データフロー解析に基づく セキュアソフトウェア開発支援技術 DFITS$_{TM}$

DFITS$_{TM}$ Technology Supporting Secure Software Development Based on Data Flow Analysis

林　良太郎　■HAYASHI Ryotaro
中西　福友　■NAKANISHI Fukutomo
橋本　幹生　■HASHIMOTO Mikio

近年，パソコン（PC）や情報端末などに搭載されたソフトウェアが不正解析され，個人情報などの機密データが流出するという事例が報告されている。このような脅威に対して安全なソフトウェアを開発するための技術への要求が高まっている。

東芝は，安全なソフトウェアの開発を支援する技術DFITS$_{TM}$（Data Flow Isolation Technology for Security）を開発している。DFITS$_{TM}$により，ソフトウェアの基になるソースコードを静的に解析し，保護が必要な機密データの洗出し作業を自動化することで，ソフトウェアの安全性強化と開発効率向上が期待できる。

There have recently been many incidents involving unauthorized release of confidential data such as personal information from illegally analyzed software installed in PCs and information terminals. Demand is therefore increasing for technologies to develop secure software with enhanced protection against such threats.

Toshiba has developed DFITS$_{TM}$ (Data Flow Isolation Technology for Security), a technology that supports the development of software more securely and efficiently by automatically performing static analysis of the source code of software and classifying confidential data in the software.

1　まえがき

家電製品の組込みソフトウェアや，PCや情報端末などのアプリケーションソフトウェア，社会インフラの制御システムソフトウェアなど，人々の身の回りにはソフトウェアがあふれているが，これらのソフトウェアの中には，機密データを処理するものがある。例えば，インターネットブラウザソフトウェアで扱うクレジットカード番号は機密データである。また，クレジットカード番号に暗号処理を行う場合は，暗号化や復号に使う鍵値も機密データである。

近年ソフトウェアの数が増加する一方で，ソフトウェアに含まれる機密データが，悪意のあるユーザーにより解析（リバースエンジニアリング）されたり，ウイルスなど悪意のあるソフトウェア（マルウェア）から攻撃されたりするという問題が顕在化してきている。実際，PC向けDVD再生ソフトウェアに含まれる機密データ（復号処理用の鍵値）が，デバッガや逆アセンブラのようなソフトウェア開発ツールを使って解析された事例が報告されている。このような脅威は，オープン化してマルウェアなどが侵入しやすい，スマートグリッドのような社会インフラシステムの場合に深刻な問題になりうる。

このような脅威に対して，ソフトウェアが持つ機密データを保護し，ソフトウェアが正しく実行されることを保証するためのソフトウェア保護技術に対する要求が高まっている。ここでは，安全なソフトウェアを開発する手順とその難しさについて述べるとともに，東芝が開発している，安全なソフトウェアの開発を支援する技術DFITS$_{TM}$（Data Flow Isolation Technology for Security）について述べる。

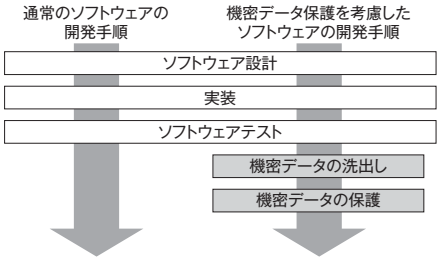

図1．ソフトウェアの開発手順の比較 ― 安全なソフトウェアを開発するには，ソフトウェアに含まれる機密データを洗い出して適切に保護する必要がある。

Comparison of development flows of regular software and software with confidential data protection

2　安全なソフトウェアの開発

2.1　ソフトウェアの開発手順

図1は，通常のソフトウェアの開発手順と，前述したような脅威に対して安全なソフトウェアを開発するための手順を比較したものである。

通常のソフトウェア開発は，ソフトウェア設計，実装（ソースコード記述），ソフトウェアテスト（機能テスト）の順で進む。このとき，ソフトウェアテストを十分に行うことで，ソフトウェアの機能面のバグ（コーディングの誤りや欠陥）を取り除くことができる。

