2006－MBL－39（6）
2006－ITS－27（6）
2006／11／16

モバイル向け異常検知ソフトウェア

池部　優佳[†1]，中山　雄大[†1]，竹下　敦[†1]，
鈴木　勝博[†2]，阿部　洋丈[†3]，加藤　和彦[†2]
†1（株）NTT ドコモ 総合研究所
†2 筑波大学
†3 科学技術振興機構

我々は携帯電話などのモバイル機器を対象とするソフトウェア異常検知技術の開発に取り組んでいる．本稿では，異常検知技術の中でも，誤検知率が低いとされる相関規則を適用するアプローチに注目した．モバイル機器を用いた実験を通じて，このアプローチによって異常検知が可能であることを確認した．さらに，低速なメモリアクセスを伴うスタックトレースを簡略化することで処理時間を削減する手法を新たに提案し，その効果を確認した．

Anomaly Detection Software for Mobile Terminals

Yuka Ikebe[†1], Takehiro Nakayama[†1], Atsushi Takeshita[†1],
Katsuhiro Suzuki[†2], Hirotake Abe[†3], and Kazuhiko Kato[†2]
†1 Research Labs, NTT DoCoMo, Inc.
†2 Tsukuba University
†3 Japan Science and Technology Agency

We have developed an anomaly detection technique for mobile terminals. In this paper, we adopt the approach that uses the association rule. From the result of the experiment, we clarified that this approach can detect an anomaly behavior. Moreover, we propose methods for reducing overhead by simplifying the stuck trace and confirm their efficiency.

1.　はじめに

PC やワークステーション，サーバ，ルータ，携帯電話，PDA など, すべての計算機は外部もしくは内部からの攻撃にさらされる可能性がある．特に，携帯電話は，緊急通報に使われるなど，ライフラインとして欠かせない計算機であり，PC よりも高度な安全性が求められている．

現在 PC で主流となっている攻撃は，計算機上で実行されているソフトウェアの脆弱性を踏み台にしたものである．攻撃者はソフトウェアの脆弱性を利用した悪意のある実行コード（ウィルスやワームなど）を計算機に送り込み，実行中のプロセスの制御を奪い，当該プロセスの権限を利用して不正操作をおこなう．ソフトウェアの脆弱性を利用した攻撃への対策として， 我々は，異常検知システム(Anomaly detection)と呼ばれる手法に着目する. Anomaly detection はソフトウェアの正常な動作をモデル化し，ソフトウェアの実行がそのモデルに従っているかを監視することによって異常を検知する手法であり，未知の攻撃・異常の検知を目指している．

金野らの研究では，Anomaly detection の一つとして，ソフトウェアの発行するシステムコールとその時点でプロセススタックに積まれたリターンアドレスとの共起関係に着目し，正常な動作を相関規則を用いてモデル化する方法を提案している．また，提案手法を，モバイル機器に搭載し運用に耐えうるシステムを構成するためのオーバヘッド削減方法も提案している．これは時間オーバヘッドの大きなリターンアドレスの取得を簡略化することにより処理時間の低減を試み

るというものである[2]．また，鈴木らはより精度の高い簡略化方法を提案している[3].

本稿では，金野らの相関規則を用いた Anomaly detection の有効性を明らかにするために，組み込み機器上に，モデル化および監視アルゴリズムを実装した．さらに，比較対象として，従来手法として広く知られている VtPath[5]を実装し，それぞれの精度の評価を行った．さらに，オーバヘッドを削減するための方法を，新たに提案した．実験により，相関規則を用いた Anomaly detection は精度の点で VtPath より優れており，また提案したオーバヘッド削減手法は処理時間低減に有効であることを確認した.

2. 既存の異常検知システム

現在用いられている異常検知システムには以下に示す二つの類型がある．ここではそれぞれの概要，問題点を説明する.

2.1. ソース解析に基づく異常検知システム

ソース解析に基づく異常検知システムは，ソースコードを静的解析することによってソフトウェアの正常な動作のモデルを作成し，ソフトウェアの実行系列がこのモデルに受理されるどうかを検査することによって，正常動作からの乖離すなわち異常動作を検知するものである[1][5][9][11]．モデル化された動作に関しては誤検知率がゼロであるという利点がある一方，通常の実行では起こり得ない挙動(Impossible Path[7][11]）をモデルが受理することによって攻撃見逃しが起きる可能性がある.

