ソフトウェアプロテクションの技術動向（後編）
—ハードウェアによるソフトウェア耐タンパー化技術—

門田暁人
奈良先端科学技術大学院大学 情報科学研究科
akito-m@is.naist.jp

Clark Thomborson
Department of Computer Science, The University of Auckland
cthombor@cs.auckland.ac.nz

前編では，ソフトウェア単体でのプロテクション技術を紹介した．本稿（後編）では，より強力な技術として，ハードウェアを併用する方式について述べる．

近年の動向として，IBM，Intel，Microsoftなどの大手コンピュータ企業を中心とする標準化団体Trusted Computing Group (TCG)が，PCに搭載するセキュリティチップ（TPM: Trusted Platform Module）の仕様を策定している☆1．また，TCGの技術を応用したアーキテクチャとして，MicrosoftはNext-Generation Secure Computing Base（NGSCB）を発表し☆2，次期Windows（Longhorn）への実装を予定している．一方，有料テレビ放送のデコーダなどの小規模なシステムには，より安価で単純なバス暗号アーキテクチャが従来より用いられている．また，近年，コンピュータの命令セットに個体差（多様性，Diversity）を持たせることで不正なソフトウェアに対する耐性を高める方法も研究されている．本稿では，これらの技術について，網羅的に紹介する．

ハードウェア支援によるプロテクション

ソフトウェア単体での保護技術は，white-box暗号システムなどの有力な方法があるとはいえ，その有効範囲は限定されている．特に，ライセンスチェックルーチンのようなプログラム中の機能分岐点を完全に隠蔽することは事実上不可能である．ライセンス認証に成功した場合，および，失敗した場合の計算機の挙動や状態を測定し，比較することで，それらの差分から機能分岐点を特定できるためである．このような動的解析からプログラムを保護するためには，メモリやバスが情報を漏らさないように“ブラックボックス化”する必要がある．

ブラックボックス実行

プログラムの実行状態を隠蔽し，ブラックボックス化する1つの方法は，プログラム全体をexecute-onlyメモリ上で実行することである．典型的には，execute-onlyメモリは，CPUと同一のハードウェアチップ上のread-onlyメモリ（ROM）として実装される．CPUが厳密なハーバードアーキテクチャを持ち，プログラムとデータが分離されている場合，ブラックボックスであるCPU内部のプログラムを，データレジスタや外部メモリ，入出力回路上に読み出すことは不可能となる．

ブラックボックスのもう1つの実装方法は，プログラムおよび関連するデータをハードウェアによる論理回路（たとえばASIC：Application Specific Integrated Circuit）として構成することで，命令フェッチなしにプログラムを実行可能とすることである．この方法は，暗号データの復号ルーチンのような，汎用的かつ安全性が要求されるプログラムにおいて特に有効である．

攻撃手段としては，ブラックボックスに対して何らかの系統立った入力を与え，その出力を観測することで，ブラックボックスの中身を推測することが考えられる．

☆1 The Trusted Computing Group: Home, https://www.trustedcomputinggroup.org/
☆2 Microsoft NGSCB Webpage，http://www.microsoft.com/ngscb/　ただし，2005年2月末の時点ではNGSCB関連資料はアーカイブページにしまい込まれており，今後の動向が注目される．

プログラムの仕様がある程度複雑で，かつ，そのI/Oインタフェースが慎重に設計されているならば，このような攻撃が成功する確率は低くなる．ただし，慎重に設計されたシステムであっても，攻撃が成功した例が知られている．BondとAnderson[1]は，IBMのCommon Cryptographic Architecture version 2.40に対し，ブラックボックス内部から暗号鍵を特定するのに成功した☆3．

その他の攻撃としては，消費電力に注目した電力解析攻撃（power analysis attack），処理時間に注目したタイミング攻撃（timing attack）などのいわゆるサイドチャネル攻撃や，チップ表面を削って回路を露出させて解析するといった物理的な攻撃が考えられる．このような攻撃に対しては，物理的，および，電気的な耐タンパーメカニズムを施すことが必要となる☆4．