これに対して，機密データ保護を考慮した安全なソフトウェ

アを開発する場合は，更にソフトウェア保護技術を導入する必要がある。具体的にはまず，ソフトウェアに含まれる保護しなければならない機密データがどこに含まれているのかを調べ，漏れなく洗い出す。そして，洗い出した機密データを保護してソフトウェアを構成する。機密データ保護の手段としては，当社が開発した，プロセッサチップ内に埋め込まれた鍵により機密データを暗号化保護するLMSP$_{TM}$（License-controlling Multi-vendor Secure Processor）[1]などのセキュリティプロセッサや，暗号化されたソフトウェアをみずから復号しながら実行するソフトウェア難読化技術[2]がある。

2.2　機密データの洗出しの困難性

安全なソフトウェアを開発する場合，前述したように機密データの洗出しが必要であるが，一方で，機密データとそれ以外を分類せずにソフトウェアに含まれる全てのデータを保護してしまえば十分であるようにも思える。しかし，ソフトウェアが扱うデータを全て保護するということは，ソフトウェアに関する情報が外部からいっさい読み書きできなくなることを意味する。すなわち，OS（基本ソフトウェア）を介した他のアプリケーションとの通信（入出力）もいっさいできないことになってしまい，基本的なソフトウェアの機能要件が満たされない。以上のことから，ソフトウェアに含まれる機密データだけを保護し，入出力などは保護しないことが不可欠である。

機密データの洗出し作業はソフトウェア開発者が行わなければならない作業であるが，これは容易な作業ではない。

まず，機密データを扱うソフトウェアは，暗号処理や認証などのセキュリティ機能を実装したものが多い。この場合，実行される暗号処理の意味を把握するなど，セキュリティに関する専門知識が必要であるが，全てのソフトウェア開発者がセキュリティに明るいとは限らない。

次に，機密データの洗出し作業は，実装されたソースコードにおいてデータの格納先を示す“変数”に注目して行う。すなわち，ソースコードに含まれる全ての変数に対して，その変数に格納されるデータが機密データかどうかを判定しなければならない。膨大な変数の中から，機密データを格納する保護すべき変数を漏れなく洗い出す（分類する）作業は煩雑であり，かつ困難を伴う。セキュリティ機能を実装すると，**図2**に示すように，入出力やデータサイズなどの通信に用いる非機密データと暗号処理に用いる機密データとが複雑に絡み合った構造になる。更に，それらの値が一時的にコピーされる変数なども含めると，セキュリティに詳しいソフトウェア開発者でさえ分類ミス（ヒューマンエラー）を起こしかねない。

機密データの洗出し作業時に，保護すべき変数を見逃してしまうと，機密漏えいなどの重大な問題につながる可能性がある。更にやっかいなことに，分類ミスはソフトウェアの機能には直接影響しないため，通常のソフトウェアテストではまったく検出されない。このため，ソフトウェアが潜在的な脆弱（ぜいじゃく）性を持ったまま出荷され，その後ソフトウェアが攻撃されてはじめて脆弱性が顕在化することになってしまう。

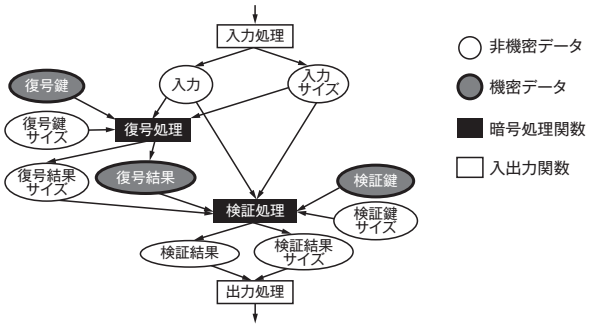

図2．セキュリティ機能の実装におけるデータの流れ ― 機密データと非機密データが複雑に絡み合い，機密データの洗出しは容易ではない。
Data flow of security function implementation