2.2. 振舞い解析に基づく異常検知システム

振舞い解析に基づく異常検知システムは，監視対象ソフトウェアの過去の動作の学習によって正常な動作を判別する規則を生成し，その規則に従って異常を検査するものである[2][6][7][13]．実際にソフトウェアを動かしたことによって得られたデータを基にモデル化を行うため，学習が十分であるという仮定ならば，ソフトウェアの正常動作を完全にモデル化することが可能であり，かつ Impossible Path の問題もない．このことから，本研究では振舞い解析に基づく異常検知システムに注目している.

Forrest らが提案するシステム[6]では，監視対象ソフトウェアが実行中に発行したシステムコールを捕らえ，システムコールのペアの相関を正常パターンとしてデータベースに保存しておき，監視時に正常パターンとの逸脱を検知する.

しかし，システムコール発行のモデル化は，ソフトウェアの動作の一部をモデル化しているにすぎず，攻撃者はモデルに受理されるようにシステムコールを巧みに発行することが出来てしまう（偽装攻撃[7][12]）ため，脆弱である.

そこで，システムコール発行パターンに加え，システムコール発行時の他のパラメータをモデル化することで，偽装攻撃を困難にする手法が提案されている[2][5][7][10].

Feng らが提案するシステム[5]は，システムコールの正当性を検証するために，スタック情報（スタックに積まれたリターンアドレスの列）を利用する．プログラム実行中にシステムコール発生時のスタック情報を取得し，システムコール発生時のプログラムカウンタとともに記録した Virtual Stack List を生成し，また，現在の Virtual Stack List と一つ前の Virtual Stack List との差分情報，すなわち比較対象のスタック情報のスタック最下位より比較検証を順次行い異なるリターンアドレスを検出してから以降のリターンアドレスの列（Virtual Path）を生成する（Fig. 1 参照)．生成された Virtual Stack List と Virtual Path からそれぞれハッシュテーブルを生成し，そのテーブルをソフトウェアのモデルとして利用する．ソフトウェアの動作を検証する際には，ソフトウェア実行中に，Virtual Stack List と Virtual Path を生成し，モデルであるハッシュテーブルとのマッチングを行い，合致していればシステムコール要求に対し許可をし，合致しなければ，異常であると判定する.

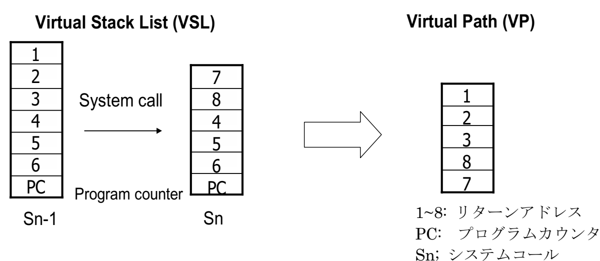

Fig. 1 Virtual Path 生成

しかし，一般的に，すべての正常動作を試行することは不可能である．そのため，未学習の正常動作は異常動作と判断せざるを得ず，誤検知の原因となる．また，Feng らのシステムでは，システムコール発行時に，スタック情報を取得するが，スタック領域は通常メモリに用意され，スタックフレームにはリターンアドレス以外にも引数などの情報が含まれることから，スタックからリターンアドレスを取得するに

は低速なメモリアクセスを頻繁に行わなければならず時間オーバヘッドが高い．

金野らが提案する手法[2]では，未学習動作に対する誤検知を改善するために，Feng らの手法を拡張し，ソフトウェアの発行するシステムコールとその時点でスタックに積まれたリターンアドレスとの共起関係に着目し，監視対象ソフトウェアの過去の動作（すなわち，システムコールとリターンアドレスの発行パターン）の学習によって正常な動作をモデル化する．正常動作を一意に特定してしまう Feng らの手法とは違い，正常動作を統計的にモデル化するアプローチを取る．これにより，未学習の正常動作も正しく正常と判定することが期待できる．統計量としては，共起関係を表す統計モデルである相関規則を採用している．また，監視を行う際には，ソフトウェア実行中に Virtual Path を生成し，当該 Virtual Path 上のリターンアドレス，システムコール識別子の共起関係が，モデルにマッチしているかを判定する．マッチしなかった場合は異常動作とみなし，Anomaly Score という値加算する．Anomaly Score とは，異常の度合いを示す値であり，Zero-frequency という確率値の逆数として定義している[14].