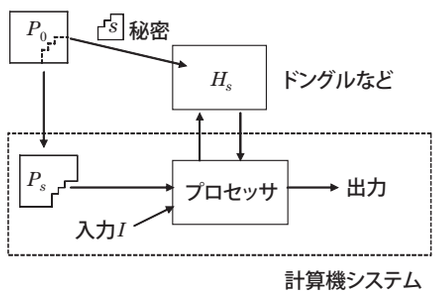


図-1　プログラムの部分的なブラックボックス化

部分的なブラックボックス

前述のような純粋なブラックボックス方式は，ハードウェアチップ内部のプログラムをアップデートする手段がないという欠点がある．アップデートするためにはブラックボックス環境全体を差し替える必要があり，大きなコストがかかる．そこで，アップデートを可能とするために，プログラムの一部のみをブラックボックスとして実装する方法が考えられる[2]．

この方法では，**図-1**に示すように，保護対象となるプログラムP_0の重要な部分（sと記す）を切り取り，耐タンパー性を持ったハードウェアモジュールH_sに実装する．モジュールH_sの媒体としては，ドングル（シリアルインタフェースやUSBインタフェースに差し込む小さなデバイス）やスマートカードが考えられる．H_sを安全なリモートホストに設置することも考えられる．sを切り取った残りのプログラム（P_sと記す）は，通常のプログラムと同様の方法でユーザに配布・販売される．このP_sは，計算機上で通常通り実行されるが，sの実行はH_sへのサブルーチン呼び出しにより行われる．

この方法の利点は，sを適切に選ぶことで，プロテクトを外すことが困難となることである．また，ドングルのような装置を追加することは多くのコンピュータにおいて比較的小さなコストで可能であるため，この方式の適用範囲は広い．

秘密sにクライアント側のプラットフォームの署名を含ませることも可能である．これにより，特定のプラットフォームでのみプログラムが実行可能となるようにP_sを特殊化できる．ただし，この場合，プラットフォームのハードウェアもしくはソフトウェアをアップグレードするたびに，P_sのアップデートが必要となる．

この方式の欠点の1つは，十分な安全性を確保するためには実装上の工夫が不可欠で，そのための労力・コストが大きいことである．攻撃者は，P_sを解析することで，H_sに含まれるプログラムを呼び出す方法を知ることができる．さらに，攻撃者は，H_s内部のサブルーチンを呼び出すためのテスト用プログラムを作成し，さまざまな入力をH_sに与えてその返り値を分析する可能性がある．特に，sが定数値を返すコードであった場合，この攻撃によりsを知ることは容易である．このような攻撃により，従来，ドングルに含まれるサブルーチンの多くが解析され，その秘密が漏洩した．十分な安全性を確保するためには，実装者は，推測が困難なsをうまく選び，P_sを難読化し，さらに，H_sの入出力を暗号化するなどして隠蔽する必要がある．なお，一般に，H_sの実行環境における計算能力はP_sの実行環境と比べて著しく小さいため，sの選定においては，実行効率も十分に考慮に入れ，「重要ではあるが実行頻度は高くない」部分をsとして選ぶ必要がある．

もう1つの欠点は，P_sをアップデートする際に，アップデートの内容がP_sだけで閉じていれば問題ないが，場合によってはsも同時にアップデートする必要があることである．この場合，暗号化されたsをクライアントへと送信するセキュアな通信路が必要となり，この部分が新たなセキュリティホールとなり得る．

☆3 Version 2.4.1ではこの攻撃を防ぐ対策が施されている．
http://www-3.ibm.com/security/cryptocards/pdfs/CCA_Basic_Services_241_Revised_20030918.pdf
☆4 FIPS PUB 140-2, NIST, http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf

On-the-fly暗号

On-the-fly暗号は，メインメモリ上の暗号化されたプログラムおよびデータを，CPU内部においてバイト単位もしくはワード単位で復号しながら実行する方式である．この方式では，復号したプログラムを保存するためのブラックボックスを必要としない．そのため，構造が単純で実装コストが小さいという特長があり，多くのバス暗号プロセッサで採用されている．

この方式の欠点は，暗号文の最小単位であるブロック長が短いため（バイト，もしくは，ワード），暗号強度が弱いことである．一般的な平文と異なり，プログラムには分岐があるため，各暗号ブロックの復号順序が一意に決まらない．そのため，CBC（Cipher Block Chaining）やCFB（Cipher Feedback）などの暗号モードを用いて暗号ブロックを連鎖させることができず，安全性を高める手段が限定される．

