Behavior Blocking方式に基づく組み込みデバイスの保護

北澤繁樹，河内清人，米田健，藤井誠司，中川路哲男[1]

近年，ユビキタスネットワークを見据えた情報家電，携帯電話，PDA（Personal Digital Assistance）などの組み込みデバイスは高度な通信機能に加え，情報管理機能やコンテンツ再生機能など多くの機能を有している．しかしながら，組み込みデバイスでは，使用できるリソースに制限があることや組み込みデバイスを対象とした攻撃がこれまで報告されていないこともあり，セキュリティ機能については，これまで，十分な考慮がなされていない．したがって，近い将来，組み込みデバイスが通常のコンピュータと同様にネットワークに接続されるようになると，ウイルスなどの攻撃対象とされてしまう危険性がある．

そこで本論文では，Behavior Blocking 方式に基づいた，組み込みデバイスへの不正アクセス対策方式を提案する．さらに，実装により，提案方式の有効性およびオーバヘッドに関して検証する．

Protection of Embedded Devices with Behavior Blocking

Shigeki KITAZAWA, Kiyoto KAWAUCHI, Takeshi YONEDA,
Seiji FUJII, Tetsuo NAKAKAWAJI[2]

Recently, an embedded device for Ubiquitous computing, such as a networked appliance, a cellular phone, PDA and so on, has not only a communication function but also more advanced functions (e.g. data management, digital-contents processing and etc.). However, since the embedded device lacks resource capacity, it is difficult to assign enough resource to keep secure. Therefore, it will face a threat like a computer virus as well as a personal computer, in a future.

In this paper we discuss a suitable way to protect an embedded device. As a result, we lead a novel idea to prevent from a malicious code execution attack. Then, we implement our idea and examine its performance. Finally, we show that our idea is quite effective and low penalty.

1 はじめに

近年，ユビキタスネットワークを見据えた情報家電，携帯電話，PDA などの組み込みデバイスは高度な通信機能に加え，情報管理機能やコンテンツ再生機能など多くの機能を有している．

組み込みデバイス上で動作する Java 言語の仮想マシンやコンテンツデコーダなどを含めた各種ライブラリ，オペレーティングシステムといった基本機能は処理速度や処理内容の理由からマシン語で実装されている．こうした，マシン語で実装されている個所へは通常のコンピュータと同様な手法によって攻撃可能である．例えば，コンテンツデータ内にマシン語で記述された不正なコードが含まれていた場合は，ライブラリのバグによりそのコードが実行されてしまうことがある [1]．しかしながら，組み込みデバイスでは，使用できるリソースに制限があることや組み込みデバイスを対象とした攻撃がこれまで報告されていないこともあり，セキュリティ機能については，これまで，十分な考慮がなされていない．したがって，近い将来，組み込みデバイスが通常のコンピュータと同様にネットワークに接続されるようになると，ウイルスなどの攻撃対象とされてしまう危険性がある [3]．

そこで，本論文では，保護対象，保守性，制限の観点から，不正コードを送りつけ，実行するような攻撃に効果がある，パターンマッチング方式と Behavior Blocking 方式 [2] を比較し，組み込みデバイス上の攻撃検知では Behavior Blocking 方式が適していることを示す．さらに，Behavior Blocking 方式に基づいた，組み込みデバイスのための不正アクセス防止方式を提案する．加えて，攻撃の検知後も安定した継

[1]三菱電機株式会社 情報技術総合研究所
[2]Mitsubishi Electric Corporation,
Information Technology R&D Center

続動作を可能とする仕組みも提案する．

本論文の構成について，まず 2 節では既存の攻撃対策技術として，パターンマッチング方式および Behavior Blocking 方式について触れ，それぞれの方式が持つ長所・短所について述べる．次に 3 節において両方式を組み込みデバイスへの実装の観点から比較し，解決すべき問題点について議論する．4 節で，本論文における提案方式の概要を述べた後，5 節で実験により提案方式の有効性を示す．最後に 6 節で本論文をまとめる．

