異常検知システム　ウイルス対策　アプリケーション監視

Collaboration Projects

移動端末向けアプリケーション挙動異常検知技術

移動端末におけるウイルスの脅威に備え，移動端末を模擬した評価ボード上に異常検知システムを試作し，評価を行った．なお，本研究は筑波大学大学院 システム情報工学研究科 加藤研究室（加藤 和彦教授）との共同研究により実施した．

先進技術研究所　池部 優佳（いけべ ゆか）　中山 雄大（なかやま たけひろ）
片桐 雅二（かたぎり まさじ）

1. まえがき

2004年6月，移動端末に対する初めてのウイルスが発見された．これ以降ウイルスの種類は年々増加し，現在その数は数百にのぼる．現状ウイルスが発見されているプラットフォーム[*1]は限定されており，日本ではあまり問題視されていないが，今後脅威となる可能性があり，効果的な対策を打つことの重要性が増してきている．ウイルス対策技術として現在主流となっているのがウイルススキャン[*2]である．ウイルスが新しく作成された場合，そのウイルスは完成時点では「未知」，セキュリティベンダに知られると「既知」であると定義すると，ウイルススキャンでは，「既知」のウイルスは効率よく検知できるが，「未知」のウイルスを検知することが難しい．「未知」から「既知」となり，さらに対応するパターンを記録したパターンファイルが作成され，配布されるまでの間，未知ウイルスによる被害者の数はとどまることなく増加する．今後ウイルスの数が増え，さらにウイルスが悪質化するほどこの期間を短くすることが重要になっていくと考えられる．そこでまず，改善効果が大きいと考えられる「未知」から「既知」への変化にかかる時間を短縮する対策に注目した．

通常，セキュリティベンダは，ハニーポット[*3]の利用やウェブサイトの調査などにより未知ウイルスに関する情報を入手しているが，これらと併せて，移動端末上で発生した異常な振舞いに関する情報を得ることにより，情報入手までの時間を短縮することができると考えられる．そこで，本研究では異常な振舞いの発生を検知することができる異常検知技術を検討することとした．異常検知技術の性能指標にはオーバヘッドおよび精度があるが，これらはトレードオフの関係にあるため，用途や利用環境に適したバランスのよい技術を検討する必要がある．本研究では，異常検知システムを移動端末で動作させることを目的としているため，計算機資源の少ない環境でも動作しなければならないという制約があり，この制約を満たしつつ精度を保つ必要がある．また，精度の指標は誤検知と見逃しに分類され，これらにもトレードオフの関係がある．そのため，前述の異常検知システムの用途を加味し，見逃しを減らすことを優先し，誤検知は検知通知を受けるセキュリティベンダが対応可能なレベルを許容することとした．異常検知技術はすでに多く提案されているが，これらすべての条件を満たす技術はこれまで提案されていない．そこで，本研究においてこれらの条件を考慮した異常検知システムを提案した．また，このシステムを移動端末と同等の性能を有する評価ボード上に実装し，精度とオーバヘッドを測定することにより，異常検知システムの移動端末上での実現可能性を初めて示した．なお本研究は，筑波大学 加藤研究室と共同で検討を行った．

2. 異常検知システム

異常検知技術とは，あらかじめ定

＊1　**プラットフォーム**：アプリケーションを動作させるためのOSや動作環境．
＊2　**ウイルススキャン**：各ウイルスの特徴をパターンとして記録し，これにマッチするものがないかを調べるウイルス対策の手法．
＊3　**ハニーポット**：攻撃を発見することを目的に，意図的に用意された攻撃されやすい場所．

義された正常なイベントとかい離するイベントを異常と判断する技術である．また，ウイルスは攻撃者の意図する悪意がある動作を行うものであるため，ウイルスに感染したアプリケーションは通常とは異なる挙動を示すと考えられる．そこで，本システムではアプリケーションの挙動をイベントとして利用する手法を採用した．本手法ではまず，ウイルスの感染を防ぎたいアプリケーションを監視アプリケーションとして選択し，各アプリケーションに対してモデル化，監視を行う．