安全性を高める1つの手段として，バス暗号プロセッサでは，アドレスバスを暗号化している．すなわち，各暗号ブロックの位置（メモリアドレス）をランダム化し，そのアドレスを鍵の一部として使う．たとえば，バス暗号プロセッサDP5002FPでは，**図-2**に示すように，3つの暗号化関数（17bitアドレスバスを暗号化するEA，8bitデータバスを暗号化するED，および，8bitデータバスを復号するED^{-1}）を用いる☆5．暗号鍵Kは，CPUチップ内の耐タンパーRAM上に格納されている．データバス暗号化関数EDは，この鍵Kとアクセスされるメモリアドレスaを用いる．CPUコアが「データdをアドレスaに書き込む」という動作を行った場合，実際にはデータ$d' = ED_{K,a}(d)$がアドレス$a' = EA_K(a)$に書き込まれる．一方，アドレスaからデータを読み込む場合には，実際にはアドレスa'からデータd'が読み込まれ，$d = ED^{-1}_{K,a}(d')$によって復号されたデータがCPUコアに渡される．

ただし，このような手段を用いたとしても，デバッガやロジックアナライザを用いた選択暗号文攻撃に対して十分安全とはいえない．攻撃者は，メモリ上に任意の暗号文を書き込み，CPUに実行させ，システムの動作（たとえば，メモリへの書き込みが発生したか否かなど）を観察することで，平文（プログラムのオペコードやオペランド）を推測できる可能性がある．Kuhn[3]は，DS5002FPに対して選択暗号文攻撃を行い，メモリ上の全データをシステム外部（パラレルポート）に出力するための暗号文（暗号化されたプログラム）を得た．システム外部へのバスは暗号化されていないため平文が出力され，復号されたプログラム全体が得られることになる☆6．1つの対策として，多くのバスプロセッサでは，書き込み／読み出し位置が知られないように，ダミーの書き込み／読み出しを追加することが行われている．この方式はBestによって数十年前に提案され，十年ほど前にGoldreichとOstrovskyによるOblivious RAMとして理論面からも研究された[4]．

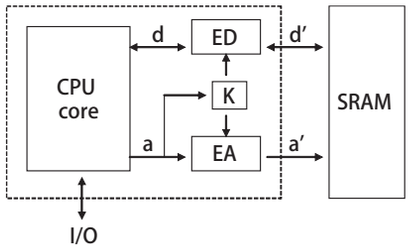


図-2　バス暗号プロセッサ（点線部分）と外部メモリ[3]

On-the-fly暗号は，コストや電力の関係から高度なメモリアクセス制御機構を実装することが難しい分野に適用される．たとえば，金融取引端末，有料テレビ放送のデコーダ，スマートカードなどである．多くのバス暗号プロセッサは，物理的に保護されたメインメモリを持ち，ロジックアナライザ等による攻撃への耐性を高めている．

CPUキャッシュメモリによる仮想ブラックボックス

前述のように，on-the-fly暗号は，選択暗号攻撃に対して弱点を持つ．安全性を高める1つの方法は，完全なon-the-fly実行をあきらめ，一度に復号する暗号文のバイト数（暗号ブロック長）を大きくすることである．この場合，復号後の命令系列を記憶するためのキャッシュメモリをCPU内部に設ける必要がある．このようなプロセッサ内部のキャッシュメモリを“仮想的なブラックボックス”として用いる方法は数多く提案されており，Cerium[5]，XOM[6]，AEGIS[7]などのアーキテクチャが提案されている．

この方法では，外部メモリ上のデータやプログラムを暗号化しておき，それらが必要になった場合にのみ

☆5 “DS5002FP Secure Microprocessor Chip”，Dallas Semiconductor, http://pdfserv.maxim-ic.com/en/ds/DS5002FP.pdf
☆6 Kuhnは論文3）を発表する前にDS5002FPの開発者にこの弱点を報告し，論文が発表された時点では対策が講じられている．

キャッシュメモリへと復号し，実行する．プロセッサチップは物理的・電気的に耐タンパー性を持つことが要求される．プロセッサ外部のメモリにデータの書き込みを行う場合には，on-the-fly方式と同様，書き込み前にデータを暗号化しておく必要がある．