2 攻撃対策技術

ここでは，既存の攻撃対策技術である，パターンマッチング方式と Behavior Blocking 方式について説明する．

2.1 パターンマッチング方式

パターンマッチング方式では，既存の攻撃からその攻撃の特徴を抽出し，その特徴に類似したデータがコンピュータ上に存在した場合，それを検知する．パターンマッチング方式の長所と短所を以下に示す．

長所 ：
- ・高速処理が可能
- ・誤検知が少ない

短所 ：
- ・未知の攻撃を検知できない
- ・デコード処理が必要

デコード処理は，入力データの詳細な検査を行う場合，各データのフォーマットごとに必要となるため，監視プログラムの肥大化や負荷増加の原因となる．

2.2 Behavior Blocking 方式

Behavior Blocking 方式は，プログラムのコンピュータリソースへのアクセスを監視して，そのプログラムの異常な振る舞いを検知・制御する方式である．監視する対象はシステムコール（ディスクアクセス，メモリアクセスなど）となる．Behavior Blocking 方式の長所と短所を以下に示す．

長所 ：
- ・未知の攻撃を検知可能
- ・デコード処理が不要
- ・メンテナンスフリー

短所 ：
- ・システムコールを呼ぶ攻撃のみ監視可能
- ・システムコール実行時の負荷が増加

3 議論

ここでは，どのような方式が組み込みデバイスに適しているのかを保護対象，保守性，制限の観点から議論する．

保護対象： 保護の対象としては，組み込みデバイスに保管されているデータが挙げられる．例えば，携帯電話や PDA 内部には利用者の個人情報が保管されており，情報家電では機器の制御情報などが保管されているため，これら内部のデータの漏洩や改竄は被害が大きい．

保守性： 組み込みデバイスは，ハードウェアに特化して実装され，また，その種類や数が膨大になるため，それら一つ一つに対して十分なメンテナンスを行うことは困難である．したがって，メンテナンスフリーであることは重要な要素となる．

制限： 組み込みデバイスでは使用できるリソースに制限があることから，実装において，実行負荷が軽いこと，および実装サイズが小さいことといった制約が存在する．

これらの観点から，パターンマッチング方式と Behavior Blocking 方式とを比較したものを表 1 に示す．

表 1 の結果から，Behavior Blocking 方式はパターンマッチング方式と比べて，メンテナンスフリーである点，コンテンツデータの監視にデコード処理を伴わない点，および実装サイズを小さくできる点で，組み込みデバイス上での実装に適しているといえる．さらに，Behavior Blocking 方式ではデータなどにアクセスする際のシステムコールも監視可能であるため，内部データの保護も可能である．ただし，Behavior Blocking 方式を用いるにあたり，以下の課題を解決する必要がある．

1. システムコール発行時の負荷の軽減
2. 検知後の処理

2 に関して，Behavior Blocking 方式は，不正コードの実行を検知するため，不正コードの検知時には攻撃者によってメモリの内容などが書き換えられている可能性がある．この場合，プロセスを継続して動作させることが困難であるため，通常は，攻撃を受けたプロセスを停止させるといった対策が行われる．しかしながら，組み込みデバイスでは，電源が続く限り処理が継続していることを前提とする場合が多い．したがって，たとえ攻撃を受けた場合であっても，

表 1: パターンマッチング方式と Behavior Blocking 方式の比較

	内部データの保護	メンテナンス	制限	
			サイズ	負荷
パターンマッチング	可能	必要	大	大
Behavior Blocking	可能	不要	小	制御ルールに依存

処理を継続できるような仕組みを考える必要がある．

以降の節では，Behavior Blocking 方式をベースとし，以上の課題を解決した攻撃検知および対策機能の実装方式を提案する．

4 提案方式

これまでの議論に基づき，ここでは組み込みデバイスのための不正アクセス対策方式を提案する．

4.1 設計指針

提案方式では，Behavior Blocking 方式に基づいて組み込みデバイス上のマシン語で記述されたコードへの攻撃であるバッファオーバフロー攻撃 [4] を防ぐことを目的とする．バッファオーバフロー攻撃は，ネットワーク不正アクセスやウイルスの感染手段として頻繁に使用される攻撃である．