2.1 モデル化

モデル化では，監視対象アプリケーションの挙動を取得し，アプリケーションの正常挙動を表すモデルを生成する．挙動の取得には，実際の挙動を学習する方法とソースコード解析を行う方法がある．前者では学習不足により，異常がないにもかかわらず異常があるとみなしてしまう誤検知が発生する可能性があるが，後者では動作条件に依存して起こり得るか起こり得ないかが決定する挙動を正しく判断できないことにより，異常があるにもかかわらず異常がないとみなしてしまう見逃しが発生する可能性がある．本システムでは見逃しを優先的に低減することを条件としているため，前者の方法を採用した．

前述のように，学習によるモデル化を利用する際には誤検知について検討する必要がある．基本的な異常検知原理では，モデルに含まれるイベントのみ正常とみなし，それ以外のイベントをすべて異常とみなすが，この場合，モデル化時に網羅的に学習が行えないと，誤検知が発生する．Xを起こり得るすべてのイベント，Yを正常イベント，Mをモデルに含まれる正常イベントの集合とすると，$Y=M$となるのが理想であるが，学習が足りない場合$Y \supset M$となる．監視時に，Mの挙動が得られた場合および$X \cap \overline{Y}$の挙動が得られた場合はそれぞれ正しく正常，異常と判断されるが，$Y \cap \overline{M}$の挙動は正常であるにもかかわらず，異常と判断されてしまう．時刻表示のような単純なアプリケーションではすべての正常イベントを容易に学習することが可能であるため$Y=M$となり誤検知の問題は発生しないが，エディタ，ブラウザのように入力や操作が複雑なアプリケーションでは，すべての正常イベントを学習することは極めて困難であるため$Y \supset M$となり，誤検知の問題が無視できない．本システムではこのような原因により発生する誤検知率の低減を試みた．

一般的に，アプリケーションはある特定の目的で利用されるため，同一アプリケーション内の挙動の傾向は類似している．つまり，未学習であっても正常である挙動（$Y \cap \overline{M}$）は，学習時に得られた正常挙動（M）と類似した特徴を有している可能性が高いと考えられる．一方，ウイルスにより攻撃された場合の異常な挙動（$X \cap \overline{Y}$）は，アプリケーション本来の目的とは異なる，データを盗む，システムを破壊するなどの攻撃者の意図する動作をしているため，学習時に得られた挙動とは異なる特徴を有すると考えられる．そこで，学習時に得られた正常挙動を統計的にモデル化することにより検査挙動とモデルとのかい離度合いを数値化し，この数値を基に異常を検知するアプローチをとった．

一般的に，ウイルスが攻撃者の意図する動作をする際にはシステムコール[*4]を発行する可能性が高い．これは，攻撃により大きな被害をおよぼすためにはファイル操作など，OSの機能を利用する必要があり，このためには多くの場合システムコールが利用されるからである．そのため，これまでにシステムコールの発行を挙動として監視する技術が多く提案されている．しかし，この手法では検知できない攻撃手法が発見された[1]．そこで本システムでは，挙動としてシステムコールとリターンアドレス[*5]を利用する手法を用い，見逃し低減を試みた．

挙動は，システムコールとそのシステムコールが発行されるときにコールスタック[*6]に積まれたリターンアドレスの種類を利用し，I_t（tは任意の整数）のように定義される．

$$I_t = \{Sys, m_a, m_{a-1}, \cdots, m_1, m_0\} \quad (1)$$

ここで，Sysはシステムコール番号，m_xはリターンアドレスを表す．また，xはコールスタック上での位置（ポジション）を表し，頂上を0とし，頂上から下がるにつれて1ずつ加算し，最下点をaとする．

＊4 **システムコール**：アプリケーションがOSの機能を呼び出すための機構．

＊5 **リターンアドレス**：関数の戻り番地．ある関数の実行が終了したとき，次に実行すべき命令が書かれた番地．

＊6 **コールスタック**：アプリケーション実行中に呼び出す関数に関する情報を記録するメモリ領域．

モデルの作成では，得られた挙動に含まれるすべてのリターンアドレスに対し，システムコール番号 *Sys* およびポジション *x* を特徴量として分類し，グループ化する．つまりモデルは，あるシステムコールが発行されたときのあるポジションに積まれ得るリターンアドレスの種類が分かる構成となっている．すべての挙動に関してグルーピングが完了すると，各グループに対して監視時に利用するスコアが計算される．計算式は(2)のとおりである．

