ソフトウェア工学の基礎XI，日本ソフトウェア科学会FOSE2004，
pp.157-160，November 2004.

マルチバージョン生成によるプログラムの解析防止

Protecting Software by Generating Multi Version Modules

山内 寛己*　神崎 雄一郎†　門田 暁人‡　中村 匡秀§　松本 健一¶

Summary. In order to obfuscate a program, we proposes a method for generating multi-version programs $P_1, ..., P_k$, from a given program P_0, with following properies; (1) $P_1, ..., P_k$ have different specifications, (2) Majority of outputs of $P_1, ..., P_k$ is identical to P_0's for an arbitrary input.

1　はじめに

今日，ソフトウェアに含まれる秘匿情報 (アルゴリズム，サブルーチン，定数，条件分岐など) を守る技術の必要性が高まっている．例えば，ディジタルデータの著作権管理を行うソフトウェアは，内部に含まれるデバイス鍵 (定数値) とその処理ルーチンを隠蔽することが必須となる [1].

従来，それらの秘匿情報を隠すために，ソフトウェアの難読化方法が数多く提案されてきた [5]．難読化とは，与えられたプログラムを，その仕様を変えずに，解析 (理解) のより困難なプログラムへと変換する技術であり，変数名や関数名などの変換，データの変換 [3]，制御フローの変換 [4] [7] [8]，自己書き換え処理の追加 [6] 等の様々な方法が存在する．しかし，仕様が保存される以上，十分な時間と労力を費やして解析することで，難読化されたプログラムから秘匿情報を取得できる恐れがある．

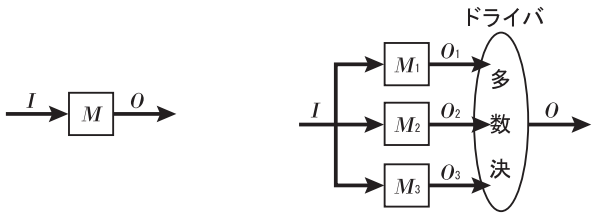
本稿では，難読化の一手法として，ソフトウェアの仕様を変化させる方法を提案する．提案方法では，仕様を少しずつ変化させた複数のバージョンのソフトウェアを生成し，それらの実行結果の多数決により正しい (本来の仕様の) 結果を得ることを保証する．攻撃者が各バージョンのソフトウェアを解析したとしても，本来の仕様と異なる実装になっているため，正しい仕様を理解することは困難である．この方法は，耐故障性の向上を目的としたマルチバージョンソフトウェア [2] の手法の応用であるが，提案方法は，1 個のソフトウェアを複製して得られる複数のバージョンに対し，互いに異なるバグを意図的に挿入する (仕様を変化させる)，いわば耐故障性を逆手にとった方法である．

2　マルチバージョン生成によるプログラム仕様の隠蔽

2.1　キーアイデア

提案方法は，プログラムの一部 (モジュール) を**マルチバージョン化**することによって，攻撃者による解析を困難にする．マルチバージョン化とは，単一のモジュー

*Hiroki Yamauchi, 奈良先端科学技術大学院大学 情報科学研究科
†Yuichiro Kanzaki, 奈良先端科学技術大学院大学 情報科学研究科
‡Akito Monden, 奈良先端科学技術大学院大学 情報科学研究科
§Masahide Nakamura, 奈良先端科学技術大学院大学 情報科学研究科
¶Ken-ichi Matsumoto, 奈良先端科学技術大学院大学 情報科学研究科



(a) 元のモジュール　(b) マルチバージョン化されたモジュール

図 1　モジュールのマルチバージョン化

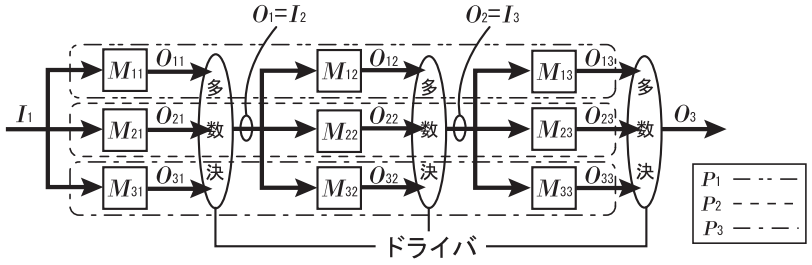


図 2　マルチバージョンソフトウェアの構成

ルから仕様の互いに異なる複数のモジュール (**バージョン**) を生成することである (図 1)．図 1 は，(a) 入力 I に対し出力 O を計算するモジュール M と，(b)M をマルチバージョン化した M_1,M_2,M_3 の例を示す．各バージョン M_1,M_2,M_3 は，次の性質 (1)(2)(3) を満たす．(1)M,M_1,M_2,M_3 は互いに異なる仕様を持つ，(2) 任意の入力 I に対し，出力 O_1,O_2,O_3 の最大多数となった出力は O と一致する，(3)M_1,M_2,M_3 は難読化されており，仕様の差分を抽出することは困難である．