そこで，ソフトウェアに含まれる機密データの洗出し作業に起因するソフトウェア開発者の負担を軽減し，ヒューマンエラーによる分類ミスを防止するためのソフトウェア開発支援技術が求められている。

3　DFITS$_{TM}$の概要

2.2節で述べたようなニーズに応えるため，当社は，保護すべき機密データの洗出しを支援するセキュアソフトウェア開発支援技術DFITS$_{TM}$を研究開発している[3]。

DFITS$_{TM}$は，ソフトウェアの基になるソースコードを解析し，情報漏えいや不正改ざんにつながるおそれのある機密データが格納される変数の洗出しを行う。DFITS$_{TM}$を用いることにより，図1に示した機密データ保護を考慮したソフトウェアの開発手順における機密データの洗出し作業を効率化することが可能になり，安全なソフトウェアを効率よく構築することができる。

DFITS$_{TM}$は，ソースコードにおけるデータの流れ（データフロー）に注目して変数の分類を行う。すなわち，ソースコードからデータフローを抽出し，このデータフローを解析して，機密情報が含まれる保護データフローとそれ以外の非保護データフローとを混在しないようにするというルールに基づき，各変数の保護と非保護を分類している。

3.1　保護属性

DFITS$_{TM}$では，変数の分類カテゴリーとして“保護属性”と呼ばれるセキュリティ型情報を定義し，**表1**に示すように，変数を“機密性”と“完全性”の二つの観点で分類する。ここで，機密性とは権限を持つ人だけが情報にアクセス可能で，権限のない人には読み取れないという性質である。完全性とは情報が不慮の事故や攻撃者による改ざんにより壊れていないという性質である。これ以降に示す図では，変数とその保護属性を表1の記号で表す。

表1. 保護属性
Protection attributes

保護属性	記号	機密性	完全性	利用例
exposed	○	なし	なし	入出力
verified	◎	なし	あり	公開鍵
concealed	●	あり	なし	未検証の復号結果
confidential	◉	あり	あり	秘密鍵

例えば，入出力データなど，OSや外部アプリケーションとやりとりを行うデータを格納する変数は，外部から書込み及び読出しが可能なexposed属性でなければならない。一方，秘密鍵のように機密性と完全性の両方が要求されるデータを格納する変数は，confidential属性でなければならない。

3.2 データフロー解析における変数分類規則

DFITS$_{TM}$では，データフロー中の変数の保護属性に以下に述べるような制約条件を定めている。

(1) 代入や四則演算などの各演算において，演算に関わる全ての変数の保護属性が一致する。例えばa＝b＋cという加算及び代入演算の場合，変数a，b，cの保護属性は同一である。

(2) 暗号処理関数（セキュリティ関数）及び入出力処理関数に関わる変数の保護属性は，処理内容に基づきあらかじめ定義された保護属性に限られる。

(1)は，非保護データと保護データを混同することによる機密情報の漏えいや実行情報の改ざんを防ぐための制約である。これにより，保護データに関わるフローと非保護データに関わるフローを分離することができる。(2)は，暗号化や署名などの暗号処理関数に関わるデータの保護，非保護を決めるための制約である。この制約条件は，ソフトウェアで利用する暗号処理関数に応じて，適宜追加することができる。ただしこの作業は，セキュリティの専門知識を持つソフトウェア開発者が行う必要がある。

暗号文が復号処理関数である場合の制約条件の例を**図3**に示す。

まず，鍵の保護属性について考える。鍵が不正に読み取られると，暗号文を不正に解読されるおそれがあるため，機密性保護が必要である。また，鍵が不正に改ざんされると，復号結果の平文が壊れてしまい，場合によっては攻撃者にその平文の内容を推定されるおそれがあるため，完全性保護も必要である。以上から，鍵を格納する変数には，機密性と完全性の両方が必要なconfidential属性が定義されている。