3 節で述べたように，Behavior Blocking 方式を用いた場合，解決すべき課題として，負荷の抑制と被害の復旧がある．そこで，本節では，これらの課題に対する解決策を考察する．

4.1.1 制御ルール

本節では，課題の一つである負荷の抑制を解決するため，バッファオーバフロー攻撃を検知可能で，かつ負荷の増加を抑えることが可能な制御ルールについて述べる．

一般的なオペレーティングシステムでは，メモリ上に実行コードを読み込む領域（テキスト領域）とデータを格納しておく領域（データ領域）に分けて管理を行っている．バッファオーバフロー攻撃により攻撃者が不正なコードを送り込む場合は，入力データの一部に実行コードを挿入するため，通常の実行コードとは異なり，データ領域に格納されている．したがって，この違いを識別できれば攻撃者によって不正に送り込まれたコードの実行を検出可能である．

そこで，我々は，システムコールが呼び出されたとき，そのシステムコールの呼び出し元アドレス（リターンアドレス）がテキスト領域内であるかどうかを検査することにより，システムコールの実行を制御することを提案する．システムコールの呼び出し元アドレスがテキスト領域外であった場合は，不正な実行コードによるシステムコール呼び出しとして検出する．

通常，テキスト領域は連続したアドレス空間が割り当てられるため，制御ルールとしては，システムコールの呼び出し元アドレスとテキスト領域の境界アドレスとの大小関係を調べることにより正当性を検証する．アドレスの大小関係を調べる処理であれば，アセンブラ言語でも数行で記述可能であり，システムコールの制御に要するオーバヘッドは小さく抑えられると考えられる．

4.1.2 プロセス状態の復旧

次に，不正コード実行による被害の復旧に関する解決策について述べる．

バッファオーバフロー攻撃により不正にコードが実行された場合には，スタック領域が破壊されている可能性が高く，そのままでは，プロセスが処理を継続することが困難である．そこで，対策として，組み込みデバイスが攻撃を受ける前の安定した状態（スタック領域とレジスタ値）を保存しておき，攻撃を検知した場合には保存してある状態まで復旧することを考える．復旧後，攻撃を受ける原因となった処理をスキップすることにより，たとえ攻撃を受けた場合であっても安定動作可能となる．

4.2 システム設計

システムの全体図を図 1 に示す．図中の各機

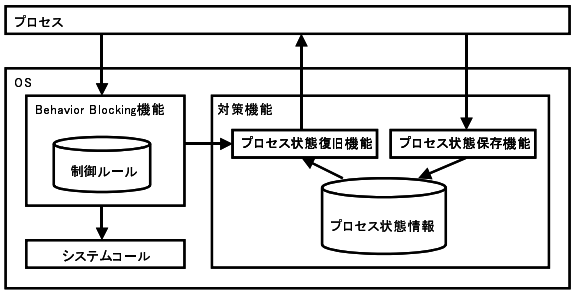

図 1: 提案方式全体図

能は以下のとおりである．

BehaviorBlocking 機能 ：システムコールをフックして正当性を検証

プロセス状態保存機能 ：プロセス状態をカーネルの領域へ保存

プロセス状態復旧機能 ：プロセスの状態を保存してあるプロセス状態へ復旧

Behavior Blocking 機能では，プロセスから発行されるシステムコールをフックし，制御ルールに基づいてシステムコールの正当性を判定する．システムコールのフックは，システムコールテーブルを書き換えることにより実現する．また，制御ルールとしては，4.1.1 節で述べたように，システムコールのリターンアドレスの検証を行う．

プロセス状態保存機能は，プロセスから呼び出され，その時点のプロセスのレジスタ値およびスタック領域をプロセス状態として保存する．

プロセス状態復旧機能は，Behavior Blocking 機能によって不正なシステムコールが検知された場合に呼び出され，保存してあるプロセスの状態情報（スタック領域，レジスタ値）を用いて，攻撃を受けたプロセスの状態を復旧する．

