0
S 0000010e
S 000000ae
S 000000af
S 000000fc
S 00000001
S 0000009e

> 080519b4
S 000000f0
S 0000005b
S 000000e0
S 0000010e
S 000000ae
S 000000af
S 000000fc
S 00000001
S 0000009e
S 00000092
S 000000c0
S 0000007d
S 00000005
S 000000c5
S 00000006
S 00000003
S 000000a3
S 000000b7
S 0000010a
S 000000c3
S 000000bf
S 0000000d
S 00000004
S 00000066
S 000000dd
S 00000014

> 08051a98
S 000000f0
S 00000092
S 000000c0
S 0000007d
S 0000005b
S 000000e0
S 0000010e
S 000000ae
S 000000af
S 000000fc
S 00000001
S 0000009e
S 00000005
S 000000c5
S 00000006
S 00000003
S 000000a3
S 000000b7
S 0000010a
S 000000c3
S 000000bf
S 0000000d
S 00000004
S 00000066
S 000000dd
S 00000014

> 08052b88
S 000000f0
S 00000092
S 0000005b
S 0000000d
S 000000e0
S 0000010e
S 000000ae
S 000000af
S 000000fc
S 00000001
S 0000009e
S 0000007d
S 000000c0
S 000000c3
S 00000005
S 00000006
S 000000c5
S 00000003
S 000000a3
S 000000b7
S 00000004
S 00000066
S 000000dd
S 00000014
S 0000010a
S 000000bf

> 0805a1dc
S 000000f0
S 00000092
S 0000005b
S 0000000d
S 000000e0
S 0000010e
S 000000ae
S 000000af
S 000000fc
S 00000001
S 0000009e
S 0000007d
S 000000c0
S 000000c3
S 00000005
S 00000006
S 000000c5
S 00000003
S 000000a3
S 000000b7
S 00000004
S 00000066
S 000000dd
S 00000014
S 0000010a
S 000000bf

> 0805a1f8
S 000000f0
S 00000092
S 0000005b
S 0000000d
S 000000e0
S 0000010e
S 000000ae
S 000000af
S 000000fc
S 00000001
S 0000009e
S 0000007d
S 000000c0
S 000000c3
S 00000005
S 00000006
S 000000c5
S 00000003
S 000000a3
S 000000b7
S 00000004
S 00000066
S 000000dd
S 00000014
S 0000010a
S 000000bf

> 0805a760
S 000000f0
S 0000005b
S 000000e0
S 0000010e
S 000000ae
S 000000af
S 000000fc
S 00000001
S 0000009e
S 00000092
S 000000c0
S 0000007d
S 00000005
S 000000c5
S 00000006
S 00000003
S 000000a3
S 000000b7
S 0000010a
S 000000c3
S 000000bf
S 0000000d
S 00000004
S 00000066
S 000000dd
S 00000014

> 0805cdf0

> 0805d174

> 080604b0

> 08064b8c
S 00000092
S 000000f0
S 000000c0
S 0000007d
S 0000005b
S 000000e0
S 0000010e
S 000000ae
S 000000af
S 000000fc
S 00000001
S 0000009e
S 00000005
S 000000c5
S 00000006
S 00000003
S 000000a3
S 000000b7
S 0000010a
S 000000c3
S 000000bf
S 0000000d
S 00000004
S 00000066
S 000000dd
S 00000014

> 08064e50
S 00000092
S 0000005b
S 000000f0
S 000000e0
S 0000010e
S 000000ae
S 000000af
S 000000fc
S 00000001
S 0000009e
S 0000007d
S 000000c0
S 00000005
S 000000c5
S 00000006
S 00000003
S 000000a3
S 000000b7
S 0000010a
S 000000c3
S 000000bf
S 0000000d
S 00000004
S 00000066
S 000000dd
S 00000014

> 080662b0
S 000000f0
S 0000005b
S 000000e0
S 0000010e
S 000000ae
S 000000af
S 000000fc
S 00000001
S 0000009e
S 00000092
S 000000c0
S 0000007d
S 00000005
S 000000c5
S 00000006
S 00000003
S 000000a3
S 000000b7
S 0000010a
S 000000c3
S 000000bf
S 0000000d
S 00000004
S 00000066
S 000000dd
S 00000014

> 08066408
S 000000f0
S 000000a3
S 000000e0
S 0000010e
S 000000ae
S 000000af
S 000000fc
S 00000001
S 0000009e
S 00000092
S 000000c0
S 0000007d
S 0000005b
S 00000005
S 000000c5
S 00000006
S 00000003
S 000000b7
S 0000010a
S 000000c3
S 000000bf
S 0000000d
S 00000004
S 00000066
S 000000dd
S 00000014

> 08066f90

> 080674d0

> 0806766c

> 08067708

> 0806781c

> 08067ca4

> 08067cf4
S 000000f0
S 00000092
S 0000005b
S 000000e0
S 0000010e
S 000000ae
S 000000af
S 000000fc
S 00000001
S 0000009e
S 0000007d
S 000000c0
S 000000b7
S 000000a3
S 00000005
S 000000c5
S 00000006
S 00000003
S 0000010a
S 000000c3
S 000000bf
S 0000000d
S 00000004
S 00000066
S 000000dd
S 00000014

> 08068410

> 0806a444
S 000000b7
S 00000092
S 0000005b
S 000000f0
S 000000e0
S 0000010e
S 000000ae
S 000000af
S 000000fc
S 00000001
S 0000009e
S 0000007d
S 000000c0
S 00000005
S 000000c5
S 00000006
S 00000003
S 000000a3
S 0000000d
S 000000c3
S 00000004
S 00000066
S 000000dd
S 00000014
S 0000010a
S 000000bf

> 0806a49c
S 000000c3

> 0806a4cc
S 000000c5

> 0806a55c
S 00000005

> 0806a600
S 00000006

> 0806a650
S 00000003

> 0806be28
S 000000f0
S 00000092
S 0000005b
S 000000e0
S 0000010e
S 000000ae
S 000000af
S 000000fc
S 00000001
S 0000009e
S 0000007d
S 000000c0
S 00000005
S 000000c5
S 00000006
S 00000003
S 000000a3
S 000000b7
S 0000010a
S 000000c3
S 000000bf
S 0000000d
S 00000004
S 00000066
S 000000dd
S 00000014

> 0806c0f8
S 0000008c

> 0806c868

> 0806c9c0
```

図 1: 適正化したシステムコール群の規則