また，オーバヘッドを低減するために，モデル化の際に，スタックに出現するリターンアドレスの数をカウントし，カウント数が多いものを終端情報として定義し，監視時に終端情報を見つけたらスタックトレースを中断し次のスタックに移動するという手法を採用する．

3. 提案手法

金野らは相関規則を用いた Anomaly detection のオーバヘッドを低減する手法を提案している．この手法は，モデル化の際に，出現するリターンアドレスの数をカウントし，カウント数が多いものを終端情報として定義し，監視時に終端情報を見つけたら現在のスタックにおけるリターンアドレスの取得を中断し次のスタックに移動するという方法である．この方法はオーバヘッドの低減は可能であるが，終端情報でスタックトレースを中断したことにより，正しい Virtual Path が取れなくなるエラーが発生する可能性が高い．そこで鈴木らは，エラーの低減を抑える終端情報の利用方法を提案し，「見つけたら即中断」ではなく，「連続した位置に出現したら中断」と改良している．これらにより，オーバヘッドの低減，エラーの減少が実現されてきたが，モバイル機器に載せるためには更なるオーバヘッドの低減が必要である．

3.1. スタックトレースの簡略化

監視における処理時間を軽減するために，提案手法は，精度を保ちつつ，低速なメモリアクセスを伴うスタックトレースをより多く簡略化する．提案手法において，Virtual Path はモデル化，監視両過程において生成される．そこで，モデル化時に生成した Virtual Path の情報から，監視時のスタックトレースを簡略化する指針を得る．ソフトウェア実行において，現在のスタック情報と一つ前のスタック情報を比較すると，関数の中で関数を呼び出す際や，ループの状況下では，前後のスタックでリターンアドレスとして等しい情報を持ち，差分とならないものが多数存在することが考えられる．すなわち，ソフトウェア監視時にこのリターンアドレスをすべて取得することは冗長である．そこで提案手法は，モデル化時に，スタックトレースを中断するべき箇所であるリターンアドレス（終端情報）を抽出し，監視の際にスタックトレースにおいて終端情報を見つけたらスタックトレースを中断し，次のスタックに移動する．本稿では，精度を落とさないために本来の Virtual Path を変化させないことを考慮しつつ，オーバヘッドをできる限り削減できるよう，従来の終端情報の定義の他，以下の 5 種類を定義した．

・従来の終端情報(1)

スタック情報に出現するリターンアドレスの出現頻度をリターンアドレスごとにカウントし，降順にリスト化する．このリストの上位のリターンアドレスから終端情報として定義する．

・境界から選択した終端情報(2)

Virtual Path 生成時のスタック比較の際に，一致部分と不一致部分の境界をつくる，一致部分の最上位のリターンアドレス（終端情報候補）を全てのシステムコールについてリスト化し（Fig. 2(a)），それぞれのリターンアドレスのリスト内の出現頻度をカウントして降順にリスト化し，終端情報リストを作成する（Fig. 2(b)）．このリストの上位のリターンアドレスから終端情報として使用する．この定義では本来中断する

べきリターンアドレスを選択肢とするため，不必要な境界以下のスタックをトレースしてしまうこと防ぐことができると考えられる．

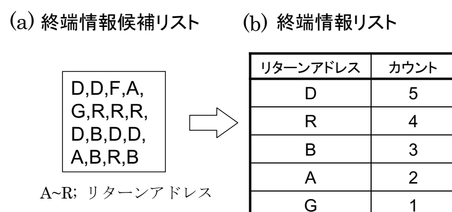

Fig. 2　終端情報リスト

・出現位置を考慮した終端情報(3)
終端情報の出現位置（スタック底からの位置）を記録し，リターンアドレスごとにその位置の和を計算する（カウント）．この値の降順にリスト化して終端情報リストを作成し（Fig. 3），このリストの上位のリターンアドレスから終端情報として定義する．この定義では，スタックトレースをよりたくさん省略可能な位置の高いリターンアドレス，中断に使われる確率が高い出現回数が多いリターンアドレスが優先的に定義されるため，効率よく省略できると考えられる．