エンドユーザには，M_1,M_2,M_3 と多数決を取るためのドライバを配布し，元のモジュール M は配布しない．そのため，M の仕様を知ることはエンドユーザにとって困難となる．プログラム内の多数のモジュールをマルチバージョン化することで，解析をより困難にできる．

2.2　マルチバージョン化の手順

プログラム P をマルチバージョン化する手順を以下に示す．

1. プログラム P を複数のモジュール $M_1, M_2, \ldots, M_i$ に分割する．
2. 複数のモジュールに分割したプログラム P に，k 個のバージョン (k は奇数) を複製 (コピー) して，プログラム $P_1, P_2, \ldots, P_k$ を生成する．バージョン x のモジュール i を M_{xi} と記す．
3. 複製されたすべてのバージョンについて，元のモジュールと仕様が異なるように内容を書き換える．書き換え方法は 3.1 で述べる．
4. 各プログラム $P_1, P_2, \ldots, P_k$ を互いに異なる難読化手法で難読化をする．これにより，各バージョンのモジュール $M_{1i}, M_{2i}, \ldots, M_{ki}$ を比較して仕様の差分を解析することが困難となる．

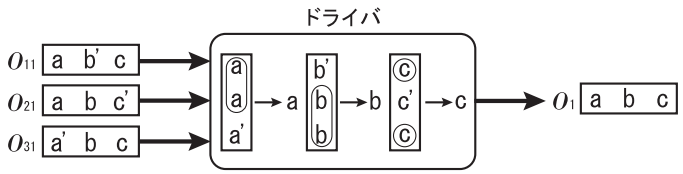
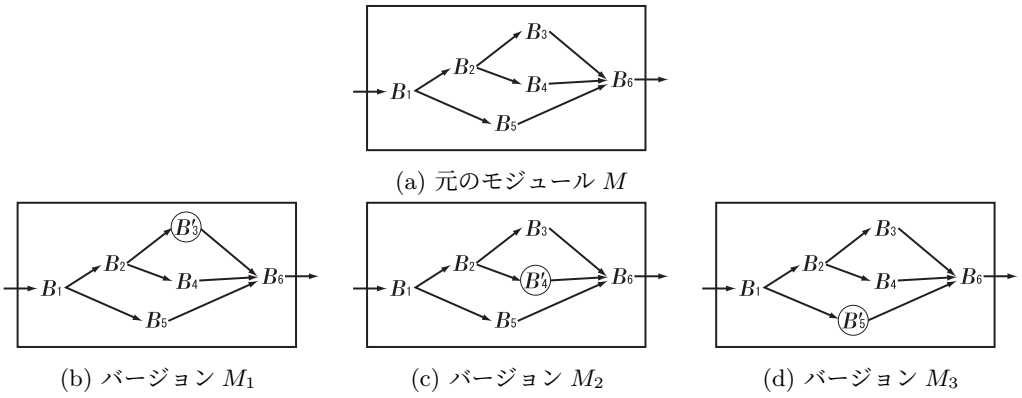


図 3　多数決による出力の制御



(a) 元のモジュール M

(b) バージョン M_1　(c) バージョン M_2　(d) バージョン M_3

図 4　バージョンの生成例

図 2 はプログラム P をモジュール 3 つに分割 (M_1, M_2, M_3) し，さらにプログラム P を 3 つに複製 (P_1, P_2, P_3) した例である．

2.3　多数決による出力の制御

モジュールの出力は多数決によって制御する．図 3 は例を示したものである．元のモジュールと同じ入力 I_1 を，実行中のマルチバージョン化したプログラムのモジュール M_{11}, M_{21}, M_{31} に入力すると，各モジュールの仕様に基づき，各バージョンの出力 O_{11}, O_{21}, O_{31} を得る．