$$Score = \frac{1}{N \times 2^{x}} \quad (2)$$

ここで *N* は各グループに含まれるリターンアドレスの種類数を意味する．モデル化例を**図1**に示す．

2.2 監視

監視では，監視対象アプリケーションの挙動を取得し，この挙動とモデルを比較することにより検査を行い，異常度合いを示す異常スコアを算出する．監視例を**図2**に示す．検査では，挙動として得られたリターンアドレスが，モデルの該当するグループに含まれているかを調べ，含まれていない場合はエラーとなり，該当グループのスコアが異常スコアに加算される．該当するグループとは，システムコールおよびポジションが同じグループを意味する．

この方法では，モデルに含まれないものはすべてスコアが加算されることになるが，モデルからのかい離度に即したスコアが与えられるように設定されている．スコアは，*N* が小さく *x* が小さい場合に大きくなり，かい離度が大きいと判定される．*N* が小さいということは，グループに含まれる要素数が少ないことを意味し，学習時に発生したリターンアドレス以外が発生する可能性は低いと考えられる．そのため，*N* が小さいグループでエラーが起きた場合は，モデルから大きくかい離していると考えられる．また *x* が小さいということは，位置が頂上に近いことを意味している．システムコールとの相関はコールスタック上位のリターンアドレスほど強くなるため，*x* の値が小さいグループでエラーが起きた場合は，モデルから大きくかい離していると考えられる．これにより，学習はしていないが，正常である挙動と異常な挙動をスコアにより判別することが可能となる．

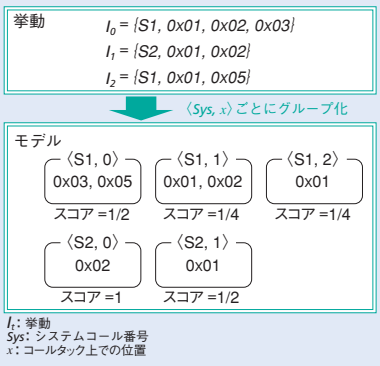

図1　モデル化例

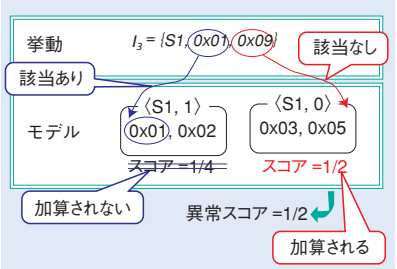

図2　監視例

3. 評価

提案した異常検知システムを評価ボード（Armadillo®*7-500）上に実装した（**写真1**）．CPUは移動端末で多く利用されているARM™アーキテクチャ*8（ARM1136JF-S, 400MHz）を搭載している．また，OSにはLinux 2.6を用いた．次に，実装したシステム上で実際のアプリケーションを用いて性能評価を行った．評価した性能は，オーバヘッドと精度である．

3.1 オーバヘッドの性能評価

評価には，アプリケーションの開始と終了が明確に測定可能なアプリケーションであるa2ps*9，scp*10，tnftp*11，vilistextum*12の4種類用いた．オーバヘッドは監視をしながらアプリケーションを実行した時間（T_{ads}）とアプリケーション単体を実行した時間（T_{app}）を用いて式(3)で求められる．

$$Overhead[\%] = \frac{T_{ads} - T_{app}}{T_{app}} \times 100 \quad (3)$$

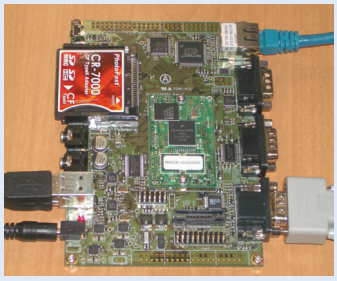

写真1　評価ボード

＊7 **Armadillo®**：㈱アットマークテクノ製の組込機器で，同社の登録商標．
＊8 **ARM™アーキテクチャ**：低消費電力を特長とする組込機器向けのCPUアーキテクチャ．ARM™はARM Ltd.の商標．
＊9 **a2ps**：文書ファイルをPostScript形式の画像ファイルに変換するオープンソースアプリケーション．
＊10 **scp**：暗号化によりホスト間でファイルを安全にコピーするオープンソースアプリケーション．
＊11 **tnftp**：サーバとファイルやディレクトリをやり取りするオープンソースアプリケーション．
＊12 **vilistextum**：HTMLファイルをテキストファイルに変換するオープンソースアプリケーション．