4.3 処理の流れ

図 2 に，プロセス状態の保存から復旧までの流れを示す．なお，発行されたシステムコールが正当であった場合は，Behavior Blocking 機能により検証後，要求されたシステムコールへ処理が移る．各処理について以下に述べる．

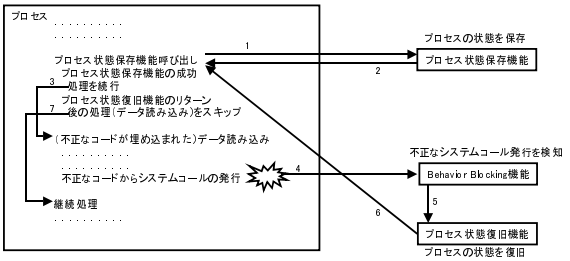

図 2: 検知時の処理

1. プロセスがプロセス状態保存機能を呼び出してプロセスの状態を保存する．
2. プロセス状態保存機能からプロセスへ処理が戻る．
3. プロセス状態の保存に成功した場合，入力データの読み込みを行う．
4. 入力データ中に埋め込まれた不正コードからシステムコールが発行される．
5. Behavior Blocking 機能が不正なシステムコールを検知し，プロセス状態復旧機能を呼び出す．
6. プロセス状態復旧機能によりプロセスの状態を復旧され，プロセス状態保存機能を呼び出した直後までプロセスの処理が戻る．
7. プロセス状態復旧機能によりプロセス状態の復旧が行われていた場合は，データの読み込みをスキップする．

5 性能実験

ここでは，提案方式を実装した際の，機能検証およびオーバヘッド測定の結果を示す．実装環境は以下の通りである．

CPU： M32R（48MHz）
OS： μITRON3.0 準拠 [5]
実装言語： C 言語およびアセンブラ言語

実装コードは，アセンブラ言語で 2000 行程度であった．

5.1 機能検証

実装した Behavior Blocking 機能，プロセス状態保存機能，およびプロセス状態復旧機能の機能を検証するため，実際にバッファオーバフローを故意に起こし，スタック上のコードを実行させた．検証に用いた関数（run_bo 関数）の概略を以下に示す [4]．

```
 1  char buf_L[100];
 2  static void run_bo(){
 3    int i;
 4    int *ret;
 5    char buf_S[50];
 6    char code[] = {/* ATTACK CODE */};
 7    ret = (int *)buf_L;
 8    for (i = 0 ; i < 25 ; i++) {
 9      *(ret + i) = (int)buf_S;
10    }
11    strcpy(buf_L, code);
12    memcpy(buf_S, buf_L, 100);
13    return;
14  }
```

6 行目はバッファオーバフローによって実行されるコードである（“ATTACK CODE” 部分には実行するバイナリコードを 16 進表記で記述）．7 ～ 10 行目の処理で，buf_S 配列の先頭アドレスを buf_L 配列のメモリ空間へコピーし，11 行目の strcpy 関数で実行コード（code 配列に定義）を buf_L 配列の先頭に書き込んでいる．この時点で，buf_L 配列は，先頭に実行コードが格納され，残りを buf_S の先頭アドレスで埋められた状態となっている．

以上のように準備した長さ 100Byte の buf_L を長さ 50Byte の buf_S へ memcpy 関数を用い

てコピーし，バッファオーバフローさせる（12行目）．このとき，buf_L の実行コードの後ろは buf_S の先頭アドレスの値で埋められているため，スタック上のリターンアドレスは，buf_S の先頭アドレスに上書きされる．

最後に，13 行目の return により関数の呼び出し元へ処理を戻そうとするが，スタック上のリターンアドレスは buf_S の先頭アドレスに書き換えられているため，本来の関数の呼び出し元へは戻らずに，buf_S の先頭に格納されている実行コードが実行される．