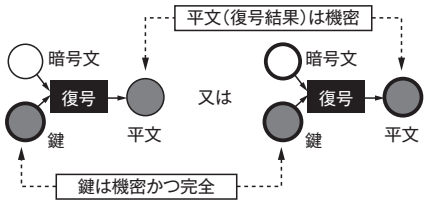

図3. 復号処理関数における保護属性制約条件 — 関数の処理内容を考慮して変数の保護属性が定義される。
Rules of protection attributes for decryption function

次に，暗号文と平文について考える。暗号文は暗号化されているので機密保護する必要はないため，対応する変数には機密性なしの保護属性が定義される。一方，復号結果である平文については，機密情報として扱う必要があるため，対応する変数には機密性ありの保護属性が定義される。また，平文の完全性の有無は，暗号文の完全性の有無をそのまま引き継いでいる。

ここでは復号処理関数の例を示したが，その他にも必要に応じて暗号化や，署名生成，署名検証，ハッシュ関数など，復号以外のセキュリティ関数についても入出力変数の保護属性が定義される。また，入力関数に関わる変数は外部から書込み可能なexposed属性，出力関数に関わる変数は外部から読取り可能なexposed属性又はverified属性に限定される。

以上の制約規則を基に，DFITS$_{TM}$はセキュリティ関数を起点として各変数を保護属性ごとに分類する。より具体的には，データフローから抽出された制約条件を満たす保護属性を求める問題（制約充足問題）を解くことで，各変数に対する保護属性（問題の解）を決定していく。

4 DFITS$_{TM}$プロトタイプシステムの試作と実証実験

前述したDFITS$_{TM}$を実証するため，C/C++言語を解析対象とするプロトタイプシステムを試作して評価した。

4.1 DFITS$_{TM}$プロトタイプシステム

DFITS$_{TM}$プロトタイプシステムは，C/C++言語で書かれたソースコード，及びセキュリティ関数の制約条件を入力としている。まず，DFITS$_{TM}$プロトタイプシステムはソースコードからデータフローを抽出する。次に，抽出したデータフローを解析して変数を保護属性ごとに分類し，変数と保護属性の組を出力する。このとき，**図4**に示すように，抽出したデータフローに保護属性情報を付与した保護属性付きデータフローも出力可能である。

試作において，データフローの抽出にElsa（the Elkhound-based C++ Parser）[4]と呼ばれるC++構文解析支援ライブラリを利用している。変数の分類では，まず，データフローの中で代入，セキュリティ関数呼出し，及び演算を行っている部分について，それぞれ制約条件の抽出を行う。次に，これらの制約条件の下で各変数の保護属性の推論を行う。この推論には，型推論のアルゴリズムを利用している。型推論とは，ソースコード中の型が宣言されていない変数に対して，その使われ方から型を自動的に求める仕組みである。DFITS$_{TM}$では，保

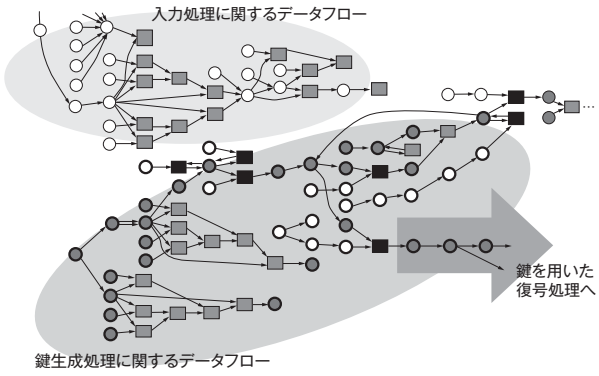

図4. DFITS$_{TM}$が出力したデータフロー（一部）— 入力処理に関わる変数は保護が不要，鍵生成処理に関わる変数は保護が必要と分類されている。
Portion of data flow generated by DFITS$_{TM}$

護属性をセキュリティレベルを表す静的な型情報とみなして型推論アルゴリズムを適用している。

4.2 実証実験

DFITS$_{TM}$プロトタイプシステムを用いて，いくつかの実用ソースコードに対して実証実験を行い，機密データが格納される変数が適切に分類されることを確認した。ここでは，デジタル著作権管理のためのシステムであるAACS（Advanced Access Content System）を実装したソースコードに対する実証実験について述べる。