この方式の欠点は，チップ外部とのデータ入出力時に暗号化／復号処理を必要とするため，実行パフォーマンスが低下することである．一般に，暗号ブロック長を大きくする，または，強い暗号方式を採用した場合には，この低下はより深刻なものとなる．実行効率の観点から，対称鍵暗号を用いることになるが，安全性の面から，全暗号ブロックで同一鍵を用いるわけにはいかない．そこで，マスターキーとなる情報，各暗号ブロックが記録されている外部メモリのアドレス，および，ブロック内の特定のビット列などからブロックごとに個別の鍵を生成し，復号することとなる．この処理によるオーバーヘッドは，復号専用のハードウェアを用いることで小さくできる．たとえば，AEGISプロセッサでは，復号アルゴリズムとして128ビット鍵のAES（Advanced Encryption Standard）を採用し，0.13 μ プロセス技術を用いた30万個の論理ゲートで構成される復号専用ハードウェアにより，128bitのブロックを20nsで復号できるとされている[7]．

アクセス制御による仮想ブラックボックス

仮想的なブラックボックスを計算機内に構築する1つの手段として，プロセッサ外部のRAMに強力なアクセス制御を行う方法が近年注目されている．TCGやNGSCBが目指すのもこの方式であり，NGSCBではブラックボックス化されたメモリ領域のことをcurtained memoryと呼んでいる．

従来の多くのオペレーティングシステム（OS）においても，各アプリケーションが占有するメモリに対し，互いにアクセスできないようにアクセス制御を行い，不正なプログラムから各プロセスを保護している．しかし，buffer overflowを利用した攻撃の問題，OSのアクセス制御機構自身が改ざんされるという問題，OSのバグによるセキュリティホールの問題，デバッガの存在，DMA（Direct Memory Access）デバイスの存在といった懸案事項があり，従来のPC環境で堅牢なアクセス制御を実現することは困難であった．

NGSCBでは，ブラックボックス実行のための専用のCPUモードを必要とする．この専用のCPUモードで実行されるプロセスは，他のプロセスやDMAデバイスから隠蔽される．また，メモリへのアクセス制御機構をnexusと呼ばれるごく小さなセキュリティカーネルに封じ込め，その仕様と実装を徹底して検証することで，セキュリティホールの問題の解決を目指している．また，前述のように，プログラムへのI/Oは攻撃対象となるため，キーボード，マウス，ビデオその他のデバイスにセキュアI/O機能（入出力データの暗号化）を持たせることを想定している．

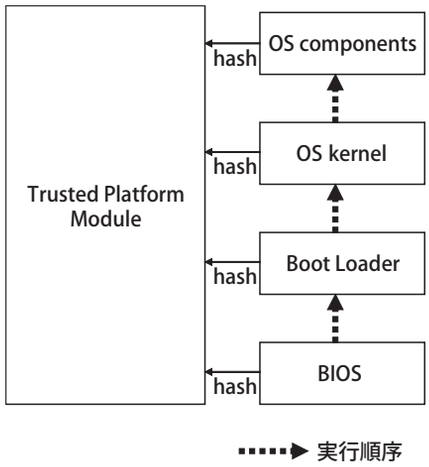


図-3　TCGプラットフォームにおけるブートプロセス

アプリケーションの開発者の立場からは，OSが正当なものである（改ざんされていない）という保障が必要となる．そのために，NGSCBやTCGでは，特殊なブートストラップ方式☆7を採用している．**図-3**に示すように，TCG準拠のシステムでは，Trusted Platform Module（TPM）と呼ばれる耐タンパーハードウェアが，システムブート時に各ソフトウェアコンポーネント（BIOS，ブートローダ，OS）のハッシュ値を計測し，保存する．これらのハッシュ値により，各コンポーネントの改ざんが順次検出できる．なお，TPM自身もEndorsement Keyと呼ばれる鍵を含んでおり，任意のコンポーネントに対してTPMの正当性を示すことができる．アプリケーションの開発者は，特定のTPMやOS上でのみ動作するよう，アプリケーションに署名をつけることができる．