実験では μITRON 標準の exd_tsk システムコール（自タスクの終了と削除を行う）のアドレスを指定してジャンプする処理をコード化して埋め込んだ．

まず，提案方式により保護されていないシステム上で run_bo 関数を呼び出した際には，バッファオーバフローにより exd_tsk システムコールが呼び出され，タスクが終了した．これにより，実験に使用する run_bo 関数が機能していることが分かる．

次に，提案方式により保護されているシステム上で run_bo 関数を以下のように呼び出した．

```
1     ...
2     rv = backup();
3     if(rv == OK) run_bo;
4     ...
```

ここで，2 行目の backup 関数は，提案方式のプロセス状態保存機能を実装したシステムコールである．また，backup 関数の戻り値により run_bo 関数を呼び出しているのは，プロセスの状態を復旧後，再び run_bo 関数が呼び出され，ループしてしまうのを防ぐためである．run_bo 関数を呼び出した際には，Behavior Blocking 機能により検知し，プロセス状態保存機能を呼び出した地点まで処理が復旧後，run_bo 関数をスキップした．

以上の結果から，提案方式はバッファオーバフロー攻撃を無効化可能であるといえる．

5.2 オーバヘッド測定

提案方式では，検査対象のシステムコールをフックし，そのリターンアドレスを元にシステムコール発行の正当性を検査している．したがって，提案方式を適用していないシステムに比べて，システムコール発行時の処理時間にオーバヘッドが生じる．また，プロセス状態の保存や復旧など，提案方式を適用するために必要な手続きも追加される．そこで，本節では，提案方式を適用した場合に発生するオーバヘッドを測定し，5.1 節で検証した提案方式の効果にかかるコストについて検証する．

実験では，以下について測定した．

- t_1: ref_flg システムコール実行時間（提案方式適用前）
- t_2: ref_flg システムコール実行時間（提案方式適用後）
- t_3: backup システムコール実行時間
- t_4: バッファオーバフロー発生直後からプロセス状態が復旧されるまでの処理時間

ここで，ref_flg システムコールは μITRON 標準のシステムコールで，イベントフラグの状態参照を行う．また，t_4 の測定については 5.1 節で用いた機能検証コードを流用した．また実験では，各処理により 10000 $\sim$ 1000000 回の範囲でループさせた総実行時間を測定し，1 回の平均実行時間を算出した．

測定の結果を表 2 に示す．ただし，backup システムコールの実行時間には，Behavior Blocking 機能による正当性検査にかかる時間も含まれている．また，プロセス状態復旧機能では，エラー処理のため，一時的なプロセス状態保存などの冗長な部分も含まれている．

表 2 より，Behavior Blocking 機能にかかるオーバヘッドは以下の式で求められる．

$$t_2 - t_1 \quad = \quad 15.29\ (\mu\text{sec})$$

つまり，検査対象のシステムコールが呼ばれるたびに約 15μsec の後れが生じる．この程度のオーバヘッドでは，たとえシステムコールが頻繁に呼ばれた場合であってもシステムの性能に大きく影響するとは考えにくい．

次に，backup システムコールの実行時間（t_3）およびプロセス状態の復旧にかかる時間（t_4）についてはそれぞれ 1msec 以下であり，また，実行されるポイントや状況が限られているため，プロセスの再起動には秒単位の時間が必要になることに比べると期待できる効果に対して十分許容できる実行時間である．

6 関連研究

バッファオーバフロー対策としては，StackGuard [6]，Libsafe [7]，Openwall [8] などのアプローチが知られている．これらの方式では，バッファオーバフローの検知が目的であるため，検知後はプロセスを終了するといった手順を踏む必要がある．

エラー処理を行う際に用いられることがある C 言語標準の setjmp 関数，longjmp 関数の概念

表 2: 実行時間測定結果

測定項目	ループ回数（回）	1 回の平均実行時間（μsec）
t_1	1000000	22.74
t_2	1000000	38.03
t_3	100000	616
t_4	10000	806

