コールスタック検査による Windows 版異常検知システム

山崎 大輔[†]　　大山 恵弘[†]

1. はじめに

UNIX を対象として、長年、システムコール列の学習による異常検知の研究が行われ、多くの手法が発表されてきた[1][2][3]。一方、システムコール列の学習による異常検知を Windows に適用した研究の発表は少ない。よって、UNIX を対象として設計、実装されている異常検知手法が Windows においても有効であるかどうかは、十分明らかにされてこなかった。本研究では、プログラムの正常な動作に関するデータベースを動的な学習によって生成する異常検知手法を扱う。具体的には、Windows 上のプログラムが発行するシステムサービス呼び出しを監視してこれを学習し、異常検知に使用する。ここで、システムサービスとは UNIX におけるシステムコールに対応する。

Windows におけるシステムサービス呼び出し列の学習による異常検知手法についての既存研究には、島本らの研究[3]がある。しかし、彼らの研究で用いられているのは、システムサービス呼び出し列の情報だけを用いた N-gram 法と呼ばれる単純な手法である。彼らの研究では、スタックの情報などを用いた高精度の異常検知システムを Windows に適用する試みは行われていない。そこで、本研究では Windows を対象として、システムサービスとスタックの情報を組み合わせて用いる異常検知システムを実装し、そのシステムの有効性を評価することを目指す。

2. スタック検査による異常検知

本研究では、Feng らによるコールスタック情報を用いた異常検知手法[3]を Windows 上に実装し評価する。これは、前回のシステムコール呼び出しから今回のシステムコール呼び出しまでの関数呼び出し履歴をそれぞれのコールスタック情報から推測し、異常検知に利用するという手法である。なお一般的に、スタック情報を用いると、mimicry attack や false positive が減少することが期待される一方で、単純な N-gram 法に比べるとオーバーヘッドが大きくなるという欠点がある。

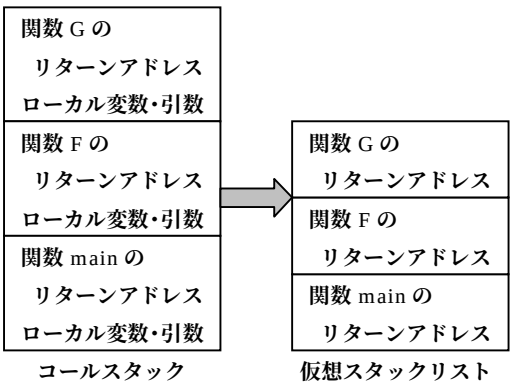

図１　仮想スタックリストの生成

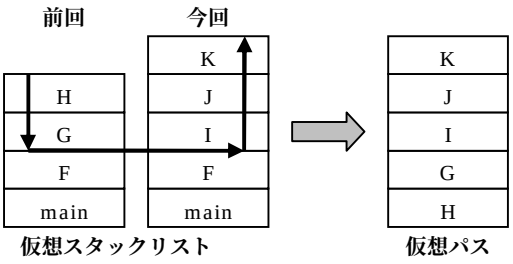

図２　仮想パスの生成

まず図１のように、それぞれのコールスタックのスタックフレームからリターンアドレスを取得して、リストを作る。これを仮想スタックリストと呼ぶ。次に図２のように、２つのリストを関数が実行された順番でつなげて、仮想的なパスを作る。つまり前回のコールスタックを底に向かって辿り、次に今回のコールスタックを上に向かって辿る。なお、

† 電気通信大学大学院電気通信学研究科情報工学専攻

前回と今回で重複するアドレスは除去する。これを仮想パスと呼ぶ。

本研究では、この仮想パスを異常検知に用いる。具体的には、まず学習モードでプログラムの正常な動作を記録、つまり仮想パス情報を収集する。次に検知モードでプログラムの実行を監視し、学習したことがない仮想パス情報が現れたら警告を発するものとする。

3. システムサービス呼び出しを監視

システムサービス呼び出しの監視には、島本らの手法[4]を用いる。具体的には、x86 アーキテクチャ命令の１つである sysenter を監視することで行う。これはカーネルモードへ移行するための命令であり、Windows 2000 までの int 命令による割り込みに変わって、Windows XP から使用されている。この命令における特に重要な動作は、命令発行時に EIP レジスタへ SYSENTER_EIP_MSR レジスタの値をロードし、特権レベル０に切り替えてカーネルモードルーチンの実行を開始する事である。すなわち、我々の監視モジュールへのアドレスを SYSENTER_EIP_MSR レジスタへロードしておけば、図３のように sysenter 命令の実行を監視することができるという事である。なお、このままでは本来のシステムサービスが呼び出されないので、監視モジュールから呼び出しておく。

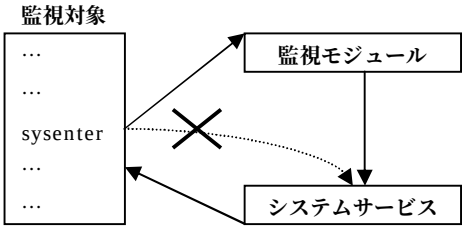

図３　監視モジュール

ユーザーレベルで監視を行った場合には、攻撃者による監視モジュールの回避をどうしても防ぎきることができないと思われる。一方この手法では（カーネルが攻撃されるような事がない限り）、監視モジュールを回避する事は不可能である。

4. 現状と今後の予定

現在は、監視モジュールにおいてシステムサービス ID、プロセス ID、スレッド ID、コールスタック情報の取得ができることを確認した。今後は、仮想スタックリスト、仮想パス、ひいてはその学習を実装し、学習の収束性、異常検知の false positive の頻度、mimicry attack への耐性、システムのオーバーヘッドの評価を行う。

参 考 文 献

[1] S. Forrest, S. A. Hofmeyr, A. Somayaji, T. A. Longstaff, “A sense of self for Unix processes”, IEEE Symposium on Security and Privacy, pp. 120－128, May 1996.

[2] R. Sekar, M. Bendre, D. Dhurjati, P. Bollineni, “A Fast Automaton-Based Method for Detecting Anomalous Program Behaviors”, IEEE Symposium on Security and Privacy, pp. 144－155, May 2001.

[3] H. H. Feng, O. M. Kolesnikov, P. Fogla, W. Lee, W. Gong, “Anomaly Detection Using Call Stack Information”, IEEE Symposium on Security and Privacy, pp. 62－75, May 2003.

[4] 島本大輔, 大山恵弘, 米澤明憲, “System Service 監視による Windows 向け異常検知システム機構”, 情報処理学会論文誌 コンピューティングシステム, Vol.47, No.SIG 12(ACS 15), pp. 420－429, September 2006.