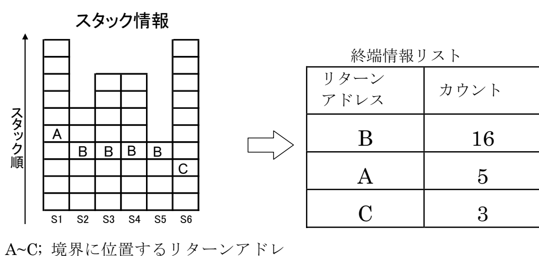

Fig. 3 終端情報リスト

・境界以下の出現位置を考慮した終端情報(4)
境界より下位に出現するリターンアドレスの出現位置を境界からの距離の逆数で記録する．リターンアドレスごとにその位置の和を計算し，この値の降順にリスト化して終端情報リストを作成し，このリストの上位のリターンアドレスから終端情報として定義する．この定義ではスタック上位で中断することにより本来必要とされるVirtual Pathがとれなくなる危険性を避けつつ，効率的にスタックトレースを省略することができると考えられる．

・境界以下の出現位置を効率的に考慮した終端情報(5)
終端より下位に出現するリターンアドレスの出現位置（終端より下位を終端からの距離の逆数で表現）をスタックごとに記録し表を作成する．リターンアドレスごとに総和を計算し，値が最大のものを終端情報リストに記録し，表中の選ばれたリターンアドレスの行を削除し，総和を再計算する．これを繰り返すことにより終端情報リストを作成し，リストの上位のリターンアドレスから終端情報として定義する．この定義では，すでに定義された終端情報の出現位置より下位に出現するリターンアドレスは，終端情報として定義しても意味がないという考えに基づいており，より有効なリターンアドレスが終端情報として定義されると考えられる．

・連続出現期間を考慮した終端情報(6)
同一リターンアドレスがスタックに同位置に連続して出現する期間をカウントし，この値の降順にリスト化して終端情報リストを作成し，このリストの上位のリターンアドレスから終端情報として定義する．この定義では，連続しているリターンアドレスを選択するため，本来中断すべきでないところで中断することを防ぐことができると考えられる．

4.　実験

4.1. 実験概要

4.1.1.　　精度測定

金野らの提案する相関規則を用いたAnomaly detection手法および従来のAnomaly detection手法であるVtPathの精度を測定するために，以下の手順で実験を行った．監視対象プログラムは携帯電話で動作させる可能性のあるアプリケーションであり，かつバッファオーバーフローを埋め込むことができる脆弱性が存在するアプリケーションとして，PDFビュアー（xpdf）を採用した．本実験は，Linuxを搭載することができる組み込み機器としてアットマークテクノ社製のArmadillo-9を用いた．Armadillo-9のスペックをTable1に示す．

Table1 Armadillo-9 のスペック

プロセッサ	EP9315 (Cirrus Logic社製)
CPUコア	ARM920T (ARM9TDMI + 16kB(I)/16kB(D)Cache + MMU + Write Buffer)
システムクロック	CPUコアクロック: 200MHz BUSクロック: 100MHz
メモリ	SDRAM 64MB / FLASH 8MB

1. Armadillo-9 上で，xpdf を動作させ，4 種類の PDF ファイルを開いたときのシステムコール及びスタックをトレースした（使用したファイル名称および正常／異常を Table 2 に示す）
2. 1 で取得されたスタックを用いてモデルを作成した．
3. 2 で得られたモデルを使い，map，attack の PDF ファイルを開いたときのプログラムの動作の監視を行った．
4. 2,3 を相関規則を用いた手法，および VtPath で行った．

Table 2　ファイルの種類

ファイル名	paper	report	map	attack
正常 / 異常	正常	正常	正常	異常
使用用途	モデル化		監視	

4.1.2.　処理時間測定

精度を落とすことなく，終端情報を用いることによってどの程度処理時間の低減ができるのかを以下の手順で調べた．終端情報による処理時間の削減割合を調べることを目的とし，ここではランタイムで監視をすることが可能な ls を採用した．