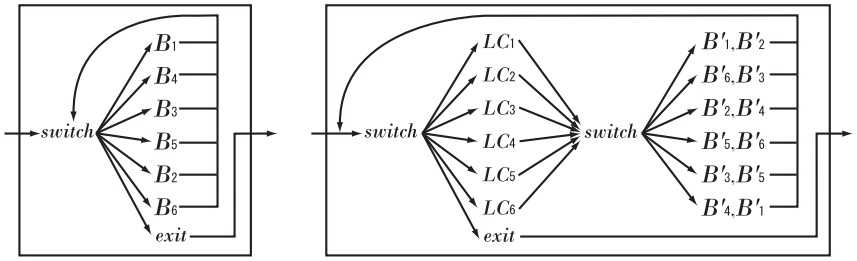
一般に，入力と出力は型付きの値の集合で与えられる．ここで，入力 I_1 に対する元のモジュール M_1 の出力を $O_1 = \{a, b, c\}$ とし，各バージョンの出力 $O_{11} = \{a, b', c\}, O_{21} = \{a, b, c'\}, O_{31} = \{a', b, c\}$ が得られたとする．ドライバでは，出力 O_{11}, O_{12}, O_{13} を要素ごとに多数決をとることで，$O_1 = \{a, b, c\}$ を得ることができる．

3　バージョンの生成例

3.1　仕様の変更例

与えられたモジュールから互いに異なる仕様を持つバージョン群を生成する方法はいくつか考えられるが，ここではごく単純な一例を示す．図 4(a) のフローチャート（基本ブロック $B_1, ..., B_6$ をノードとみなした制御フローグラフ）で示されるモジュール M について考える．

まず，M をコピーしてバージョン $M_1, ..., M_3$ を生成する．次に，M_1 内のブロック B_3 の仕様を変更し，さらに，M_2 内の B_4，M_3 内の B_5 を変更する．$M, M_1, ..., M_3$



(a) 条件分岐の難読化　　(b) (a) に制御フローの難読化を適用したモジュール

図 5　制御フローが難読化されたモジュール

が互いに異なる仕様を持ち，かつ，$M_1, ..., M_3$ の出力の多数決が必ず M の出力と一致することに注目されたい．

3.2　モジュールの難読化の例

攻撃者は，モジュール間の基本ブロックの対応関係を調べ，差分を抽出することで，正しくないブロック（前節の B'_3, B'_4, B'_5）を特定できる可能性がある．そこで，バージョン間の基本ブロックの対応付けを困難にする方法として，制御フローの難読化を行う例を示す．図 5(a) は，Wang らの方法によって制御フローを平坦化した例である [8]．図 5(b) は，Chow らの方法によって基本ブロックの結合と変数の割り付けの変更を行った例である [4]．各バージョンについて互いに異なる難読化を施すことで，仕様の差分の抽出を困難にできる．

4　おわりに

本稿では，ソフトウェアを不正な解析行為から保護することを目的として，プログラムをマルチバージョン化する方法を提案した．今後は，マルチバージョン化における仕様変更のアルゴリズムについて，さらに検討していく予定である．

参考文献

[1] 4C Entity: Content protection for recordable media specification - Introduction and common cryptographic elements, rev. 1.0, 31 pp., Jan. 2003.
[2] A.Avizienis and L. Chen: On the implementation of N-version programming for software fault tolerance during programming execution, Proc. COMPSAC '77, pp.149-155, 1977.
[3] S. Chow, P. Eisen, H. Johnson and P.C. van Oorschot: A white-box DES implementation for DRM applications, Proc. 2nd ACM Workshop on Digital Rights Management, pp.1-15, Nov. 2002.
[4] S. Chow, Y. Gu, H. Johnson, and V. Zakharov: An approach to the obfuscation of control-flow of sequential computer programs, Information Security Conference (ISC2001), Lecture Notes in Computer Science, Vol.2000, pp.144-155, 2001.
[5] C. Collberg and C. Thomborson: Watermarking, tamper-proofing, and obfuscation – tools for software protection, IEEE Trans. on Software Engineering, vol.28, no.8, pp.735-746, June, 2002.
[6] 神崎雄一郎, 門田暁人, 中村匡秀, 松本健一: 命令のカムフラージュによるソフトウェア保護方法, 電子情報通信学会論文誌, vol.J87-A, no.6, pp.755-767, June 2004.
[7] 門田暁人, 高田義広, 鳥居宏次: ループを含むプログラムを難読化する方法の提案, 電子情報通信学会論文誌, vol.J80-D-I, no.7, pp.644-652, July 1997.
[8] C. Wang, J. Davidson, J. Hill, J. Knight: Protection of software-based survivability mechanisms, Proc. Int'l Conf. on Dependable Systems and Networks, Goteborg, Sweden, July 2001.