計算機上のデータに対する従来のアクセス制御は，不正なユーザによるアクセスを防ぐために，OSがユーザ

☆7 authenticated boot，または，trusted bootと呼ばれる．

を認証し，特定の人間のみにアクセス権を与えている．一方，クラッキングからプログラムやデータを保護する上では，特定のユーザを信頼してアクセス権を与えることは危険である．その代わりに，NGSCBでは，プログラム開発者が信頼する特定のハードウェア（TPM），OS，プログラムに対してアクセス権を与える仕組みを提供している．この仕組みにより，プログラム開発者は，信頼できない計算機にプログラムやデータを送信する場合であっても，その計算機上で動作するTPMやプログラムが信頼できるならば，それらにアクセス権を与えることで，送信したプログラムやデータの機密性を保つことができる．

RAM上に構築された仮想的なブラックボックスは，code-injection attackから防御することが必須となる．この攻撃は，ソフトウェアのセキュリティホールとなっているbuffer overflowを利用し，メモリ上のデータ領域を溢れさせてトロイの木馬となるプログラムを書き込み，そこに実行を移すものである．その対策として，最近のIntel，AMD，Transmeta等を始めとする多くのCPUでは，プログラム領域とデータ領域を完全に分離することをCPUレベルで保障しており，メモリ上の各セクションはwriteかexecuteのどちらか一方しか許可されない（W^X: Write xor eXecute，NX: No eXecute，または，DEP: Data Execution Prevention機能と呼ばれる）ようになっている☆8．

命令セットのランダム化

近年，コンピュータに個体差（多様性，Diversity）を持たせることで不正なソフトウェアに対する耐性を高める方法が研究されている[8]．たとえば，OpenBSD 3.4では，ランタイムローダld.soがライブラリをランダムな順序でロードすることで，ウィルスやワームへの耐性を高めている．

命令セットのランダム化は，コンピュータに個体差を持たせるために，プロセッサ（もしくはインタプリタ）ごとに異なる命令セットを用意する手法である．各プロセッサ上で動作するソフトウェアはそれぞれ異なる命令セットで記述されるため，特定のプロセッサ向けに作られたウィルスやワームへの耐性を高めることができる．

この手法は，on-the-fly暗号の一種であり，各プロセッサは，暗号化されたオペコードと復号されたオペコードの間の一対一写像を持つ[8]．具体的には，プロセッサごとに異なる鍵Kを持たせ，プログラム実行時に（メモリから命令がフェッチされるたびに）各命令と鍵KのXORを取ることで命令を復号する．この方法はプロセッサの命令フェッチユニットにわずかな機構を追加するだけで実現でき，実行パフォーマンスの低下もほとんどない．

ただし，命令セットのランダム化は，単純な換字暗号であると言えるため，攻撃者がメモリを覗き見できる場合には，安全性は低い．この方法が有用となるのは，攻撃者がシステムに対して非常に限定したアクセスしかできない場合である．たとえば，ファイヤウォールで守られたネットワーク上の計算機において，コンピュータウィルスやトロイの木馬による攻撃を防ぐ場合に有効である．

命令解釈の暗号化

命令解釈の暗号化（encrypted interpretation，または，obfuscated interpretation）は，on-the-fly暗号の一手法であり，プロセッサ内部のデコーダ（命令解釈ユニット）に追加の機構（復号器）を設置することで実現される[9]．この方法は，命令セットのランダム化と同程度の実行効率で，バス暗号と同程度の安全性を実現する．

この方式は，命令セットのランダム化と同様，各オペコードとオペランドが1個の暗号ブロックとなる．前述のように，プログラムには分岐があるため，CBCモード等により暗号ブロックを連鎖させることができないが，この方式では，プログラム中の分岐の各合流点の直前に，復号器の内部状態を変化させるためのダミー命令をあらかじめ挿入しておくことで，暗号ブロックの連鎖を可能としている．**図-4**に，オペコードaddとsubを暗号化したプログラムの例を示す．復号器としては，有限状態オートマトンを用い，オートマトンの各状態にそれぞれ異なるオペコード復号関数（入出力関数）を持たせることで安全性を高めている．プロセッサがオペコードを解釈するたびに（オペコードの種類に依存して）オートマトンの状態遷移が起こり，次に解釈する暗号ブロックに対する復号関数が変化し，暗号ブロックの連鎖が起こる．プログラムの分岐による誤動作を回避するために，ダミー命令によって，プログラムの各合流点において必ず特定の状態へとオートマトンが遷移するように制御される．