1. Armadillo-9 上で，監視対象プログラムとして ls を動作させ，システムコールおよびスタックをトレースした．
2. 1 で得られたスタック情報を元に終端情報を使わずモデルを作成した．
3. 再び ls を動作させ，1 で得られたモデルを用いて監視を行い，処理時間を測定した．
4. 1 で得られたスタック情報から終端情報を設定した．
5. 1 で得られたスタック情報から終端情報を用いてモデルを作成した．
6. 再び ls を動作させ，5 で得られたモデルおよび終端情報を用いて監視を行い，処理時間を測定した．

ここで，終端情報の利用方法は，「定義された終端情報がスタック底から見て同じ位置に連続して出現したらスタックトレースを中断する」とした．

4.2. 実験結果

4.2.1.　精度測定

それぞれの手法による Anomaly Score の値を Table 3 に示す．（Anomaly Score は全システムコール数で除算する）

Table 3　Anomaly Score

	相関規則	VtPath
map	0.03	0.52
attack	1.05	0.18

4.2.2.　処理時間測定

終端情報を使用したときと使用しない場合の処理時間を各終端情報の 3.1 に記載のそれぞれの定義の仕方で比較した (Fig. 4,Fig. 5)．その際，定義する終端情報の個数は 5 個，定義できる最大の個数の 2 種類で行った．

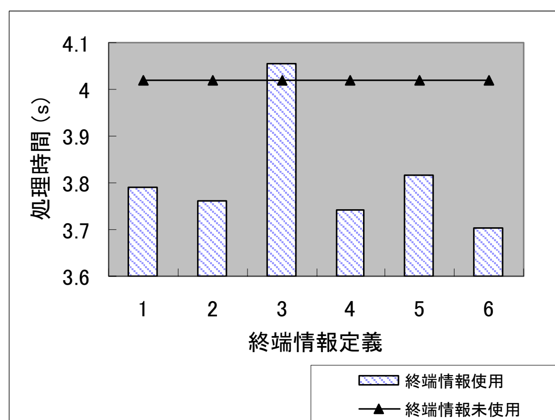

Fig. 4　終端情報を用いた効果(5 個)

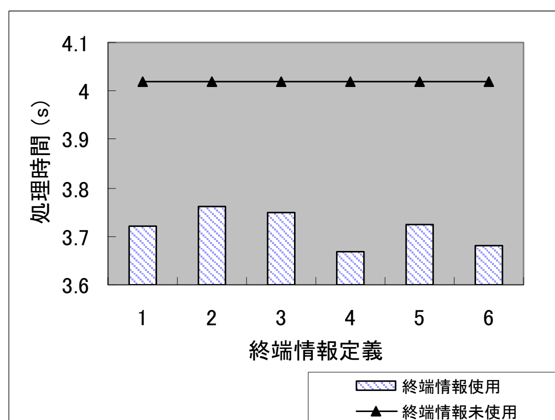

Fig. 5　終端情報を用いた効果(最大数)

5. 考察

● 精度

Table 3 より, 提案手法では正常動作である map の Anomaly Score は小さく, 異常の動作である attack の Anomaly Score は大きくなっており，正常動作と異常動作を見分けることが可能であることが分かる．一方，従来手法の VtPath では，正常動作より異常動作の Anomaly Score の方が大きくなっており，攻撃見逃し，誤検知が起こる可能性が高いことがわかる．

これは，VtPath ではモデル化の際に出てきた動作のみを正常と判断するため，未知の正常動作を異常と判断してしまうことが原因である．これがいわゆる誤検知となる．今回の実験では paper と report でモデルを作成したが, 監視対象とした map ではこれらに含まれていなかった動作が多く含まれていたと考えられる．paper や report は文字が中心のファイルであるのに対し，map では図が中心であるため，表示する際の動作が異なるためである．また，map と attack は双方とも未知の動作であるが, map の方が値が大きくなっている．これより map は attack よりもモデル化の際に含まれていなかった動作が多く含まれていることが分かる．これに対し，提案手法では， map より attack の方が Anomaly Score が大きくなっている．VtPath の結果の議論から，map は attack よりモデルにない動作が多く存在することが分かったが，それにもかかわらず，attack の方が Anomaly Score が小さくなっている．これより，相関規則により，モデル化の際に出てきた動作から，それ以外の正常動作を推測できているということが分かる．つまり，未知の動作の中でも，正常な可能性の高い動作と異常な可能性が高い動作を見分けることができていると解釈できる．このため，未知の正常動作でも異常とみなさず，誤検知を低減することができると考えられる．

