M-063

カーネル内の不正プログラムを検出するシステムの設計
Design of a Detection System for Malicious Programs in the Kernel

花田 智洋† 多田 好克†
Tomohiro Hanada Yoshikatsu Tada

1. はじめに

計算機システムにたいする不正アクセスが，大きな社会問題となっている．近年では，攻撃者が不正アクセスを行う際に使用する，カーネル内で動作する不正プログラムが出現し，問題になってきている [1]．この不正プログラムがカーネルに組み込まれると，自己の存在を隠ぺいするために，通常の処理結果にたいして情報を改ざんする．これにより，ユーザプログラムでは不正プログラムの検出が非常に困難になる．

そこで本研究では，カーネルに組み込まれた不正プログラムを検出するシステムを設計する．本研究での検出システムは，カーネルモジュールで設計されている．カーネルレベルで監視を行うことで，ユーザプログラムを用いたシステムでは検出できない不正プログラムの検出を可能にする．

2. 不正プログラム

攻撃者はシステムへの侵入後，今後のアクセス手段確立，システムへの攻撃，証拠の隠ぺい等を行う．これらを簡単かつ円滑に行うために，攻撃者は不正プログラムを作成し，使用する．

本研究では，ウィルス，ワーム，トロイの木馬など，悪意のあるプログラムのことを不正プログラムとする．また，検出対象とする不正プログラムは，システムへの攻撃を行うために，攻撃者がカーネルに組み込むものとする．

2.1 ステルス技術

不正プログラムが隠密活動をするための技術をステルス技術という．ステルス技術には，不正プログラム自身の隠ぺい，不正プログラムが実行されていることの隠ぺいなどがある．

すべての不正プログラムがステルス技術を用いて隠密活動をするとは限らない．しかし，隠密活動は多くの不正プログラムによく見られる特徴といえる．

2.2 ユーザレベルの不正プログラム

ユーザレベルの不正プログラムは，既存のユーザプログラムを悪意のあるプログラムに改ざんする．この場合，改ざんされたプログラムを正常なプログラムと比較することで，システムの異常を検出することが可能である [2]．

2.3 カーネルレベルの不正プログラム

カーネルに組み込まれる不正プログラムは，ユーザプログラムを直接改ざんしない．かわりに，ユーザプログラムが呼び出すオペレーティングシステム（以下，OSと略記する）の機能を改ざんする．したがって，ユーザプログラムは正常時と同じまま，システムを改ざんすることができる．

†電気通信大学 大学院情報システム学研究科, The Graduate School of Information Systems, University of Electro-Communications.

不正プログラムをカーネルに組み込むと，ファイルサイズが増加していないように見せること，チェックサムをオリジナルのものから変化させないようにすること，カーネル構造を改ざんすることなどができる．

カーネルに組み込まれる不正プログラムの代表的なものに，LKM-rootkit [1] がある．これは，LKM (Loadable Kernel Module) の技術を利用して，rootkit という不正プログラムをカーネル内に組み込むものである．

図1はカーネルに組み込まれる不正プログラムの概念図である．ユーザプログラムは正しい動作を行う．しかし，ユーザプログラムの呼び出し先であるカーネルが汚染されているので，ユーザは改ざんされた情報を得ることになる．

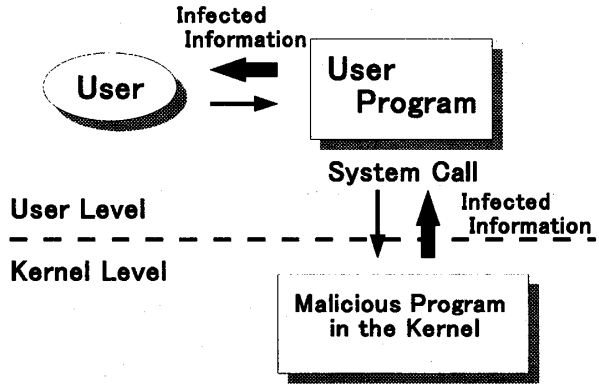

図 1: カーネルに組み込まれる不正プログラムの概念図

3. 本システムでの検出

本研究では，カーネルに組み込まれる不正プログラムを検出するカーネルモジュールを作成する．検出システムをカーネルモジュール化することで，検出動作をカーネルレベルで行うことができる．これにより，ユーザレベルからでは検出できない不正プログラムを検出することが可能になる．

3.1 カーネルレベル検出

ユーザレベルからの検出では，図1のように，ユーザプログラムを用いる．したがって，OSのサービスを利用して検出を行うことになる．カーネル内に不正プログラムが組み込まれると，OSやOSのサービスが信頼できなくなる．それゆえ，ユーザレベルから検出を試みても，ユーザは常に不正プログラムによって改ざんされた情報を得ることとなる．