有限状態オートマトンによる暗号は，暗号方式としてはDESやAES等の現代暗号よりも低い安全性しか持たない．ただし，実装コストが大きくない（CPUのキャッシュメモリを必要としない）ことから，バス暗号などの他のプロテクション方式と併用することで，システムの安全性を高めるのに役立つと期待される．

☆8 Windows XPでは，Service Pack 2よりこの機能が利用可能となっている．

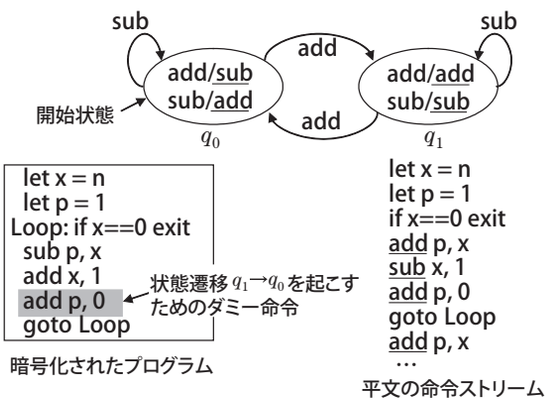


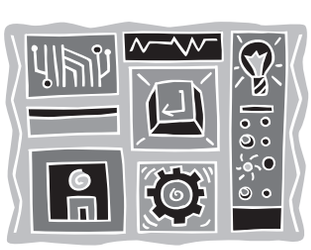
図 -4　オートマトンによるプログラムの復号

むすびにかえて

本稿では，ハードウェアの支援によりソフトウェアを耐タンパー化する技術について紹介した．特に，TCG，および，NGSCBは，今日のコンピュータ産業を支える中心的企業が推進しているだけに，今後プラットフォームの普及が加速していくと予想される．ただし，計算機のユーザを信頼せず，プラットフォームやベンダを信頼するというアプローチに対する批判もある☆9．個人ユーザへ普及させるためには，ユーザに一定の信頼を置くような仕組みが必要になるであろう．

参考文献
1）Bond, M. and Anderson, R. : API-level Attacks on Embedded Systems, Computer, Vol.34, No.10, pp.67-75（Oct. 2001）.
2）Mana, A. and Pimentel, E. : An Efficient Software Protection Scheme, Proc.16th IFIP International Conference on Information Security, pp.385-401（2001）.
3）Kuhn, M. : Cipher Instruction Search Attack on the Bus-encryption Security Microcontroller DS5002FP, IEEE Trans. on Computers, Vol.47, No.10, pp.1153-1157（1998）.
4）Goldreich, O. and Ostrovsky, R. : Software Protection and Simulation on Oblivious RAM, J. ACM, Vol.43, No.3, pp.431-473,（1996）.
5）Chen, B. and Morris, R. : Certifying Program Execution with Secure Processors, Proc. 9th Workshop on Hot Topics in Operating Systems, USENIX, pp.133-138（May 2003）.
6）Lie, D., Thekkath, C. and Horowitz, M. : Implementing an Untrusted Operating System on Trusted Hardware, Proc.19th ACM Symposium on Operating Systems Principles, pp.178-192（Oct. 2003）.
7）Suh, G., Clarke, D., Gassend, B., van Dijk, M. and Devadas, S. : AEGIS: Architecture for Tamper-evident and Tamper-resistant Processing, Proc.17th International Conference on Supercomputing (ICS2003), pp.160-171（June 2003）.
8）Barrantes, E., Ackley, D., Forrest, S., Palmer, T., Stefanovic, D. and Zovi, D. : Randomized Instruction set Emulation to Disrupt Binary Code Injection Attacks: Proc.10th ACM Conference on Computer and Communications Security (CCS2003), pp.281-289（2003）.
9）Monden, A., Monsifrot, A. and Thomborson, C. : Tamper-resistant Software System based on a Finite State Machine, IEICE Trans. on Fundamentals, Vol.E88-A, No.1, pp.112-122（Jan. 2005）.

（平成17年3月8日受付）



☆9 Anderson, R. : Trusted Computing FAQ, version 1.1（Aug. 2003）. http://www.cl.cam.ac.uk/~rja14/tcpa-faq.html