● 処理速度

Fig. 4 の結果を踏まえ，それぞれの定義と結果について考察する．

➢ 境界から選択した終端情報

この定義では本来中断するべきリターンアドレスを選択肢とし，不必要な境界以下のものをトレースしてしまうこと防ぐことができため，Fig. 4 のように，従来の定義(1)より処理時間が短縮できている．

しかし，定義される個数が限定される場合，境界のリターンアドレスが定義されていないスタックでは，境界以下で中断される可能性が少なく，スタック底までトレースしてしまうといった欠点もあると考えられる．

➢ 出現位置を考慮した終端情報

この定義では，スタックトレースをよりたくさん省略できる位置の高いリターンアドレス，中断に使われる確率が高い出現回数が多いリターンアドレスが優先的に定義される．

しかし，今回の終端情報の利用方法では，定義された終端情報が連続して同位置に出現していないとスタックトレースの中断は行われない．高い位置に出現するリターンアドレスは，低い位置に出現するリターンアドレスよりも，リターンによりスタックからなくなる可能性が高く，連続して出現する割合も低くなる．その場合スタックトレースを省略するためのものとして有効に機能しない．そのため省略できる時間が少なく，また終端情報とのマッチングの時間がプラスされるので，Fig. 4 のように，終端情報を利用しない場合と比較してわずかに増加した処理時間となってしまったと考えられる．

➢ 境界以下の出現位置を考慮した終端情報

「出現位置を考慮した終端情報」では，出現位置に関わらず終端情報の候補としたが，あまり高い位置に出現するリターンアドレスを終端情報とすることは，境界より上位で中断してしまう可能性を高くすると考えられる．その場合，本来必要とされる Virtual Path がとれなくなる．

この定義では，このような危険性を避けつつ，効率的にスタックトレースを省略することを重視し，境界以下のリターンアドレスを候補としている．これにより，スタック中位で連続的に出現するものが定義され，スタック底までトレースしてしまうことを避けられているため，Fig. 4 のように，処理時間が短縮できているこ

とが分かる.

- 境界以下の出現位置を効率的に考慮した終端情報

この定義では，すでに定義された終端情報の出現位置より下位に出現するリターンアドレスは，終端情報として定義しても意味がないという考えに基づいている.すでに中断されていたら，それより下位で中断することはないためである．動作は,「境界以下の出現位置を考慮した終端情報」で終端情報が定義されるごとに，それより下位のスタック情報を削除することにより実現する.

しかし，Fig. 4 に示すように,「境界以下の出現位置を考慮した終端情報」(4)より処理時間が多くなっている.終端情報として定義されたリターンアドレスを RA とすると，RA より下位のリターンアドレスの情報は，RA で中断されるスタックにおいては意味のない情報であるが, RA を持たないスタックにおいては重要な情報となりうる．この情報を省いてしまったことによって効率的な終端情報の定義を妨げてしまったと考えられる.

- 連続出現期間を考慮した終端情報

この定義では，スタックに長期間存在したリターンアドレスが優先して定義される．そのため，プログラム初期につまれ，後期までリターンされない，スタック下位に出現するリターンアドレスが終端情報として定義される．Fig. 4 より，非常に効率的に処理時間が削減できている．終端情報によりスタックトレースを中断する際には，同位置に連続して同じ終端情報が出現した場合という条件がある．この定義により選ばれた終端情報はもともと連続して出現しているため，この条件を満たしやすく，他の定義よりも大きな処理時間の短縮が実現できていると考えられる.

以上の検討より，終端情報の定義は「連続出現期間を考慮した終端情報」が最も有効であると考えられる.