そこで，カーネルレベルで検出を行えば，カーネルそのものの状態を取得することができる．カーネルレベルで監視することで，不正プログラムによって改ざんされ

た情報を用いずに，システムから不正プログラムを検出することができる．

3.2 検出手法

本システムでは，不正プログラムを検出するために，以下のような検出手法を用いる．

3.2.1 カーネル構造改ざん検出

カーネルに組み込まれた不正プログラムの多くは，隠密活動を行う．その際，ユーザレベルで見える部分から隠ぺいするために，ファイルやプロセスなどを管理するデータ構造から切り離すように見せかける．そこで，カーネルレベルで見えるファイルやプロセスの状態と，ユーザレベルで見える状態とを比較する．得られる状態が異なる場合，カーネルレベルでのみで得られる情報を糸口に，不正プログラムの検出を行う．

カーネル構造改ざん検出では，頻繁にプロセスの生成や削除などが起きるシステムにおいて，正確にシステムの状態を把握することが困難である．システムの状態取得時と比較時の状態が異なるからである．リアルタイムに検出を試みると，オーバヘッドが非常に大きくなる．それゆえ，オンデマンドに検出を試みる手法を採用する．

3.2.2 EPA（Execution Path Analysis）[3]

カーネルに組み込まれる不正プログラムには，システムを通常時と同じ動作させることと並行して，隠ぺい活動を行うものがある．例えば，ある特定の文字列だけ表示させないようにするために，ユーザに処理結果を返す前に値を改ざんする．その場合，通常の処理をする命令と，隠ぺいの処理のための命令が必要である．EPAは，隠ぺいを行うために通常時よりも命令数が多くなる，という特徴に着目し，システムに異常が発生したことを検出する．

3.2.3 ファイル整合性検査

不正プログラムがファイルとして組み込まれる方法には，新規にファイルが追加される方法，既存のファイルに追加する方法がある．これに対処するために，システムの正常時と現在の状態から得られる情報とを比較することで，システムの整合性を検査することができる．

3.2.4 カーネルメモリスキャン

不正プログラムは，メモリにロードされることで活性化される．そこで，カーネルのメモリ領域をスキャンすることにより，不正なプログラムがロードされていることを検出する．

カーネルメモリスキャンでは，監視する場所を誤ると，メモリの状態が頻繁に変わるにも関わらず，不正プログラム検出に役立たない場所を監視することになる．また，不正プログラムが利用するメモリの場所を特定することは非常に困難である．しかし，攻撃者が狙うメモリの位置を特定できれば，非常に有効な検出方法である．

4. 関連研究

カーネルに組み込まれる不正プログラム検出のために使用するツールには，ユーザレベルで動作するもの，カーネルレベルで動作するものに分類される．

ユーザレベルで動作するツールには，chkrootkit [2]，kstat [4] などがある．

chkrootkit はパターンマッチングにより不正プログラム検出を試みるツールである．したがって，不正プログラムが既知のものである場合のみ検出が可能である．

kstat はカーネルのメモリ状態を，正常時と比較することにより，システムの状態を監視するツールである．

カーネルレベルで動作するツールには，alamo [5] がある．alamo は，ディレクトリエントリを操作するシステムコール getdents と同等の機能を持つツールである．攻撃者が getdents をフックすると，ファイルやディレクトリの情報を自由に操作することが可能である．alamo 経由でファイル情報を取得することで，getdents のみをフックする攻撃には対処が可能である．しかし，getdents 以外のシステムコールがフックされると無力である．

5. まとめ

本研究では，カーネルに組み込まれる不正プログラムの包括的な検出を目指したシステムを設計する．本システムで検出を行えば，カーネルレベルからシステムの状態を監視することができる．そのため，既存のツールなどを用いたユーザレベルからでは検出できない不正プログラムを検出することができるようになる．

本研究で行う検出手法にたいする課題として，検出システムが不正プログラムによって攻撃されても，正常な動作をさせることがあげられる．検出システムを無効化されないためにも，何らかの防御手段を講じる必要がある．検出システムにトラップを設置し，検出システムに異常が発生した場合，アラートを発生できるようにすることを考えている．

今後は，システムを実装し，実際に不正プログラムを走らせた上で，本システムの有効性を検証する必要がある．また，未知の攻撃手法，検出システムへの攻撃にたいする対策も行う必要がある．

参考文献

[1] Jesus Molina and William Arbaugh, “Using Independent Auditors as Intrusion Detection Systems,” ICICS 2002, pp.291–302, December 2002.

[2] chkrootkit, http://www.chkrootkit.org/

[3] Jan K. Rutkowski, “Execution path analysis: finding kernel based rootkits,” Phrack, Volume 0x0b, Issue 0x3b, 2002.

[4] kstat, http://www.s0ftpj.org/

[5] alamo, http://www.rackspace.com/alamo/