は，提案方式に類似しているといえる．あらかじめ setjmp を呼び出して戻るポイントを記録しておき，障害が発生した場合に longjmp を呼び出すことにより保存した時点までプログラムの処理を戻すことができる．setjmp，longjmp を使うことで，プロセス自身が自律的にエラー処理として状態を保存・復旧することはできるが，他のプロセス（カーネルなど）からのプロセスの復旧はできない．提案方式において setjmp に相当するプロセス状態保存機能は OS のシステムコールとして実装され，longjmp に相当するプロセス状態復旧機能はカーネルの内部関数として実装されている．したがって，プロセス側で明示的に復旧を要求するというよりも，OS 上でプロセスを安定動作させるための機構として機能している．

2001 年，光来らは，プロセスクリーニングという考え方を提案している [9]．プロセスクリーニングでは，主に Web サーバなどのサーバプログラムを対象としており，サーバ起動時の状態を保存しておき，外部との通信処理が終了した際に，保存してある状態（初期状態）まで戻す．これにより，外部との通信により安全性が保証できない状態になっているサーバプログラムを保証できる状態に戻すことができる．光来らの論文では，プロセス状態保存機能およびプロセス状態復旧機能を OS のシステムコールとして実装しており，setjmp，longjmp 同様，プロセス側から明示的にプロセスの復旧機能システムコールを呼び出す必要がある．したがって，バッファオーバフロー攻撃などにより不正なコードが実行されている間は，プロセス状態を復旧することができない．

7 まとめ

本論文では，使用可能なリソースに制限がある組み込みデバイス上で実現可能な攻撃検知および対策機能について考察し，組み込みデバイス上で実装する方式としては，Behavior Blocking 方式が適していることを示した．

次に，Behavior Blocking 方式をベースとした攻撃検知方式と，検知後の対策としてプロセスの状態を復旧する方式を提案した．加えて，提案方式を実装し，機能検証およびオーバヘッドの測定を行った．これらの結果から，提案方式は，組み込みデバイス上でコンテンツに不正なコードを埋め込むような攻撃に対して効果があり，そのオーバヘッドもシステムの実行に影響を及ぼす程ではないことが分かった．これにより，組み込みデバイスに対する攻撃の検知・遮断が可能となり，かつ，攻撃を受けた後も安定した動作の継続が可能となった．

今後の課題としては，制御ルールのさらなる考察が挙げられる．本論文では，制御ルールとして，リターンアドレスの検証という極単純なルールを用いているが，より高度な攻撃を想定した場合，今回の制御ルールでは対応できない場合が予想される．しかしながら，制御ルールをより複雑なものにすればするほど，トレードオフとしてオーバヘッドが高くなると考えられるため，高機能な制御ルールを，いかに少ない処理で実現できるかが鍵となる．

参考文献

[1] CAN-2002-1327, http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-1327

[2] Behavior blocking repels new viruses, http://www.nwfusion.com/news/2002/0128antivirus.html

[3] ウイルスの脅威，携帯電話と PDA は「今のところ安全」, http://www.zdnet.co.jp/news/0010/02/mobilevirus.html

[4] unyun: バッファオーバーフローの危険性, http://www.shadowpenguin.org/sc_documents/spsdocument08.pdf

[5] Sakamura, K.: uITRON 3.0 Specification, http://www.assoc.tron.org/jpn/document.html

[6] Cowan, C., Pu., C., Maier, D., Walpole, J., Bakke, P., Beattie, S., Grier, A., Wagle, P., Zhang, Q. and Hinton, H.: StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks, Proc. the 7th USENIX Security Symposium, pp.63–78(1998).

[7] Baratloo, A., Tsai, T. and Singh, N.: Transparent Run-Time Defense Against Stack Smashing Attacks, Proc. USENIX Annual Technical Conference(2000).

[8] Openwall Project: Non-Executable User Stack, http://www.openwall.com/.

[9] 光来健一, 千葉滋: サーバのアクセス制限を安全に変更するための機構, 情報処理学会論文誌, Vol.42, No. 6, pp.1492–1502(2001).