また，Fig. 5 では Fig. 4 に比べてどの定義でも処理時間の短縮が実現できている．終端情報の個数の増加によりマッチングコストが増加することが懸念されたが, それにも増して，省略時間が大きいことが分かる．これより，終端情報をたくさん用いた方が，よいことが分かる．しかし，Fig. 5 の監視では, 0 であるはずの Anomaly Score が 4~6 とあがってしまった．スタックトレースの大きな省略は精度の低下につながるため，今後精度と処理時間の関係を考慮しながら，適切な終端情報の個数を検討する必要がある.

6. まとめ

本稿では，ソフトウェアの動作をランタイムで監視する研究を行う上で，監視対象ソフトウェアが過去に動作した際に発行したシステムコールと，その時点でスタックに積まれたリターンアドレスの共起関係から相関規則を生成することによりモデルを作成する方法に注目した。この手法を携帯電話で用いられる ARM アーキテクチャ上に実装し評価を行った結果、従来手法として知られる VtPath に比べて誤検知が低減できることが示唆された.

また，監視時において，スタックをトレースしリターンアドレスを取得するが、このスタックトレースは低速のメモリアクセスを含むため，オーバヘッドの増大が懸念される.そこで, モデル化時に見出した終端情報を用いることにより，監視時のスタックトレースを簡略化し，オーバヘッドを低減することを試みた．その結果，新たに提案した終端情報を用いることにより，監視にかかる処理時間を低減できることが分かった.

今後，使用するデータの数，種類を増やして検証を行うと共に，更なる処理時間の低減へ向けた検討をしていく.

参考文献

[1] 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 : コンピューティングシステム， Vol. 45, pp. 11-20, 2004.

[2] 金野 晃, 池部 優佳, 中山 雄大, 竹下 敦, 鈴木 勝博，阿部 洋丈，加藤 和彦，" 携帯端末向けソフトウェア異常検知技術", 情報処理学会: システムソフトウェアとオペレーティングシステム，Vol.2006, pp.1-8, 2006.

[3] 鈴木 勝博，阿部 洋丈，加藤 和彦，金野 晃，池部 優佳，中山 雄大，竹下 敦，" スタック探索の簡略化による異常検知システムの高速化", コンピュータセキュリティ, 2006-CSEC-34, pp.183-190, 2006.

[4] R. Agrawal, "Mining Association Rules between Sets of Items in Large Database.", ACM SIGMOD Conference, Vol.22, pp.207-216, 1993.

[5] H. H. Feng, et al, "Anomaly Detection Using Call Stack Information", 2003 IEEE Symposium on Security and Privacy, p.62, 2003.

[6] S. Forrest et al, "Intrusion Detection using Sequences of System Calls", Journal of Computer Security Vol. 6, pp.151-180, 1998.

[7] D.Gao et al, "On Gray-Box Program Tracking for Anomaly Detection," 13th USENIX Security Symposium, pp.103-118, 2004.

[8] J. Han, H. Pei, and Y. Yin. "Mining Frequent Patterns without Candidate Generation.", ACM symposium on applied computing, Vol.29 pp.1-12, 2004.

[9] L. Lam et al, "Automatic Extraction of Accurate Application-Specific Sandboxing Policy," EUROCOM International Symposium on Recent Advances in Intrusion Detection (RAID), Vol.3224, pp.1-20, 2004.

[10] G.C.Necula et al, "Proof-carrying code". 24th ACM SIGPLAN-SIGACT Symposium on Principles and Programming Languages, pp.106-119, 1997.

[11] D. Wagner et al, "Intrusion Detection via Static Analysis", IEEE Symposium on Security and Privacy, p. 156 2001.

[12] D. Wagner et al, "Mimicry Attacks on Host-based Intrusion Detection Systems", 9th ACM Conference on Computer and Communications Security, pp.255-264, 2002.

[13] C. Warrender et al, "Detecting Intrusions Using System Calls: Alternative Data Models", IEEE Symposium on Security and Privacy, pp.133-145, 1999.

[14] I. Witten and T. Bell, "The zero-frequency problem: estimating the probabilities of novel events in adaptive text compression", IEEE Trans. On Information Theory, Vol.37, pp.1085-1094 1991.