T_{app}は同じアプリケーションであっても変動し，これに伴いオーバヘッドも変化する．そこで，本実験ではT_{app}を変化させる要因として入力データサイズを選び，1kB, 10kB, 100kB, 1MB, 10MBのファイルを利用し測定を行った．測定結果を**表1**に示す．

これより，本実験においては測定した結果の60％以上のオーバヘッドが15％以下となった．その他のオーバヘッドが15％以上と大きくなった原因は，監視のために必要な固定の時間があり，T_{app}が小さいと，この時間が相対的に大きくなるためと考えられる．実際に，表1でオーバヘッドが15％以上になったものはすべてT_{app}が0.32秒以下であった．つまり，本実験においてはT_{app}が大きい場合オーバヘッドが小さく，T_{app}が小さい場合オーバヘッドが大きくなった．後者ではオーバヘッドは大きいが，実行時間は短いため，ユーザに与える不快感は小さいと考えられえる．以上より，本システムのオーバヘッドは実用に耐え得る見込みがあると判断できる．

3.2 精度の性能評価

評価には，ウイルス作成が可能な脆弱性を有するアプリケーションであるa2ps，emacs[*13]，greed[*14]，vilistextumの4種類を用いた．本実験においては異常な挙動をさせるために擬似的にウイルスを作成する必要があるため，まず脆弱性をつくアプリケーションを作成した．モデルは，各アプリケーションに対して1種類の入力を与えることにより得られる挙動を用いて作成した．精度の評価はこれらを利用して，正常な挙動を監視したときの異常スコアと，作成した擬似ウイルスにより起こる異常な挙動を監視したときの異常スコアの値を出力し比較することにより行った．実験の結果得られた異常スコアを**表2**に示す．これよりすべてのアプリケーションにおいて，異常挙動は正常挙動よりも大きい値を有していることが分かる．既存技術であるVtPath[2]は，提案システムと同様にシステムコールとリターンアドレスを利用する異常検知技術であるが，前述の学習したイベントのみ正常とみなす基本的な異常検知原理を用いているため，本実験の学習量ではすべてのアプリケーションに対して誤検知が発生してしまう．一方，提案手法では，正常な挙動と異常な挙動を異常スコアにより判別し，誤検知や見逃しを発生させないことが可能であることが確認できる．

表1　オーバヘッド測定結果（%）

アプリケーション / 入力データサイズ	a2ps	scp	tnftp	vilistextum
1kB	55	8	69	測定不可
10kB	38	11	76	測定不可
100kB	13	12	81	26
1MB	7	3	61	≒0
10MB	3	2	15	≒0

15%以下

表2　出力された異常スコア

アプリケーション / 監視挙動	a2ps	emacs	greed	vilistextum
正常	0.37	0.53	0.017	9.69
異常	132	526	763	454

4. あとがき

ウイルスによる異常挙動を検知可能な異常検知システムを提案し，移動端末と同等の環境に実装することにより，提案システムの有効性を明らかにした．

今後，より実用に近いアプリケーションや，より高度な攻撃手法の擬似ウイルスを利用した評価を行うことにより詳細な評価を実施し，移動端末搭載へ向けて検討を続けていく．

文　献

[1] D. Wagner and P. Soto：“Mimicry attacks on host-based intrusion detection systems,” in Proc. ACM Conference on Computer and Communications Security, pp.255−264, 2002.

[2] H.H.Feng, O. Kolesnikov, P. Fogla, W. Lee, and W. Gong：“Anomaly detection using call stack information,” in Proc. 2003 IEEE Symposium on Security and Privacy, pp.62−75, 2003.

＊13 **emacs**：高機能なテキストエディタ機能を有するオープンソースアプリケーション．

＊14 **greed**：ファイルをダウンロードする機能を有するオープンソースアプリケーション．