実験では，AACSの鍵生成及び復号処理部分を実装したソースコードに対して，DFITS$_{TM}$プロトタイプシステムを適用した。このとき，検証関数や復号関数など，10種類のセキュリティ関数の制約条件を定義した。実験の結果，337個の変数を含むデータフローが抽出された。そのうち前記のセキュリティ関数に関連する189個の変数について保護属性が分類され，その中で72個の変数が機密性ありと分類された。

図4は，DFITS$_{TM}$プロトタイプシステムが出力した保護属性付きデータフローの一部である。図4から，セキュリティ関数に直接入出力される変数だけでなく，これらに関連する膨大な変数についても保護属性の分類ができているようすがわかる。また，AACSの仕様書で指定されている機密データを格納する変数が，DFITS$_{TM}$プロトタイプシステムの出力結果において機密性及び完全性あり（confidential属性）と正しく分類されていることも確認できた。

一方，DFITS$_{TM}$プロトタイプシステムにより保護属性が分類されなかった148個の変数については，解析の結果，その変数に関連するセキュリティ関数の制約条件が存在しなかったことがわかった。

4.3 評価

DFITS$_{TM}$プロトタイプシステムの実証実験の結果から，DFITS$_{TM}$を用いることで，ソフトウェア開発者は機密データを格納する変数の洗出し作業を効率化できるとともに，分類ミスを防止できることが示された。また，図4に示すような保護属性付きデータフローは，データフローにおける機密データ及びそれ以外のデータの流れを“見える化”できることから，これもソフトウェアの開発に活用することが可能である。

5 あとがき

悪意のあるユーザーによる不正な解析や改ざんに対して安全なソフトウェアの開発を支援する技術DFITS$_{TM}$について述べた。機密データの洗出しにDFITS$_{TM}$を用い，機密データ保護にLMSP$_{TM}$などのソフトウェア保護技術を用いることで，ソフトウェアの安全性とソフトウェア開発効率の向上が期待できる。

今後は，DFITS$_{TM}$の解析能力向上及び適用範囲拡大とともに，データフロー解析手法の形式的評価についても検討を進めていく。

文 献

(1) 橋本幹生 他．オープンソースOSと共存可能なセキュリティプロセッサ技術．東芝レビュー．**60**，6，2005，p.44－47.
(2) 橋本幹生 他．オープンソフトウェアにおけるソフトウェア保護．東芝レビュー．**58**，6，2003．p.20－23.
(3) 林良太郎 他．“セキュリティに注目したデータフロー解析技術DFITSのデジタル著作権管理システムへの適用”．2011年暗号と情報セキュリティシンポジウム（SCIS）．小倉，2011-01，電子情報通信学会 情報セキュリティ研究専門委員会．4E2-1．（CD-ROM）．
(4) McPeak, S. ; Necula, G. C. "Elkhound: A Fast, Practical GLR Parser Generator". 13th International Conference on Compiler Construction, Duesterwald, E. Barcelona, Spain, 2004-03, Technical University of Catalonia. Berlin, Springer-Verlag, p.73－88.

林 良太郎 HAYASHI Ryotaro, D.Sci.
研究開発センター コンピュータアーキテクチャ・セキュリティラボラトリー研究主務，理博。ソフトウェア保護技術及び暗号技術に関する研究・開発に従事。電子情報通信学会会員。
Computer Architecture & Security Systems Lab.

中西 福友 NAKANISHI Fukutomo
研究開発センター コンピュータアーキテクチャ・セキュリティラボラトリー。ソフトウェア保護技術に関する研究・開発に従事。
Computer Architecture & Security Systems Lab.

橋本 幹生 HASHIMOTO Mikio
研究開発センター コンピュータアーキテクチャ・セキュリティラボラトリー主任研究員。ソフトウェア保護技術に関する研究・開発に従事。電子情報通信学会，情報処理学会，ACM会員。
Computer Architecture & Security Systems Lab.